Universidad
Carlos III de Madrid

TESIS DOCTORAL

LOCAL USER MAPPING VIA MULTI-MODAL FUSION FOR SOCIAL ROBOTS

Autor:
Arnaud Ramey
Directores:
Miguel A. Salichs Sánchez-Caballero
María de los Ángeles Malfaz Vázquez

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA
Leganés, June 1, 2015

TESIS DOCTORAL (DOCTORAL THESIS)

LOCAL USER MAPPING VIA MULTI-MODAL FUSION FOR SOCIAL ROBOTS

Autor (Candidate): Arnaud Ramey

Director (Adviser): Miguel A. Salichs Sánchez-Caballero
Directora (Adviser): María de los Ángeles Malfaz Vázquez

Tribunal (Review Committee)

Firma (Signature)

Presidente (Chair): _____

Vocal (Member): _____

Secretario (Secretary): _____

Título (Degree): Doctorado en Ingeniería Eléctrica, Electrónica y Automática
Calificación (Grade): _____

Leganés, de de

To my loved ones

“It is by looking and seeing that we come to know what is where in the world.”

— David Marr

Acknowledgments

Being a PhD student is no easy task, and I can testify that by myself I would never have reached the end. The list of people that made this PhD dissertation possible is long [1] and I owe all of you many thanks.

First and before all, thanks to the *Direction générale de l'armement* [2] for giving me the great opportunity to gather expertise in the field of robotics for several years, and for supporting me. I am aware of how lucky I am, I have worked hard, and will work hard, to justify this investment.

I cannot thank enough times my adviser, Miguel Ángel: he introduced me to the wonderful team of Social Robotics of the UC3M, was a helpful, resourceful and patient leader for my work, and managed to transform my erratic research into coherence. The work of my co-adviser María cannot be stressed enough either: in spite of a packed schedule, she always managed to find time to advise, help, and motivate me. She was also the courageous proofreader of this PhD dissertation. The Social Robots team is also of key importance to this work: thanks to Alberto, for fixing all the code I broke without ever losing your smile; Fer, for your limitless dynamism and hard work; Ana, for your kindness and your talent; Alvaro, for your wicked humor and pragmatism (and forcing me into taking swimming classes!); Irene, for bringing fresh and rigorous methods into the team practices; Javi, for all these meaningful ideas, the innovative perspective you always brought and this unique taste in music; Victor, for being an unlimited source of knowledge and wisdom – and leading the ROS battle; Esther, for your artistic skills; Raul, for your resuscitating capabilities: you can turn legacy code into something working; Jose Carlos, for assuring continuity to the presenting work; and last but not least, David, for being an incredible workmate and finding a solution to all problems. Thanks to all the other members of the Robotics Lab for the happy years in Leganés, among others Juan, Jorge, Tamara, Choukri, Martin, and all the others.

More generally, thanks to the University Carlos III of Madrid for these master and PhD

[1] Way too long to fit a single page, really!

[2] the French Government Defense procurement agency responsible for the program management, development and purchase of weapon systems for the French military

programs, and the related post-graduate administration (CEAS): Ruth, Margarita and the others. Thanks to the Carlos Corre runners, who found the secret way to convert suffering into a great joy, and the swimming classes members (especially Jesús) for helping us poor students to get out of the lab and staying fit.

As popular wisdom says, the real friends are the ones that come the day you move in / out: my deepest thanks to Alex and Avinash for making these years in Madrid unique, and the ones around the world: Léo, Jérémie, Johan, Baptiste, Florian, Benjamin... All my love to the Sambanés players for rocking my life. Thanks to my family, for being there whenever I needed you.

Ainara, there is no words to express how you enlightened the way.

And many thanks to the many others that helped at one or another point of the road.

Merci à tous.

Abstract

User detection, recognition and tracking is at the heart of Human Robot Interaction, and yet, to date, no universal robust method exists for being aware of the people in a robot's surroundings. The presented work aims at importing into existing social robotics platforms different techniques, some of them classical, and other novel, for detecting, recognizing and tracking human users. These algorithms are based on a variety of sensors, mainly cameras and depth imaging devices, but also lasers and microphones. The results of these parallel algorithms are then merged so as to obtain a modular, expandable and fast architecture. This results in a local user mapping thanks to multi-modal fusion.

Thanks to this user awareness architecture, user detection, recognition and tracking capabilities can be easily and quickly given to any robot by re-using the modules that match its sensors and its processing performance. The architecture provides all the relevant information about the users around the robot, that can then be used for end-user applications that adapt their behavior to the users around the robot. The variety of social robots in which the architecture has been successfully implemented includes a car-like mobile robot, an articulated flower and a humanoid assistance robot.

Some modules of the architecture are very lightweight but have a low reliability, others need more CPU but the associated confidence is higher. All configurations of modules are possible, and fit the range of possible robotics hardware configurations. All the modules are independent and highly configurable, therefore no code needs to be developed for building a new configuration, the user only writes a ROS *launch* file. This simple text file contains all wanted modules.

The architecture has been developed with modularity and speed in mind. It is based on the Robot Operating System (ROS) architecture, a *de facto* software standard in robotics. The different people detectors comply with a common interface called `PeoplePoseList` Publisher, while the people recognition algorithms comply with an interface called `PeoplePoseList` Matcher. The fusion of all these different modules is based on Unscented Kalman Filter techniques. Extensive benchmarks of the sub-components and of the whole architecture, using both academic datasets and data acquired in our lab, and end-user application samples demonstrate the validity and interest of all levels of the architecture.

Resumen

La detección, el reconocimiento y el seguimiento de los usuarios es un problema clave para la Interacción Humano-Robot. Sin embargo, al día de hoy, no existe ningún método robusto universal para para lograr que un robot sea consciente de la gente que le rodea. Esta tesis tiene como objetivo implementar, dentro de robots sociales, varias técnicas, algunas clásicas, otras novedosas, para detectar, reconocer y seguir a los usuarios humanos. Estos algoritmos se basan en sensores muy variados, principalmente cámaras y fuentes de imágenes de profundidad, aunque también en láseres y micrófonos. Los resultados parciales, suministrados por estos algoritmos corriendo en paralelo, luego son mezcladas usando técnicas probabilisticas para obtener una arquitectura modular, extensible y rápida. Esto resulta en un mapa local de los usuarios, obtenido por técnicas de fusión de datos.

Gracias a esta arquitectura, las habilidades de detección, reconocimiento y seguimiento de los usuarios podrían ser integradas fácil y rápidamente dentro de un nuevo robot, re-usando los módulos que corresponden a sus sensores y el rendimiento de su procesador. La arquitectura suministra todos los datos utiles sobre los usuarios en el alrededor del robot y se puede usar por aplicaciones de más alto nivel en nuestros robots sociales de manera que el robot adapte su funcionamiento a las personas que le rodean. Los robots sociales en los cuales la arquitectura se pudo importar con éxito son: un robot en forma de coche, una flor articulada, y un robot humanoide asistencial.

Algunos módulos de la arquitectura son muy ligeros pero con una fiabilidad baja, mientras otros requieren mas CPU pero son más fiables. Todas las configuraciones de los módulos son posibles y se ajustan a las diferentes configuraciones hardware que puede tener el robot. Los módulos son independientes entre ellos y altamente configurables, por lo que no hay que desarrollar código para una nueva configuración. El usuario sólo tiene que escribir un fichero *launch* de ROS. Este sencillo fichero de texto contiene todos los módulos que se quieren lanzar.

Esta arquitectura se desarrolló teniendo en mente que fuese modular y rápida. Se basa en la arquitectura Robot Operating System (ROS), un estándar software *de facto* en la robótica. Todos

los detectores de personas tienen una interfaz común llamada `PeoplePoseList` Publisher, mientras los algoritmos de reconocimiento siguen una interfaz llamada `PeoplePoseList` Matcher. La fusión de todos estos módulos se basa en técnicas de filtros de Kalman no lineares (Unscented Kalman Filters). Se han realizado pruebas exhaustivas de precisión y de velocidad de cada componente y de la arquitectura completa (realizadas sobre ambos bases de datos académicas además de sobre datos grabados en nuestro laboratorio), así como prototipos sencillos de aplicaciones finales. Así se comprueba la validez y el interés de la arquitectura a todos los niveles.

Résumé

La détection, la reconnaissance et le suivi des utilisateurs est un problème fondamental pour l'Interaction Homme Robot, et pourtant, il n'existe actuellement pas de méthode universelle et robuste qui permette à un robot de percevoir les personnes autour de lui. Le travail présenté ici vise à importer dans différentes plateformes existantes de robotique sociale des techniques pour détecter, reconnaître et suivre des utilisateurs humains, certaines classiques, d'autres innovantes. Ces algorithmes se basent sur des capteurs variés, principalement des caméras et des périphériques permettant d'estimer la profondeur, mais aussi des lasers et des microphones. Ces algorithmes s'exécutent en parallèle, et leurs résultats sont ensuite fusionnés, ce qui donne une architecture modulaire, extensible et rapide. Ceci produit une cartographie locale des utilisateurs, basée sur de la fusion multi-modale.

Grâce à cette architecture de perception des utilisateurs, l'on peut facilement et rapidement permettre à n'importe quel robot de détecter, reconnaître et suivre ses utilisateurs, en déployant les modules qui correspondent à ses capteurs et ses capacités de calcul. L'architecture fournit toute les informations pertinentes sur les utilisateurs autour du robot. Elles peuvent alors être utilisées pour des applications de plus haut niveau, qui pourront adapter leur fonctionnement selon le comportement de ces utilisateurs. La gamme de robots sur lesquels l'architecture a été déployée inclut un mini-véhicule, une fleur articulée et un robot humanoïde d'assistance.

Certains modules de l'architecture sont très rapides mais peu fiables, d'autres sont plus gourmands en puissance de calcul mais ont une meilleure fiabilité. Toutes les configurations de modules sont possibles, ce qui permet de s'adapter à de nombreuses configurations de plateformes robotiques. Chaque module est indépendant et hautement configurable, ainsi l'on n'a pas besoin d'écrire de code pour créer une nouvelle configuration, mais seulement un fichier texte, appelé fichier *launch* de ROS. Ce simple fichier texte liste tous les modules désirés.

Notre architecture est pensée pour offrir une grandes modularité et une importante vitesse d'exécution. Elle se base sur l'architecture Robot Operating System (ROS), un standard logiciel *de facto* dans le monde de la robotique. Tous les algorithmes de détection de personnes se conforment à une interface commune, appelée `PeoplePoseList` ***Publisher***, tandis que les

algorithmes de reconnaissance de personnes utilisent une interface appelée `PeoplePoseList Matcher`. La fusion de tous ces différents modules se base sur des filtres de Kalman (*Unscented Kalman Filter*). Des évaluations exhaustives de toutes les sous-composantes et de l'architecture complète, grâce à des bases de données académiques d'une part, et des données acquises par nos soins d'autre part, ainsi que des exemples d'application de haut niveau, démontrent la validité et l'utilité de notre architecture à tous les niveaux.

Contents

List of Tables

List of Figures

List of Algorithms

Code listings

CHAPTER 1

Introduction, problem definition and goals

A ROBOT is, according to the definition of the Oxford dictionary, *a machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer*. The frontier between an automated machine and a robot is not clear, but the latter is traditionally characterized by a degree of intelligence and understanding of its surroundings in its decision making, while the former repeats pre-written sequences of actions. A robot can then be seen as an entity which has, to some extent, autonomy in its behavior and the actions it performs. As such, **Robotics**, the field of science and technology that aims at designing, building and operating robots, is a field at the convergence of mathematics, computer science, physics, mechanics, philosophy and many other fields.

Social robotics is a field of robotics that aim at making robots daily companions, that interact with human users, for helping and entertaining them in their everyday life, and follow social behaviors and rules. It spans over a wide range of applications and types of users. Examples include helping children with their homework, taking care of elderly people, giving information and advice to people in public places, etc. The relation between the human user and the robot can be on a short term, say, a robot indicating directions in a shopping mall ([Kanda et al., 2009]), or long term, for instance, a robot delivering mail and food to employees in a lab on a daily basis over a period of several months ([Lee et al., 2009]).

These examples suggest how the interaction flow between the human user and its robotic companion, called **Human-Robot Interaction** (HRI), is at the core of social robotics: a social robot aims at helping its users and as such, it needs to grow an **User awareness**. This consists of the robot having the knowledge of how many users are around it, where they are, who they

are, and what they are doing.

Our goal in this PhD is to give this ability to social robots. This skill is, to date, a challenging problem in robotics.

Subdividing the problem: how our brain tackles the issue. Paradoxically, user awareness can appear very natural and trivial to achieve for us humans. Indeed, the human brain is a highly sophisticated and specialized machinery which dedicates an important part of its activity for this very same task. The retina of the eye is made of 150 millions cones and rods, which allow us to perceive that many colors at the same time. About 30% of the human brain activity is due to the analysis of the information given by the vision system, while the touch processing represents about 8%, and the sound 3%. This would be the equivalent of a 150 MP digital camera. Our brain constantly hunts for visible human faces, and more generally, human bodies in its surroundings, thanks to its vision system. The task of counting the number of people in an image is carried out by the human brain in only 100 ms. People counting is boosted by the so called *subitizing*, which is the ability of the brain to immediately know how many objects are present in the scene without counting them, if they are not too numerous (termed coined by [Kaufman and Lord, 1949]).

It is interesting to look at how user awareness is performed in the human brain. It seems the brain splits this difficult tasks into independent sub-tasks of a smaller complexity, that are easier to achieve (*divide and conquer* strategy).

Some patients who suffered different types of accidents, such as seizures or car accidents, had some very specific parts of their brain damaged, leaving the rest intact. This helps understanding the way our brain works, and especially where the different functions are located and how the whole system is articulated. For instance, patients of the so-called *prosopagnosia*, also called face-blindness, cannot recognize the identity of known faces while rest of the brain functionality is intact, such as object recognition or even face detection [Grossman et al., 2000, Damasio et al., 1982, Yin, 1970]. It seems that the way people awareness is achieved in the brain can be divided in three modules: *people detection*, which consists of locating people around us thanks to the instantaneous data stream of our sensory system; *people recognition*, dealing with knowing who they are; *people tracking and mapping*, which is a higher-level understanding their motion about keeping a spatial and temporal coherency in this perception of the people.

1.1 Potential benefit of the PhD

Interaction with their human users is the key function of social robots. Yet, the data their sensors give is of high dimension and complexity, and almost deprived of any semantic meaning: a scan given by a laser range finder is made of hundreds of numerical values (corresponding to distances to objects), or an image from the camera of millions of integers, but this plentiful of data is deprived of meaning for end-user application. The distance of the closest object at a given angle, or the exact color of the pixel at given coordinates, do not really matter, but the

presence of a user in that laser scan or camera image does.

This is where our system is needed: end-user applications need both semantically rich and compact representations of the world to focus on the Human-Robot Interaction (HRI) flow. In other words, the work presented in this PhD dissertation can be seen as a reduction of data size from a verbose sensors input stream to a brief, semantically rich textual description of the different users around the robot.

In addition, some other meaningful processing blocks can then be located between our user awareness system and the end-user application, such as human action recognition (understanding what the users are doing) or easier recognition of the emotional state of the users, through the analysis of their face for instance. These blocks depend on the user awareness system as input, and provide some higher-level information as output.

We have to take into account that giving awareness to social robots is a challenging problem, that has been tackled already by numerous authors and with a wide range of sensors and techniques: many algorithms already exist that solve part of the problem: user detection, user tracking, user recognition, etc. However, none of the existing integrated architectures propose an easy integration of these different modules, or reconfiguration of the different modules used.

1.2 Goals of the PhD

As previously stated, the goal of the work presented in this PhD dissertation is **to give user awareness to social robots**. This main goal can be split into a set of sub-goals.

- **Comprehensive review of the existing algorithms** for both user detection and recognition. Select the ones which are the most appropriate for the needs and capabilities of our robots hardware and software.
- **Design of a lightweight common data structure supplied by all user detectors and used by all recognizers.** While encapsulating an amount of data that must be as small as possible, it must at the same time be complete enough to give a real user-awareness to the robot. This includes, for each detected person, her identity, her accurate position, and some other characteristics about her. The use of this common data structure confers modularity of the user awareness architecture: because all modules comply with this interface, it is easy to add or remove some of them according to the robot specifications.
- **Integration of the most relevant detection and recognition algorithms** into the robots software architecture, using the common data structure mentioned above. The different robots used, later described, have very different shape, hardware and functionality, which represent a good testbed for the modularity of the proposed architecture.
- **Fusion of these detection and recognition algorithms into a user mapping module**. We will use the results of all these algorithms to obtain consistent information of where the

users spatially are and who they are, on a map in a static frame. This includes the spatial tracking of each user.

- **Experimental benchmarking of each component** for verifying their usefulness and accuracy. This also includes lowering as much as possible their computation requirements.

- Finally, **design of one or two end-user applications of the system**, which will make use of the user detection and recognition system seen above. It can be for instance a game, etc. Building advanced applications is not the scope of this thesis, these end-user applications are proofs of concept: we will demonstrate that the robot can perceive who the users are, and personalize its interaction.

These goals structure the work that has been carried out during the PhD. However, they make no sense if no robot, i.e., no hardware platform, is available for implementing and testing them. The available resources will now be presented, along with the constraints they present.

1.3 Resources and constraints

The target platforms for the work presented in this PhD are social robots. They are complex systems, and integrate a wide range of processing blocks. For instance, these include drivers for all the physical devices, low-level motion capabilities, higher-level motion planning and localization, a voice/non verbal sound architecture, etc. As such, our system must be compatible with the highly complex architecture of these robots. The implications are twofold.

First, the software architecture of our system must be compatible with the existing processing blocks. The robots use a particular Operating System at a particular version, a set of particular drivers, and a particular software architecture. Our software architecture needs to be compatible with this set of software constraints.

Second, the central processing unit (CPU) of these robots are limited. As such, the CPU needs of our system cannot be too high: this would make the proper parallel execution of all blocks difficult, and makes the robot less reactive. Furthermore, the CPUs vary between different platforms: mobile robots will often sport an embedded computer with a limited CPU, while tabletop robots, with an electric supply by cable, can embed more powerful CPUs.

Robots used in this PhD dissertation will now be briefly presented.

1.3.1 Specifications of the robots

Two robots of the RoboticsLab of the University Carlos III of Madrid are extensively used as experimental platforms for the different works presented in this PhD: *Maggie* and *MOPI*.

1.3.1.i The robot Maggie

Maggie is one of the research robots at the Robotics Lab. It is an experimental platform for studying Human-Robot Interaction (HRI). An illustration of Maggie is below, on Figure 1.1. Here is a summary of the extensive description of Maggie in [Barber and Salichs, 2002].

Figure 1.1: *The robot Maggie.*

Maggie is designed as a 1.35 meters-tall girl-like doll. Its base is motorized by two differentially actuated wheels and a caster wheel on both sides. It is also equipped with 12 bumpers, 12 infrared optical sensors and 12 ultrasound sensors. Above the base, a laser range finder (Sick LMS 200) has been added. A Red Green Blue Depth (RGBD) camera (Microsoft Kinect) is located next to this device.

The upper part of the robot incorporates the interaction modalities. Two arms, one at each side of the belly, have one degree of liberty each. A *tablet PC* (Acer Travel Mate 370) on its belly allows user interaction through touch inputs, and information display. On top of the body comes an anthropomorphic robot head with an attractive design. The head has two degrees of liberty. The mouth also embeds a hidden webcam. Invisible touch sensors are integrated in several parts of the body, such as the shoulders, the hands and the head.

Maggie relies on a main computer hidden inside its chest which controls all its skills. It is a quad-core i5-3550 CPU @ 3.30GHz. For image acquisition, the camera (hidden in the mouth), is a Logitech QuickCam Pro 9000, with a VGA resolution (640$\times$480 pixels) and a frame rate of 30 frames per second. The previously mentioned Microsft Kinect is mounted at the belt, enabling acquisition of synchronized RGB and depth images with also a VGA resolution and 30 frames per second. The integration of this device on Maggie is presented in [Ramey et al., 2011].

All these features are illustrated in Figure 1.2.

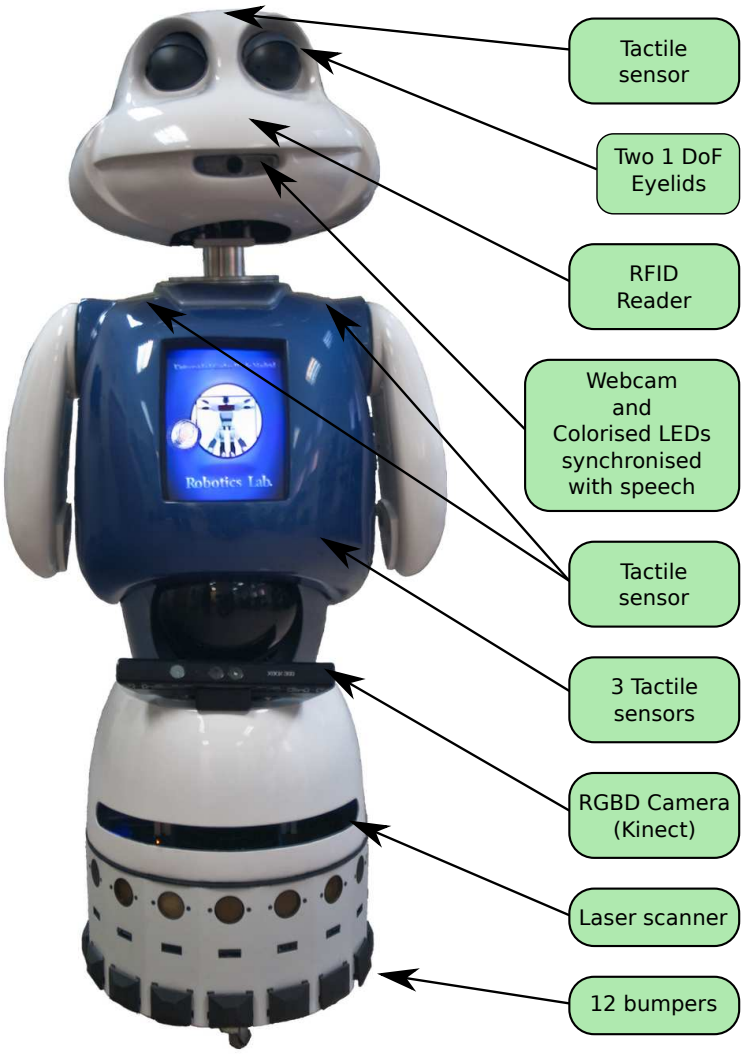


Figure 1.2: *The hardware equipping Maggie.*

1.3.1.ii The robot MOPI

The modular robot **MOPI** is aimed at working within an architecture of modular robots. It is a home-brew robot of the RoboticsLab, shaped as a mobile, car-like platform, and is visible in Figure 1.3.

The base is moved with a differential drive thanks to two gear motors, each one in charge of moving the two wheels of one side. It is powered by an embedded Intel Atom @ 1.60 Ghz. Note that this embedded CPU is significantly more limited than the one in Maggie, which justifies the required modularity of the system. MOPI uses a Li-Ion Battery, and communicates via a 802.11n WIFI connection. Environment sensing is made by a (non tiltable) Hokuyo laser scanning range

finder, and a Microsoft Kinect device tilted at 45 degrees. This enables seeing the entire bodies of the surrounding human users up to 4 meters away.

The whole configuration gives the robot great possibilities for navigation and environment mapping, while its limited skills in HRI are balanced by additional robots that can be placed on top of it. However, two LCD screens of 3.2 inches can display the robot's eyes, used for expressing the robot inner state.

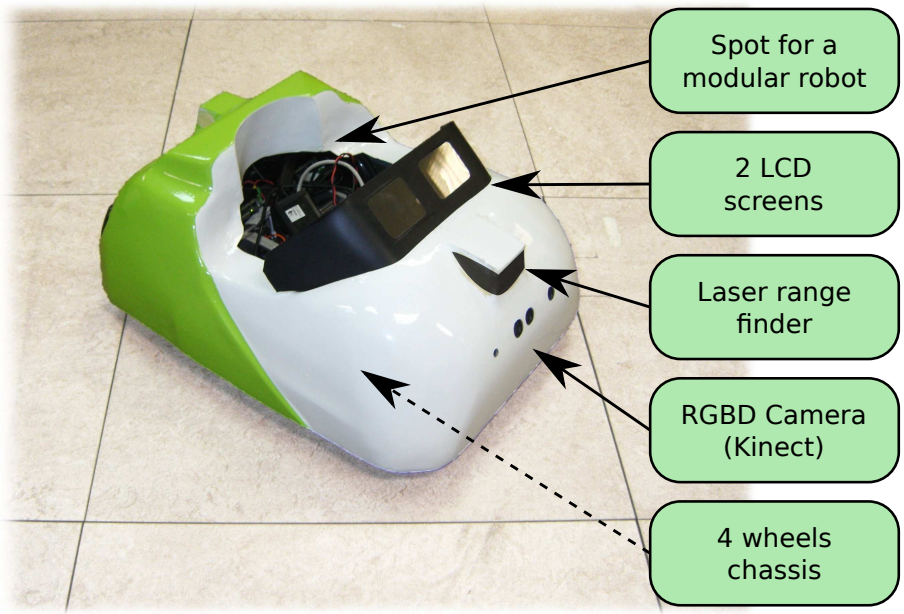


Figure 1.3: *The hardware equipping MOPI.*

1.4 Methodology

In this part, I will explain how the problem of giving user awareness to the social robots of the RoboticsLab was tackled.

We previously saw how user awareness could be split in fairly three modules, inspired by the way the brain performs the task: people detection, people recognition, and people mapping.

We will tackle separately the three sub-problems to achieve the goal of this PhD, i.e. giving user awareness to robots: detecting the visible users around the robot; recognizing them; and building a consistent representation of this knowledge on a map (*mapping*). Such a subdivision is conceptual, as all subproblems need to be solved simultaneously when the robot interacts with users.

To make things clearer, let us consider the fictional case of Figure 1.4. We can see that the

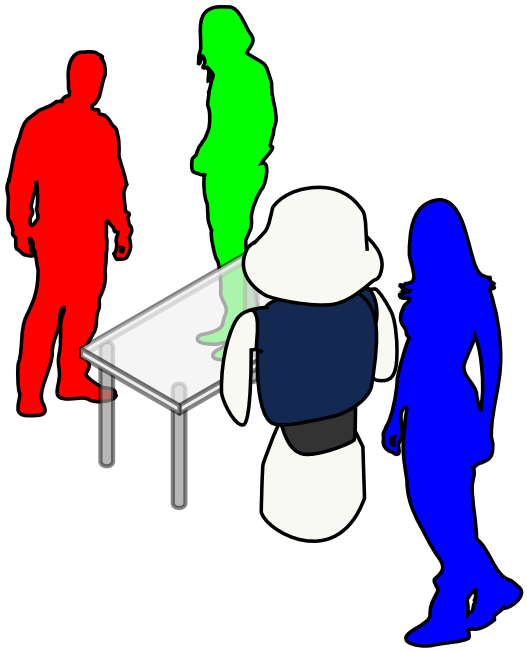


Figure 1.4: *A fictional situation of user awareness around a robot.*

robot, here drawn as the blue and white shape in the middle [1], is surrounded by three users. **User detection**, for the robot, consists of detecting in the current data input these three users around it and knowing where they are. This knowledge could be represented in plain words from the robot perspective as shown in Code listing 1.1.

Code listing 1.1: *People detection representation from the robot perspective*

```
There are three users around me.
* Visible user #1 is standing still in front of me, looking at me.
* Visible user #2 is standing to my right, looking towards my left.
* Visible user #3 is moving behind me.
```

User recognition consists of matching each of these temporary IDs to a permanent identity. In other words, it consists of obtaining a long-term coherency with the way it detects the users. Using the previous example, this knowledge could be represented as in Code listing 1.2.

Code listing 1.2: *People recognition representation from the robot perspective*

```
* Visible user #1 is most likely Bob.
* Visible user #2 is maybe Lea, but most likely I have never seen her
  ↪ .
* Visible user #3 is most likely Lea.
```

[1] It is a representation of the robot Maggie, presented in subsubsection 1.3.1.i, page 5.

All this knowledge about the surrounding users can be represented in a visual way, in a way similar to how humans visualize their surrounding people. **User mapping** consists in representing these user detections into a consistent structure, such as a map in a static frame of coordinates. It also includes being aware of the user behind it, even though this user is not visible to the different sensors of the robot [2]. See Figure 1.5 for an example of this representation.

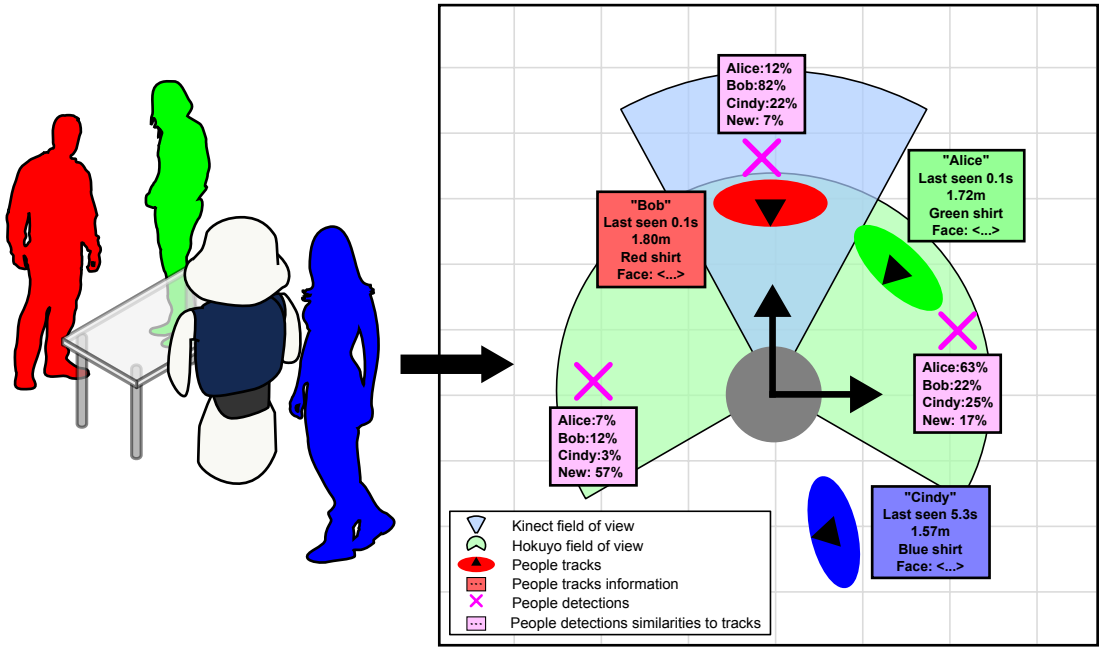


Figure 1.5: *Knowledge representation of the user awareness in the fictional situation of Figure 1.4. The fields of view of the different sensors are drawn as angular ranges.*
The purple crosses correspond to ***user detections****, using algorithms such as face detections.*
The different users perceived by the architecture are represented as tracks and drawn as ellipses. The knowledge about each user is in the colored frame box next to the ellipse.
User recognition *is used to match detections and tracks: we can compute the similarities between detections and tracks (the higher the similarity, the more probable the detection corresponds to the track).*
Data fusion *helps updating these detections into the matching tracks, to generate* ***user mapping****.*

1.4.1 Software architecture

1.4.1.i ROS: the Robot Operating System

The Robot Operating System (ROS) [Quigley et al., 2009] [3] is a software architecture specifically developed for using with robots. The ROS version used in the robots is ROS *Groovy* running on

[2] Of course, some sensor has to be able to detect the presence of a person at some point for the robot to be aware of that person at a later instance.

[3] `http://www.ros.org/`

top of Ubuntu 12.04. ROS offers different advantages: modularity of the processes, distributed architecture, many-to-many and many-to-one communication mechanisms, and great stability. Let us describe briefly the ROS architecture.

A ROS architecture is made of independent processes that run in parallel, called *nodes*. A special node, called *master*, is in charge of maintaining the list of active nodes.

Messages Exchange of data between these different nodes is made easy, thanks to the use of *messages*. Messages are data structures that can include primitive types (`bool`, `int`, `string`, etc.), arrays of these primitive types, and other messages. For instance, the `LaserScan` message, used by laser drivers, is defined as following [4]:

```
float32 angle_min
float32 angle_max
float32 angle_increment
float32[] ranges
```

Note how the `LaserScan` message is a combination of the primitive types.

Transfer of these messages from one node to another is made via a so-called *publisher/subscriber* paradigm. Each data channel is identified by a channel name called *topic*, which is a simple string, for instance `"/data"`. If a node wants to make some data available to others, it will first *advertise* the topic, in other words, notify the master of the existence of the topic and the type of message that will be exchanged on this topic. Every time it has data that can be shared, it can *publish* a message containing this data, i.e. make the data public for all nodes interested in this data.
Now, if another node is interested in obtaining some data from a topic, it will first *subscribe* to the topic, i.e. notify the master that it is interested in obtaining the messages made available on the topic, along with the callback function that should be called upon receiving a message on that topic. For these reasons, the message mechanism follows a *many to many* paradigm: any node that publishes or subscribes to a given topic is connected to the others.

Let us give an example to clarify these concepts. Let us call N_1 a first node that would be a driver for a laser range finder. It communicates with the physical device and obtains the different laser scans. N_1 advertises a topic called `"/laser_scan"`, of type `LaserScan`.
Now, let us call a second node N_2 which will detect obstacles thanks to the laser data. N_2 subscribes to the `"/laser_scan"` topic. A peer-to-peer connection is then created between N_1 and N_2. Every time N_1 publishes a laser scan, this scan is being serialized in N_1, send via the ROS communication layer to N_2, and deserialized in N_2, where the callback function associated with this topic is finally called with the deserialized data. If N_1 or N_2 is stopped or crashes, this connection is silently and cleanly closed.

[4] It is actually a simplified version, the curious reader can see the full message at `http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html`

The use of ROS most notably gives the possibility to redistribute the computational workload. Different machines can be connected to the same master, and the messages will be transferred between nodes running on both machines via WiFi or LAN connection, without having to perform any change: the topic name is a unique ID for the data channel. For example, we can send easily via the wire or a wireless connection raw data from the sensors to remote computers for processing. The latter can be more powerful than the embedded computers integrated within the robot, thus lightening the computation workload of the main computer. Then, the processed data, such as odometry, is sent back to the robot. Using the previous laser example, N_1 has to be running on the robot computer (access to the physical device), but N_2 can be running on a desktop computer.

Note that the serialization and deserialization of a message to a string equivalent for its transmission is automatically made by ROS. The different messages are in fact defined in a simple description language called `msg`, and follows the distributed object paradigm [5]. Users can define their own messages using the primitives types and other messages. During compilation time, ROS will transform `msg` files into equivalent headers in several programming language (C++, Python, Perl to date, more to be integrated). Any node written in one of these languages can include these headers and publish this kind of messages. Furthermore, the nodes do not have to be written in the same programming language: the laser driver can be written in C++ and the laser processing node in Python, for instance. As such, the definition of data classes does not depend on the programming language, it is called *language abstraction*.

Services There is another communication mechanism than messages: *services*. Services are identified by a unique string name, like the topic name, and a pair of of strictly typed messages: the request and the response. Just like web-services, service correspond to a *many-to-one* paradigm: they can be called by various nodes but can be offered by at most one. They correspond to a remote function call, where the arguments are in the request, and the response contains the results of the computation by the node offering the service.

An example of service could be a voice generation node. Many nodes may need to generate voice (games, dialog managers, etc.). However, only one node can make the proper sound generation: two voices must not be heard at the same time. As such, a fictional `/GenerateVoice` service would have the following definition:

```
string sentence
---
bool was_said
```

Note that the dashed line divides the definition in two parts: the first part corresponds to the request and is filled by the calling node before calling the service, the second to the response and is filled by the voice generation node. In that case, the success or failure of the voice generation is indicated by a simple boolean value.

[5] `https://en.wikipedia.org/wiki/Distributed_object`

Params Finally, a last feature of ROS is *params*. They are numerical parameters that can be accessed and changed in real-time. This gives the possibility of configuring easily the behavior of a node without modifying its source code. These parameters can be set from any node, or through command line arguments.

For instance, the previously seen voice generation node could subscribe periodically to the param `/RobotLanguage` that would set the language spoken by the robot, and make on-the-fly translation of the sentences if needed [6].

ROS summary To sum up the ROS landscape, processes are called *nodes*. They are independent and run in parallel. They can exchange data via messages sent on *topics*. A function of a given node can be called from another node thanks to a *service*. Finally, the behavior of each node can be dynamically tuned thanks to *params*.

1.4.1.ii The Automatic-Deliberative (AD) architecture

The software of Maggie and MOPI, the previously presented robots of the RoboticsLab, work according to the Automatic-Deliberative (AD) paradigm, as presented in [Barber and Salichs, 2002]. This paradigm handles skills relying on primitives. Primitives are in direct communication with the physical devices of the robot, and send elementary orders to them. This includes the base motors, the laser sensor, the camera, etc. A skill is the ability of the robot to do a specific action. It relies on the data supplied by the primitives. The actions generated by a skill can be numerous: move the car to a given point, play games with the user, interact with electric appliances, etc.

Since they are experimental platforms, the robots have also been used since 2010 as a bridge between the traditional implementation of AD, as seen in [Rivas, 2007], and a new one relying on the communication mechanisms of ROS.

1.5 Structure of the PhD

The following PhD dissertation will be structured as follows. First, in **chapter 2, page 15**, the existing architectures for user detection, tracking and recognition will be reviewed, along with their relative advantages and limitations.

Part I, page 31: The first part of this dissertation will focus on user detection. **chapter 3, page 31** will focus more on techniques based on vision and image-processing, **chapter 4, page 67** on other fields, such as laser scan processing and audio analysis. In the former chapter, we will also present the lightweight data structured mentioned in the goals, called `PeoplePoseList` (`PPL`). Some of the vision techniques are classical vision techniques that

[6] This was actually implemented in one of the robots of the RoboticsLab, in [Alonso-Martin et al., 2011].

we imported and adapted to PPL, such as Viola-Jones face detector or the Polar-Perspective Map (PPM)-based detector.

Part II, page 89: the second part, made of **chapter 5, page 89** and **chapter 6, page 129,** will deal with user recognition techniques. The same structure is adopted: the first chapter is about vision-based recognition techniques, the other about other techniques, such as voice recognition.

Part III, page 139: All the algorithms presented in the two first parts give hints about how many users are around the robot, where they are and who they are. However, these hints can be combined to obtain more robust estimators: this point will be covered in the third part, made of the **chapter 7, page 139**. Tracking and fusion techniques such as Kalman filters and its extensions for non-linear problems will be presented and used for merging the different user detection and recognition algorithms seen in the previous parts. On top of that, an innovative technique using tracking of blobs in the depth image thanks to 2D image processing techniques will be presented.

Part IV, page 197: different applications, taking benefit of the user awareness architecture, are detailed in the final part of this PhD Dissertation. After a brief state of the art in **chapter 8, page 197**, a surveillance algorithm using our architecture for the robot MOPI will be introduced in section 8.2, page 203 and some games between the robot Maggie and users will be presented in section 8.3, page 215. These applications will make great use of the PPL-based user awareness system (made of detection, recognition and mapping) and show its modularity by adapting its structure to the robots hardware and software capabilities.

Finally, the main contributions of this PhD, along with conclusions and future works, will be presented in **chapter 9, page 223**.

CHAPTER 2

Related work

As presented in the Introduction chapter, the aim of the work presented in this PhD dissertation is to give user awareness to a social robot, in other words, to give it the ability to detect and recognize the users evolving around it.

In this chapter, we will review a wide range of social robots and the methods they use to acquire awareness of the users around them, developed by other labs and companies around the world. Some of the presented platforms are more than a decade old, but we will focus our attention on the most recent social robots. Note that this chapter does not cover technical solutions to detect and track humans: each of the following chapters will include a short state of the art concerning the field it focuses on.

2.1 Social robots without any user awareness mechanism

The uses of social robots are more and more numerous, from both academic labs and industries. Contrary to industrial robots, social robots can help all kinds of users, and especially people that have no technical knowledge of how they work. They have two main types of audiences: children and elderly people. For children, robots can both help them in their education and entertain them. For elderly people, social robots aim at taking care of them and ensuring they remain healthy. These applications can only be made possible if the robot is aware of the user position and engagement. And yet, very surprisingly, many social robots are actually incapable of detecting the users around them. These robots will be covered in this section.

For instance, *"Keecker"* [1] is a mobile robot launched in 2014 that can move through the house for entertaining users thanks to its embedded projector and loudspeaker. Pictures of the robot are visible in Figure 2.1. The control of the robot is made through a mobile phone, and although it can move autonomously through the house, users are merely detected as obstacles for navigation. They are not recognized whatsoever, and their preferences are known from the mobile phone remote application. The embedded camera is used for remote control and surveillance only. This lack of user awareness is somewhat paradoxical for a personal device.

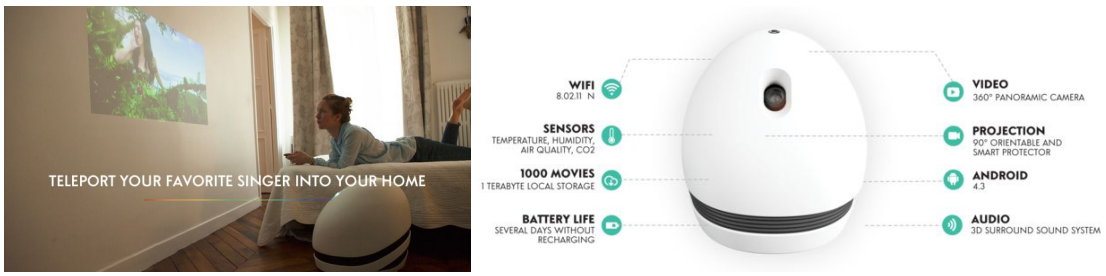


Figure 2.1: *The social robot "Keecker". The embedded projector can display a movie on the wall, while the integrated loudspeakers renders the sound. Images from* `http://www.keecker.com/`.

The *"Echo"* device [2] proposed by the company Amazon in 2014 is the combination of a loudspeaker and a microphone aimed at integrating into homes for helping the family. It is built around its speech recognition capacities: the users questions and orders are used to trigger actions or get information, as the device is permanently connected to the cloud. The robot is visible in Figure 2.2. As far as we can judge from the information given by the company, Echo is made for speech recognition, but there is no perception whatsoever of the users around it: the speaker is not recognized, and anybody can take control of it. Similarly, the customization of the interface is provided by the services configured inside the robot (personal account, webservices, etc.) but not by the detection of the position of the user or similar techniques.

These two examples of social robots made by companies focus on entertainment and even without user awareness, they can perform their task. More surprisingly, many academic articles focusing on Human-Robot Interaction (HRI) itself do not include any user detection or recognition mechanism. Indeed, the robot is often remotely controlled by a human operator. This technique, known as "Wizard of Oz", enables to trick the user into believing the robot is autonomous. By these means, the scientists ensure that no algorithmic failure flaws the experiments. This is for instance the case of the *"iCat"* robot playing chess against children in [Castellano et al., 2009]. This is visible in Figure 2.3. The robot assumes the user is in front of it, and its emotions are computed by the state of the game only.

Similarly, *"DragonBot"*, the dragon robot released in 2011 from MIT Social Robotics lab is aimed at being a platform for cloud-based social robotics. A picture is in Figure 2.4. All the processing is done on-line, the embedded phone CPU and sensors being only in charge

[1] `http://www.keecker.com/`
[2] `http://www.amazon.com/oc/echo`

Figure 2.2: *The social robot "Echo". A user case is also visible: note the robot lying on the table and the users interacting with it. Images from* `http://www.amazon.com/echo`.

Figure 2.3: *The social robot "iCat".*

of acquiring information and moving effectors. DragonBot is thought to be a platform for experimenting HRI in the wild. However, all the motion is remote-controlled: an operator chooses the gestures and behavior of the robot remotely.

2.2 Detecting users as obstacles

We saw in the previous section how some social robots do not detect explicitly at all who the users around them are. In this section, we will see some social robots that do perceive the users around them, but merely as geometrical areas where the robot should not go or move, i.e. as non-free zones. In this category fall many navigation robots from the previous decades.

For instance, *"RHINO"* ([Burgard et al., 1998]) is one of the earliest autonomous navigation and guide in museums, dating back from 1998. For navigation, there is no awareness of the users. They are modeled as mere obstacles on a dynamic occupancy map. The museum robot Minerva ([Thrun et al., 1999]) deepens the HRI of this robot, as it has an internal emotional state.

Figure 2.4: *The social robot "DragonBot". Images from* `http://www.adamsetapen.com/`.

Comparing the dynamic map with a static map of the environment, we can know if the path is blocked by users. According to its emotional state, the robot asks the users to make way in a more or less polite way. However, there is still no awareness of the individual users.

Another system where user detection is at stake is autonomous vehicles: for the safety of pedestrians, their detection is of paramount importance so that no harm is done to them. See for instance the pedestrian detection system by [Howard and Matthies, 2007]. [Austin and Kouzoubov, 2002] also tackles the challenge of safe navigation and aims at detecting humans for not putting them at stake.

Industrial robots Contrary to social robots, industrial robots have been used for several decades. However, they perform repetitive actions that can be precisely described. They often have an important strength to perform the industrial tasks they are in charge (bend metallic parts, saw, drill, melt iron, etc). For these reasons, important safety measures exist and the huge majority of industrial robots does not collaborate with humans or is aware of its environment. Important safety perimeters or safety cages prevent accidents with human co-workers. However, latest advances in this field see the distance between human workers and robots decrease. For instance, the latest product of the Swedish robotics society ABB, *"YuMi"* [3] is a dual-arm manipulator robot designed for collaborative work with its human co-workers. A picture is in Figure 2.5. Safety is at the core of the robot, so that it can work with humans without posing any risk. However, the human perception seems to be assured by force-sensing motors that can detect if the motion does not happen as expected.

[3] `http://www.abb.com/yumi`

Figure 2.5: *The collaborative industrial robot "YuMi". Images from* `http://new.abb.com/products/robotics/yumi`.

2.3 Short term awareness

Whereas in the previous section, we saw some social robots that could perceive users as mere obstacles and constraints for their motion, in this section we will present prototypes of robots that can be aware of the users around them, but only in a short term way. In other words, they can know when a user wants to interact with them and behave in an adapted way. However, if that same user happens to interact again later on with the robot, it will not take into account their previous encounter.

2.3.1 User awareness through explicit actions

A simple method for detecting users willing to interact with the robot is that they declare their presence in an explicit way.

For instance, the social robot *"RHINO"* previously presented considered the users as obstacles. On top of that, Human-Robot Interaction (HRI) is made through an on-board interface, made of four colored buttons, a screen and loudspeakers. In other words, the user demonstrates its presence by pressing the buttons. This is visible in Figure 2.6.

The robot dog *"Aibo"*, released by Sony in 1999 and discontinued in 2006, was equipped with a variety of sensors and buttons on the body of the dog. Pictures of Aibo are visible in Figure 2.7. These buttons are pressed by the user when he wants to reward or punish the dog. That way, when one of these buttons is pressed, the robot knows the user has pressed them and can behave accordingly.

One of the first robot receptionists was called *"Valerie"* and was used in Carnegie Mellon university, visible in Figure 2.8 ([Gockley et al., 2005]). Valerie is involved in long-term interaction as it stayed during several months in a booth at the entrance of offices. Although the behavior of the robot is mostly identical every day, the authors have found that many visitors continue to

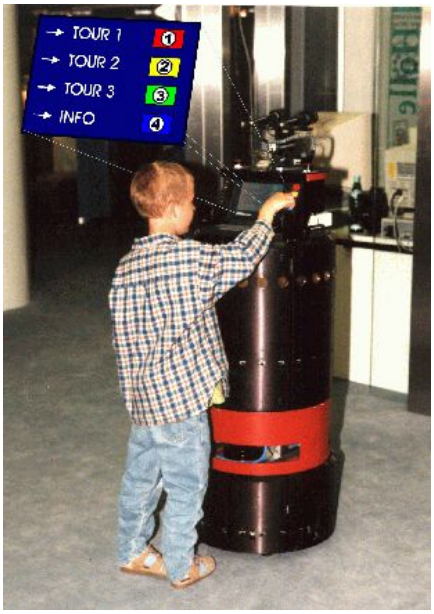


Figure 2.6: *The social robot "RHINO" and its interface.*

Figure 2.7: *The first version of the social robot "Aibo" released in 1999, chasing a pink ball.*

have short interactions with the robot on a daily basis during several months. Valerie can detect users thanks to laser range finders or a keyboard. However, the user awareness is made by the user entering what she wants to say on the keyboard in front of the robot: user awareness is obtained thanks to this action of the user.

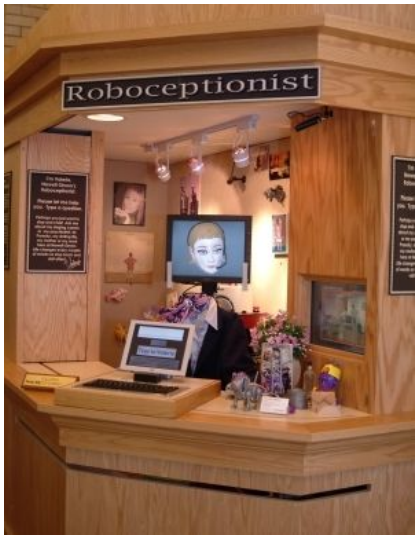


Figure 2.8: *The social robot "Valerie" in her booth.*

On our own social robot *"Maggie"*, presented in the Introduction chapter and visible in Figure 1.2, page 6, several capacity sensors have been embedded in the body of the robot. That way, when users touch the body of Maggie, a signal is triggered that can be handled by the architecture of the robot. In the several games that were implemented to allow a rich interaction with children, ([Gonzalez-Pacheco et al., 2011]), we came to the conclusion that these simple sensors were especially interesting as they were much more reliable than any algorithmic detection mechanism, that were always subject to faults. On the other hand, the interaction with this kind of device can be seen as not very natural, as it does not really correspond to the way a user would correspond with a fellow.

2.3.2 Autonomous user detection

This subsection focuses on robots that detect users automatically without needing them to do any explicit action.

The social robot *"Kismet"*, developed in the 90s by the MIT and visible in Figure 2.9, can perform a closed-loop active vision thanks to face and eye detections. Audio attention is also supplied via an embedded microphone that performs speech processing and recognition. There is no long term coherency or memory of the user, the robot plays with whoever is in front of it.

[Kollar et al., 2012] presents a robot receptionist, coined *"Roboceptionist"*, that helps users finding their way in offices. The interaction is short term: the users ask a few questions to the robot then go. The users are detected thanks to proximity and gestures: a short term perception of the user is built, and the robot rotates and looks toward this detected user. There is no long term recognition of the users. This is visible in Figure 2.10.

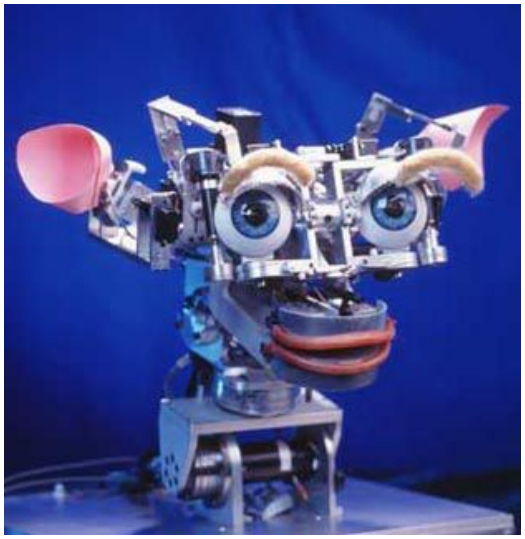


Figure 2.9: *The social robot "Kismet".*

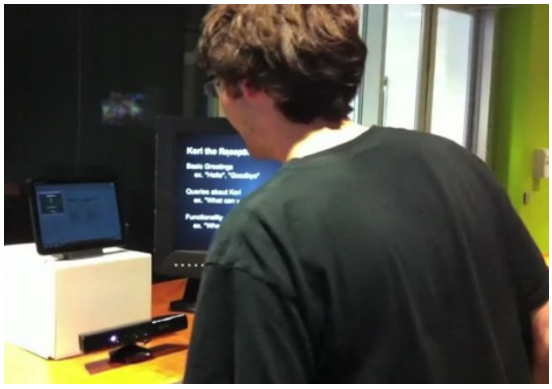


Figure 2.10: *The social robot "Roboceptionist".*

In [Geiger et al., 2013], the social robot *"ALIAS"* is presented as a gaming platform for elderly persons. This is visible in Figure 2.11. In this article, the only game presented is Tic-Tac-Toe, played through the means of a touchscreen located between the robot and the user. The user is detected thanks to voice detection and a face detection algorithm, but no recognition whatsoever is performed. The Human-Robot Interaction (HRI) is made through the use of the tablet computer on the chest of the robot. The authors stress the interest of such a platform for aging populations, but the lack of long-term user awareness limits it.

In [Gross et al., 2009], the robot TOOMAS helps customer in a home improvement store. This is visible in Figure 2.12. The potential users are found thanks to different vision approaches, namely a skin color detector, a motion detector and a face detector, and a local occupancy map-based approach (thanks to the use of a map made via a Simultaneous Localization and Mapping (SLAM) algorithm and a laser range finder). Thus, the robot is aware of the users

Figure 2.11: *The social robot "Alias".*

locally present around it. However, the main detector being the frontal face detector, the robot has trouble localizing potential users if they do not face it.

Figure 2.12: *The social robot "TOOMAS".*

2.3.3 Intelligent environments

Some articles focusing on social robotics propose a user awareness mechanism based on a modification of the environment: by equipping it with several additional sensors, extra information can be obtained.

For instance, in [Satake et al., 2009] a shopping mall is equipped with several Laser Range Finders (LRFs) mounted on the walls of the mall to generate a map of the users. Algorithms can detect the speed of the visitors and their behavior (in a rush or wandering). The social robot, in

charge of advertising shops, uses this information to approach humans in a relevant way: it will locate itself properly and break a conversation only if the visitor is wandering.

In the work presented in this PhD, we are not interested in the approach of intelligent environments: our social robots aim at being able to be used in any setup, including homes, hospitals, schools, etc., which becomes difficult if the environment needs to be prepared.

2.4 Long term awareness

The previous section presented robots that could detect and interact with users on a short term basis: if the same user happened to come back later, the robot would not remember their former interaction. However, it is because we are able to identify individuals that we can develop a unique relationship with each of them. Nonetheless, it is difficult for a robot to recognize the people around it with auditory sensors: many people can talk at once, the background noise and the noise of the robot body distort the voices, etc. The visual input is challenging too: the environment can be unstructured, the shapes and colors of the objects might be to complex for algorithms to understand them without error.

ID cards For this reason, some Human-Robot Interaction (HRI) authors focus their long term awareness on person identification (ID) cards.

In the social robot *"Valerie"*, presented previously ([Gockley et al., 2005]), user recognition is made possible thanks to the use of a magnetic card reader: the users are unambiguously identified by swiping their ID card. User recognition is possible by swiping a magnetic card-stripe ID card, but the authors point that the implemented scenario offers little benefit to recognition except being greeted by name. Regardless, some of these users identifying themselves , called repeat visitors, spent more time and with more frequency with the robot, justifying the interest of user recognition.

A similar solution is used in [Kanda et al., 2004]. The authors study the evolution of the relationship over time in an 18-day field trial between 119 first- and sixth-grade pupils and a humanoid robot. This is visible in Figure 2.13. They chose to perform user recognition using wireless Radio-Frequency Identification (RFID) identification tags and sensors. Nameplates with embedded wireless tags were given to all children. The range of the RFID reader can be changed online, and using zones of proximity, the robot can detect the participant and the observers. The use of RFID tags guarantees the precision of the detection; but it is unable to handle a multi-participant dialogue, and requires to wear the tags.

[Lee et al., 2012] also stresses how personalization is of paramount importance for long-term HRI. The study is made through *"Snackbot"* ([Lee et al., 2009]), an autonomous robot in charge of delivering snacks and other products inside a university building, during an experiment that lasted several months. This is visible in Figure 2.14. The authors conclude that the relationship between the user and the robot is reinforced when the robot adapts its behavior to her. In these

Figure 2.13: *The social robot "Robovie" interacting with Japanese pupils.*

examples user detection is at stake in such behaviors for the safety and acceptability of the robot in a human environment.

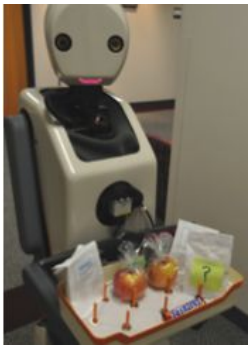


Figure 2.14: *The social robot "Snackbot".*

Snackbot has a long term user awareness. However, this awareness is obtained thanks to the snack order system and the structured environment: user preferences and profile is known through the web interface used to order, and the office of each worker being known, the robot assumes the person detected in the office of the delivery is the expected user. Note that the robot was operated using a Wizard of Oz technique, so that the personalization could be manually fitted.

The same user awareness mechanism, user awareness through gathering of information with a web-service, is found in the PR2 robot, turned into a service robot, in [Garage, 2010]. The beverage orders are made through a web-interface, and the user indicates her position. However, there is no real perception of the user: the beverage is delivered to the first face detected in the surroundings of the position indicated by the user.

Automatic user recognition We already saw some companies that sell small robot companions, that will assist the users in their everyday life for a range of tasks, such as the Keecker or the

Echo. *"Jibo"* [4] is a robot aimed at integrating into the daily life of families, that can learn, teach, entertain and help, and that had a launching campaign in 2014. Pictures are in Figure 2.15. Jibo recognizes the users around it thanks to its vision system (face detection and recognition) and microphones (speech recognition). This robot being a connected device, it gathers a lot of information about the users thanks to the applications and media she consumes, so that the interaction is personalized. However, this product not being released yet, details about the user awareness system are not known.

Figure 2.15: *The social robot "Jibo". A user case is visible on the right, where Jibo is helping a user with her hands busy thanks to speech recognition. Images from* `http://www.myjibo.com/`.

[4] `http://www.myjibo.com/`

Summary of the chapter

We have seen in this chapter a wide range of social robots and reviewed how they acquire user awareness. Some robots are barely aware of their environment, similarly to mechanical puppets, while other perceive and recognize their users. In other words, there are different levels of user awareness.

At the lowest level, some social robots have no perception at all of the users around them. They are merely obeying direct orders, for instance teleoperation orders. This is the case of the Amazon Echo device, or HRI robots in Wizard of Oz mode.

Some autonomous robots perceive the users as mere obstacles. While this ensures the safety of the users while moving, it does not give any satisfying social behavior in the presence of users. This is the case of some industrial robots or some older HRI platforms.

Some social robots offer a short term awareness of the user. In other words, they can detect and interact the users around them, but will not remember the previous interactions with a given user. The users can notify of their presence thanks to explicit actions, such as buttons pushing. This is the case of some HRI robots from previous decades. The robots can also detect the users automatically, thanks to Laser Range Finders or face detections with cameras. They can detect and track the users, as long as they stay in the vicinity of the robot. Most modern social robots work this way.

There is a scarcity of robots that have a long term user awareness. Indeed, the few prototypes that exist use Wizard of Oz techniques. Another technique is to use ID cards, such as RFID or visual tags, that are accurate, but clumsy for a natural HRI.

As a conclusion, each existing user awareness system is tailored for the robot where it is used. There is no generic user awareness system that can adapt to the variety of hardware configurations and sensors found in social robots. In the remainder of this PhD dissertation, we will present our solution for this problem, a generic, multimodal user awareness architecture, divided in three parts: user detection; user recognition; and data fusion and user mapping.

Part I

User detection

CHAPTER 3

Vision-based person detection

Introduction

The goal of the work presented in this PhD dissertation is to give user awareness to social robots. In other words, these robots must be able to detect if there are users in their surroundings (*detection*), and in that case, who they are (*recognition*) and they are (*tracking / mapping*). This problem can be divided into two subproblems: on the one hand, user detection, which consists of detecting the users close to the robot thanks to the sensors equipping it, on the other hand, user recognition, which deals with recognizing these users against a set of already known users.

This chapter will tackle the problem of user detection using knowledge and algorithms coming from the field of image processing and computer vision [1]. **Image processing** refers to the set of techniques and algorithms that extracts knowledge thanks to the analysis of an image, whereas **Computer vision** is a higher-level field whose goal is to extract knowledge from the analysis of images of the real world around the device. These images typically are supplied via a camera stream, i.e., a continuous flow of instantaneous images. Image processing is a process that maps an image to knowledge, whereas computer vision maps the world to knowledge through images.

[1] Note that following chapter 4, page 67 will focus on the same problem, but using different techniques from image processing.

Detecting human shapes in an image stream is a problem that is related to many more fields than just social robotics: may it be for home security ([Jung et al., 2012]), playing video games ([Berliner and Hendel, 2007]), for monitoring the activity of sick people ([Kepski and Kwolek, 2012]), etc. Even in robotics, the applications are multiple: they include gesture recognition ([Ramey et al., 2011]), follow-the-leader behaviors ([Bellotto and Hu, 2009, Ramey et al., 2013]), turning the robot into a gaming companion ([Leite et al., 2012]), and many more. The important amount of research that has been done results in a wide range of methods and algorithms that have been proven to give satisfactory results for people detection.

First, in section 3.1, page 32, the algorithms for user detection existing in the different fields mentioned above, and that are the most relevant according to our goals (which were specified in the Introduction chapter), will be reviewed. Second, our own contributions to user detection will be detailed in section 3.2, page 38. These contributions include the integration of some of the existing algorithms seen in the literature review, such as face detection with Viola-Jones detector ([Viola and Jones, 2001]), the Histogram of Oriented Gradients (HOG) detector ([Dalal and Triggs, 2005]), or the Polar-Perspective Map (PPM) detector, used in autonomous driving for pedestrian detection [Howard and Matthies, 2007]. The contributions then include improvements of the existing algorithms, most notably thanks to the joint use of color and depth images, the latter being used for false positive removal.

3.1 State of the art

As said earlier, detection of human shapes in a stream of images is a problem that has been tackled by researchers several times in the past. Note that many articles propose ad-hoc solutions: in other words, the user detection requires a non-natural modification of the environment or of the interaction flow. For instance, when the user wears a specific color of clothing, different from the background, the segmentation is made much easier. This is the case in [Mikić et al., 2003] where the actors wear a red suit. In the challenging case of locating an operator by an underwater robot, ad-hoc visual markers were used in [Dudek et al., 2007] (more about these so-called fiducial markers will be explained in the next chapter).

In this section, we discard this category and focus on generic (non ad-hoc) detection algorithms. Among the many existing algorithms, a limited number of them will be reviewed. These are the ones that are at the same time the most accurate and the most relevant to the goals we defined earlier in section 1.2, page 3.

The first two classes of algorithms are two-dimensional (2D): they are based on the analysis of the color image, without any depth information. Namely, they focus on face detection (subsection 3.1.1) and a technique called *Histogram of Oriented Gradients*, based on the analysis of the variations of the contrast (subsection 3.1.2).

Then, two relevant algorithms using the depth information will be introduced. First, the middleware performing the detections of the players for the Microsoft Kinect device, called *NiTE*, will be presented in subsection 3.1.3. Then, another algorithm called *Polar-Perspective Map*,

originally using stereo vision and made for pedestrian detection in autonomous vehicles, will be introduced in subsection 3.1.4.

3.1.1 Face detection

Neuroscientists have put in evidence that our brain is highly specialized in recognizing faces. An important part of the cerebral activity is dedicated to spotting faces in the flow of data supplied by the vision system. Actually, it is so active that very simple sets of shapes, a few circles and lines, will make us see faces in those shapes, as one can observe looking at clouds, or the shape of the moon for instance. It thus makes sense to apply some similar techniques to robots to supply them with an accurate perception of the users. **Face detection** consists of determining if human faces are visible in a video stream, and if it is the case, what are their position in the image. This is one of the classical challenges in vision.

In this part, we will explain how we robustly detect human faces in the video stream of the robot. First, faces are detected in the color video stream. The depth information is then used to discard the detections that do not correspond to a real face, called *false positives*.

Viola Jones detector. Nowadays, some more or less standard techniques are available for face detection and give a very good accuracy rate. The technique presented by Viola and Jones in several articles ([Viola and Jones, 2004]) is one of the most used in robotics and its lightweight computational cost allows a high frequency analysis of the video stream. The Viola-Jones object detection framework can actually be trained to detect any object ([Viola and Jones, 2001]), but it is especially popular for face detection.

The method of Viola and Jones is an example of supervised learning. In other words, it first requires a training phase, in which it learns features from a training set of images, some of them containing faces, while others do not. Then, in the detection phase, these features can be applied to determine if a sample image contains faces or not.

The learning is based on the appearance of the training images. The process consists of seizing the content of each image by computing so-called *characteristics* in rectangular zones of the image that overlap each other. These characteristics are a synthetic and descriptive representation of the values of the pixels, and are more efficient to be dealt with. They characterize the difference of sum of pixels of values of adjacent rectangular zones of the image. Some of them are visible in Figure 3.1. In order to be able to compute these characteristics quickly, Viola introduces the concept of *integral image*, which is an image the same dimensions than the original one, where each pixel P contains the sum of the pixels of the original image located up and left of P. The computation of a characteristic over two zones then needs at most six accesses to the integral image values, and hence a constant time.

The second key element in Viola and Jones is the use of a boosting method in order to select the best characteristics. **Boosting** is a technique that enables the building of a *strong* classifier with a linear combination of *weak* classifiers. In this method, characteristics are

seen as weak classifiers. The learning process of the weak classifier hence only consists of learning the threshold value of the characteristic so as to split better positive samples from negatives. The original detector uses three different characteristics, while the modified Lienhart and Maydt [Lienhart and Maydt, 2002] detector adds two others, and includes two diagonal orientations.

For the detection, the classified structure of the boosted classifiers enable a fast detection: Viola and Jones obtain, on a Pentium III @ 700 MHz, an average processing time of 67 milliseconds, which is considerably faster than then-available similar methods and enables real-time processing on a video feed. The strength of the detection is that wide areas of the test image, when marked as negative, are decimated in the first steps with relatively little data processing.

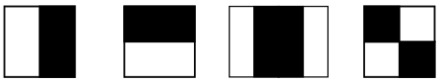


Figure 3.1: *Some of the characteristics used by Viola and Jones.*

Using neural networks Another very widespread face detection algorithm is the CMU detector presented in [Rowley et al., 1998]. It only works well with frontal, upright faces. It is based on neural networks. More detail is available in the original paper. Thanks to its accuracy, after its release in 1998, this algorithm became the most widespread and is used in numerous works of the following years (see for instance [Rehg, 1999]) till the publication of the previously presented Viola-Jones detector. The latter being more accurate and about fifteen times faster ([Viola and Jones, 2004]), it became the weapon of choice.

3.1.2 Human body detectors: Histogram of Oriented Gradients

A **Histogram of Oriented Gradients** (HOG) is a feature used in computer vision for object detection. It has turned out to be a very efficient technique for the detection of human shapes. It was presented first by Navneet Dalal et Bill Triggs [Dalal and Triggs, 2005]. Note it is a two-dimensional (2D) algorithm: it needs a Red Green Blue (RGB) image as input and returns as output the rectangular estimations of the people: it does not need, neither uses, a depth image. It makes use of histograms of *image gradients* [2].

[2] Note that we will here not define what histograms are, as they are fully defined in another chapter (chapter 5), but they can be seen as compact representations of sets of data. For a given set of numerical data, a histogram is a set of values that represent the way this data is spread. It is made of a series of so called *bins*, which can be seen as a cell contain one numerical value. Each of these cells represents a range of possible values for the data, such as the range of a cell begins where the range of the previous cell ends.

Image gradient definition The gradient of a two-dimensional function is, at each point, a 2D vector with its components equal to the derivative of this function in the horizontal and vertical directions. This definition can be extended for an image: a one-channel image can be seen as the values of a monotonous function in discrete points (the pixels). The **Image gradient** is, in each pixel, equal to the norm of the function gradient at this pixel value.

However, the image gradient cannot be directly computed because the monotonous function values are only known on discrete points. It can be approximated, though, using different one-dimensional (1D) filters, such as $[-1, 0, 1]$ (approximation of the horizontal gradient) and its transpose (vertical).

Histogram of Oriented Gradients (HOG) computation The HOG algorithm is based on the computing of local histograms of the gradient orientation, called *HOG descriptors*. The concept of the HOG descriptor is that the distribution of the gradient intensity or the direction of the edges can describe the appearance and the shape of a physical object in an image.

To compute the HOG descriptors of an image, first, the image is divided into a continuous grid of small areas called *cells*. In each cell, for each pixel of the cell, the directions of the gradients is computed. An histogram of all these directions is then built. The HOG descriptor is then made of the set of these local histograms (one histogram per cell).

Note that the histogram of a cell is normalized in contrast using the values of the image on a *block*, an area centered on the cell and larger than it. The shape of the block can be circular or rectangular, the corresponding descriptors are called respectively *C-HOG* or *R-HOG*.

Classification The HOG algorithm then needs data for supervised learning. For this purpose, a dataset of images is created, where the position of the humans is manually labeled. The HOG descriptors are computed on each image, and a simple binary SVM classifier ([Cortes and Vapnik, 1995]) is trained with them for classification. Note that the SVM classifier is a non-sophisticated classifying algorithm on purpose, so as to demonstrate the efficient description of the objects by the HOG descriptors.

Result of the original algorithm On the INRIA set, the C-HOG and R-HOG descriptors produced a *detection miss rate* of roughly 0.1 at a 10^{-4} *false positive rate*: in other words, if the algorithm is tuned to detect a non-existing human every 10000, then it only misses one person out of ten. Two samples detections on images of the RoboticsLab team are visible in Figure 3.2.

Further use The original HOG detector has been re-used and improved by several authors. Two speedups in the computation, but without considerable accuracy improvement, can be mentioned: in [Zhu and Yeh, 2006], researchers of Mitsubishi have coupled the same descriptors with cascade classifiers, similar to the ones used by Viola and seen previously in subsection 3.1.1, page 33, leading to a up to 70 times speedup, but their improvement is protected by a patent.

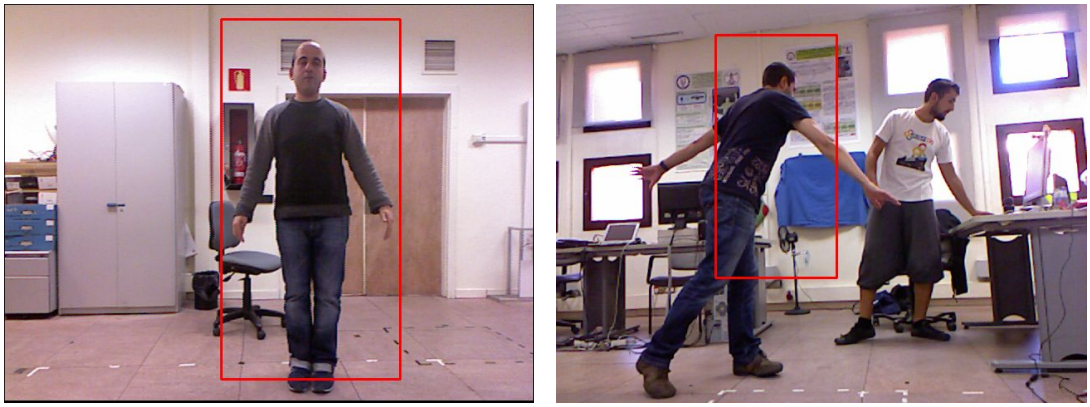


Figure 3.2: *Sample detections with the original HOG algorithm. Note the miss detection on the right: the white user was not detected.*

In [Pang et al., 2011], sub-cell interpolation computes efficiently the HOG descriptors per block, and the reuse of the features between overlapping blocks leads to a further 5 times speedup.

Note that my contribution to HOG, later explained in subsection 3.2.4, page 50, does not involve any speedup, but it improves the detection accuracy by using the depth image.

3.1.3 Kinect API: NiTE

The patented PrimeSense NiTE middleware [Berliner and Hendel, 2007] allows detecting and tracking human shapes from depth maps. It is the piece of software that powers the detection and tracking of the users for the Microsoft Kinect device, developed for the XBox 360 ([Latta and Tsunoda, 2009]).

Algorithm Although the technique employed and the source code are not available, it is likely that motion analysis and clustering techniques are at the core. Indeed, detection of a human player is activated by her motion. In other words, a still user is not detected, and a moving object will be detected as a human user if it has similar dimensions. Note that the algorithm is very lightweight, as we experimentally had it working at about 30 frames per second (FPS) even with average CPUs.

Data supplied The NiTE middleware supplies three data for higher-level applications: the Red Green Blue (RGB) image, the depth image and the multi-mask.

The **Red-Green-Blue (RGB) image** is a color image of what is visible in the field of view of the device, exactly like a classical webcam. Note that some devices only supply a greyscale version of this image. The default frame rate of the Kinect is 30 fps and resolution is 640×480 pixels. The color depth is 24 bits (B-G-R channels, one byte per channel).

The **Depth image** is an image of the same resolution as the RGB image. It only has one channel, but the values of this channel are floating point numbers, unlike the RGB image, which has three channels of bytes (integers between 0 and 255). More accurately, the value at a pixel p is the distance in millimeters between the optical center of the Microsoft Kinect device and the closest object being physically at pixel p. In other words, for each pixel (x, y) in the depth map, the pixel value at (x, y) corresponds to the distance of the closest object intersecting the three-dimensional (3D) ray from the optical center of the camera and passing by this pixel, in millimeters. The range depends on the device. For the Microsoft Kinect, they are typically within one to ten meters. However, the constructed light patterns projected by the device used to measure the distance to the objects can be reflected, for instance by shiny surfaces. The depth map contains some undefined values at those pixels, represented by `NaN` . The acceptable range of the Microsoft Kinect for the depth image is typically within one to ten meters.

Finally, the **Multi-mask** is a one-channel byte image. This image, synchronized with the RGB and the depth ones, indicates where the users are: if a pixel p of the users muti-mask has a value of 0, it means there is no user in p, while if has a value of 1, p corresponds to a pixel of the user 1, etc. For a given user, a **User mask** is the multi-mask image where all pixels that do not belong to this user are set to 0 (i.e. "erasing" the other users).

A sample of triplet (rgb, depth, user multi-mask) given by the NiTE middleware is visible in Figure 3.3.

(a) **(b)** **(c)**

Figure 3.3: *Sample images generated with the NiTE middleware.*
(a): the RGB image;
(b): the depth image, remapped to grayscale;
(c): the users multi-mask. Each value is indicated by a different color. In this sample, there are two users. Note that the segmentation is not perfect, as the leg of one user is attributed to the other one.

NiTE middleware limitations As said before, the users need to move to start their detection and tracking. The NiTE middleware is aimed at playing video games on the Microsoft XBox 360, and as human players are bound to move their body to play, the need for motion is not a problem in this context. For Human-Robot Interaction (HRI) though, detecting a still, motionless audience

becomes challenging. This becomes an issue of paramount importance if the user stands still in front of the robot while speaking with him.

Furthermore, the segmentation can fail. For instance, in Figure 3.3 (c), the leg of one user is erroneously labeled as belonging to the other user, as it must appear geometrically closer to the latter than to the former. Similarly, when users stand close to walls, pieces of wall are frequently added to their user mask.

Finally, another kind of encounter error is called *ID swaps*: when two walking users cross in front of the Kinect (thus the user in front occludes the user in the back), NiTE will, every now and then, erroneously exchange their IDs, i.e., recognize the first user as the second one.

In conclusion, the NiTE middleware is a lightweight and fast piece of software that supplies a fairly robust detection and tracking of the users, but that is only appropriate for a static camera configuration and requires the users to move to detect them. Nevertheless, it will be used in subsection 3.2.5, page 53 as one of the people detectors.

3.1.4 Polar-Perspective Map (PPM)

Another approach for people detection thanks to depth images is presented in [Howard and Matthies, 2007], where it is part of a complete pedestrian detection system only based on stereo-vision.

The authors use a two-dimensional (2D) occupancy map with a polar resolution, called Polar-Perspective Map and abbreviated to **PPM**. The three-dimensional (3D) point cloud of the environment is projected into the PPM: for every point of the cloud, the corresponding cell in the map is incremented by one. The clusters of the map with a high accumulation factor are produced by vertical objects in the original point cloud. They correspond to regions of interest: they are segmented from the original image.
Then a simple Bayes classifier determines if the 2D shape of each cluster is similar to a human shape. The system is compatible with classical image processing techniques, such as appearance-based algorithms.

Stereo-vision is, according to authors, a very liable solution for navigation and pedestrian detection. However, their system is designed for stereo cameras mounted on outdoors vehicles, with ranges between 5 and 50 meters. It makes it difficult to use for indoor robotics platforms. We can mention that a very similar system was presented almost at the same time in [Gavrila and Munder, 2006]. Note that the users detection based on PPM will be further discussed and tested in our implementation of it and benchmarking, in subsection 3.2.6, page 56.

3.2 Research contribution

The previous state of the art revealed how detecting human shapes in a color or depth image is a problem that has already been tackled by many researchers. This is why my contributions to this

field mainly focus on integrating the most relevant and efficient algorithms into our architecture.

After a brief image processing introduction of how to retrieve a user mask from a depth image and a seed pixel in (subsection 3.2.1), the common data structure, shared by all the algorithms we integrated and that will be further explained, will be presented in subsection 3.2.2, page 41.

In total, five algorithms for user detection based on computer vision have been integrated in our user awareness architecture.

Two algorithms that were originally based on color images analysis have been implemented, adapted to use this data structure, and improved thanks to the use of the depth image: face detection, in subsection 3.2.3, page 47; and Histogram of Oriented Gradients (HOG) people detection, in subsection 3.2.4, page 50.

On top of these two 2D algorithms, three other algorithms and based on the analysis of the depth image (3D algorithms) have been integrated: the NiTE middleware, powering the Microsoft Kinect presented in subsection 3.2.5, page 53; an algorithm based on the projection of the 3D point cloud into a clever map called Polar-Perspective Map (PPM), in subsection 3.2.6, page 56; and a novel point cloud clustering algorithm based on tabletop object detection techniques, explained in subsection 3.2.7, page 59.

Finally, the comparative performance of each algorithm will be reviewed in subsection 3.2.8, page 63 and conclusions about the most relevant algorithm will be drawn.

3.2.1 Preliminaries: User mask from depth image and seed pixel

In this brief introduction needed for the following user detection algorithms, we will explain how to generate a full user mask given two inputs: the depth image, and a so called **seed** pixel, which is in fact a pixel position that we know for sure that it belongs to a user.

This seed point can be obtained through a wide range of algorithms. It can come from instance from algorithms running on the Red Green Blue (RGB) image stream, such as a face detection, as explained in subsection 3.1.1, page 33, or Histogram of Oriented Gradients (HOG) detectors, seen in subsection 3.1.2, page 34, or even optical tags being detected, which will be introduced in the following chapter.

The idea is to run a propagation algorithm in the depth image, starting from the seed, and stopping at the depth edges, which are pixels where there is an edge discontinuity. This discontinuity can be found at the edge of a user: the background floor is several meters behind the user, which generates a gap in the depth value, as can be seen in the depth images in Figure 3.4 (a).

Depth Canny filter The Canny filter [Canny, 1986] originally helps us to detect edges in a grayscale image. It requires two thresholds, a low and a high one. Two edges maps are obtained by passing, first, a Sobel operator on the grayscale image, then, thresholding with the two

thresholds. A Sobel operator is a linear filter based on a simple 3×3 kernel, and it approximates the gradient operator on the grayscale image.
The high threshold edge map contains broken, discontinuous edges, but they are likely to belong to the real contours of the objects. On the other hand, the low threshold edge map contains continuous edges, but with many edges that are not useful. The two maps are combined to create an *optimal edge map*: if a chain of the low threshold map enables to connect two pixels of the high threshold map otherwise disconnected, this chain is added to the final edge map. All the isolated chains of the low threshold map are then removed.

Coming back at our seed-to-blob problem, the edges can be obtained in the depth image applying a Canny edge detection algorithm on it: the filter is then called a *Depth Canny*. A floodfill propagation from the seed in the Depth Canny edge image will then include all points of the user.

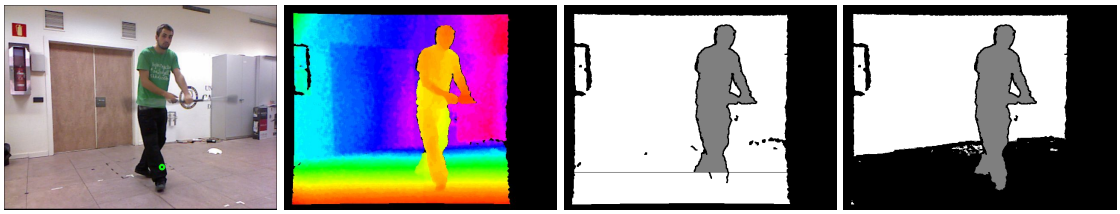
Issues generated by the presence of the ground In some cases, it occurs that the camera of the robot is not tilted upwards, and then the ground and the feet of the user are visible, as for instance in Figure 3.4 (a). The feet of the user are most of the time in contact with the ground, and so there is no depth discontinuity between the user and the ground. As such, the propagation from that seed will carry on further than the user, and include all the pixels of the ground.

To address this issue, I developed two different techniques:

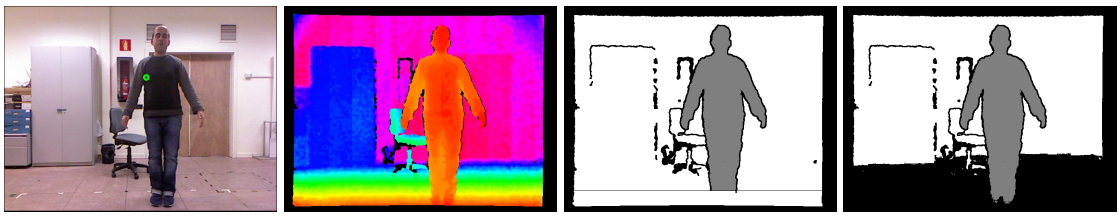
- **Cluster closing:** A cluster closing is a simple test on the edges image to find abrupt transitions from a narrow cluster to a wide one. Starting from the row of the seed, we find the leftmost and the rightmost pixels that are no edges. The width of the user cluster at this row corresponds to the column difference between the left and the right pixel. We reiterate the process for the rows below, and obtain the width of the cluster at those rows. When there is an important increase in the width from one row to the next one, we have most probably reached an opening, for instance when reaching the feet. The cluster is closed by drawing a horizontal edge line at the last row before the opening.

- **Ground plane estimation:** This method consists of estimating the equation of the ground plane, then obtain a binary mask of all the points of the depth image that belong to the ground. The depth image is reprojected to 3D, giving us a three-dimensional (3D) point cloud. If the ground is visible in the depth image, then statistically, a great amount of the 3D points belong to that ground. However, some of the 3D points do not belong to the ground, such as object pixels. Statistically speaking, most 3D points are *inliers* for the ground plane equation, while some are *outliers*. The ground plane equation can thus be restored with a statistical algorithm operating on the point cloud and capable of discarding outliers while optimizing the equation for the inliers. We used the RANSAC algorithm for this task ([Fischler and Bolles, 1981]), implemented in the Point Cloud Library (PCL, [Rusu and Cousins, 2011]).

Results We show in Figure 3.4 different results obtained for the user mask computation, given a depth image and a seed pixel.



(a)



(b)

Figure 3.4: *User mask computation given a depth image and a seed pixel.*
(a) , (b): from left to right, the input RGB image and the seed (green circle); the input depth image; the output final mask for both methods (third picture: cluster clothing and fourth picture: ground plane estimation). In the final mask, the black pixels are either a lack of measure by the device or the edges of the Depth Canny; and the gray pixels correspond to the final user mask. Note how the ground has disappeared in the ground plane estimation mask.

3.2.2 Design of a common interface for user detectors

As seen in the literature review, a wide range of user detectors already exists, and more generally of people detectors [3]. To ease the process of integration of all these detectors into the architecture of the robots of our lab, we designed a framework for user detectors.

Common practice in computer science consists of standardizing the communication layer and the data that is exchanged by the different modules, better than the proper structure of these modules [4]. For this reason, we designed a common data structure that describes the data of a user detection and needs to be filled by any user detector that we want to integrate.

We define the `PeoplePose` (`PP`) data structure: it corresponds to a single user detection and contains all the information worth being shared: the three-dimensional (3D) pose of the user, the confidence of the detection, a Red Green Blue (RGB) and depth images of the user if they are available (for vision-based algorithms), etc.

[3] These two terms refer to algorithms that will do the same thing, which is finding human shapes in the sensors input. However, a *user detector* is more specifically used when a positive interaction between the system and the human person is possible. This can be the case in robotics, in multimedia kiosks, etc.

[4] *Remote Procedure Calls* (RPC), for instance, follow this paradigm.

The PP data structure is implemented as a ROS msg ([Quigley et al., 2009]). This is a simple message description language for describing data structure exchanged by ROS nodes, as previously explained in subsubsection 1.4.1.i, page 9. The details of the PP message are in Code listing 3.1. Note the ':' characters describe the inner fields of a field.

Code listing 3.1: *The PeoplePose message*

```
// static value: the value of "person_name" when no recognition was
   ↪ made (for instance, a simple face detection, but no face
   ↪ recognition)
string NO_RECOGNITION_MADE=NOREC

// static value: the value of "person_name" when the recognition of
   ↪ the person failed (for instance, the face recogntion could not
   ↪ load the model)
string RECOGNITION_FAILED=RECFAIL

// the header, useful for the stamp and the frame
std_msgs/Header header
:time stamp
:string frame_id

// the estimated position and orientation for the head of this person
geometry_msgs/Pose head_pose
:geometry_msgs/Point position
::float64 x, y, z
:geometry_msgs/Quaternion orientation
::float64 x, y, z, w

// the standard deviation of the estimated pose
float32 std_dev

// the supposed name of recognized people (for instance, "Bob"). Only
   ↪  filled by user recognition methods, such as ARToolkit or
   ↪ multimodal fusion.
string person_name

// the confidence (between 0=really unsure and 1=very sur)
float32 confidence

// the color mask of the user.  Most of the time, it does not
   ↪ correspond to the full RGB input image, but a tight crop that
   ↪ only keeps the region of interest of that image.
sensor_msgs/Image rgb
```

```
// the depth mask of the user
sensor_msgs/Image depth

// the binary mask of the user.  The image pixels are = 0 where the
   ↪ user is not,
// and >0 where she is (that is, any point of her body).
sensor_msgs/Image user

// the offset of the rgb, depth, user images in the original input
   ↪ image. Can be used for correct \ac{3D} reprojection for
   ↪ instance
int16 images_offsetx, images_offsety

// a list of attributes of this person, for instance her height, her
   ↪ preferences...
string[] attributes_names

// the values of the attributes definited in attributes_names
string[] attributes_values
```

All fields of the message will not be filled by all methods: for instance, a person detector based on the information of a 2D range finder will not use the image fields.
Note that the `person_name` field already implies some higher-level user recognition routine, that most user detection algorithms do not have. For this reason, most of the time, this field is set at the default value `NO_RECOGNITION_MADE`.

Then, with one given data input, a detector can detect several users at once. For instance, a face detector can find several users in the same RGB pictures. The different PPs generated by each detection are then gathered into a single message: this collection of PPs is then called a `PeoplePoseList`. In addition to the list of PP, the `PeoplePoseList` (PPL) message also contains the name of the algorithm used by the detector and a time stamp. The full structure is visible in Code listing 3.2.

Code listing 3.2: *The PeoplePoseList message*

```
std_msgs/Header header
// the name of the method used, ex: "face_recognition_eigen"
string method
// the list of found poses
people_msgs/PeoplePose[] poses
```

The PP and PPL ROS `msg` files [5] are then transformed during compilation time into equivalent

[5] These file types have been presented in the ROS introduction in subsubsection 1.4.1.i, page 9.

headers in several programming language (C++, Python, Perl to date, more to be integrated). An algorithm capable of generating PPL messages is called a PeoplePoseList Publisher (PPLP) and can be written in any of the programming languages using the corresponding headers. All the people detectors that will be integrated in the robotic architecture will then exchange this common data structure PPL. An example of processing flow with several PPLPs is visible in Figure 3.5.

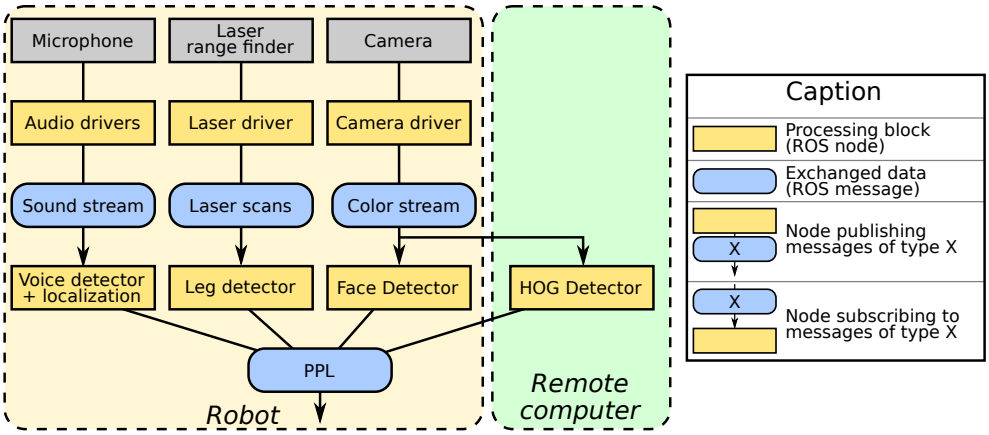


Figure 3.5: *An example of distributed people detection thanks to several PeoplePoseList Publishers (PPLPs) in a robot equipped with a microphone, a laser range finder and a RGB webcam. Note that the different input streams can be shared between different PPLPs: in this example two of them share the color stream of the camera. However, the HOG PPLP is costly and then chosen to run on a remote computer. The topics are relayed between the computers using ROS communication layer. Delays depend on the network capabilities, but the cost generated by the serialization and deserialization of the messages is optimized.*

Advantages of the use of PeoplePoseLists (PPLs) The use of this common data structure has numerous advantages.

- **Programming language abstraction.** The PPL data structure is actually implemented as a Robot Operating System (ROS) msg message file. The ROS compilation system then generates headers corresponding to the message file in different languages, to date C++, Python, and LISP. We can then wrap our algorithm, written with any of these languages, into a PPLP by including these headers. In other words, defining a standard on the message type and not on the design of the algorithm itself results in programming language independence.
- **Workload distribution.** Each ROS node is running in its own thread. Thus, several PPLPs can run in parallel seamlessly and be written in different languages. The distributed nature of ROS even allows these different nodes to run on different computers. For instance, if an algorithm has a high computational cost, its input data can be transferred to an auxiliary CPU where the computation takes place, as illustrated previously in Figure 3.5. ROS

communication layer was optimized to generate a temporal delay as small as possible on the messages handling: even when the network connectivity is unstable, safe data connections are created between publishers and subscribers.

- **Integration of new algorithms.** To be integrated into our architecture, a new user detection algorithm has only one feature to respect: publishing ROS messages of the `PPL` type. This makes their implementation much easier and agile.
- **Debugging made easier.** Because they publish the same data structure, the different `PPLP`s can be debugged thanks to common tools. For instance, a series of tests with annotated images can check the output of a given `PPLP` in basic cases: images with no users in the image, with one user easy to spot, etc.
- **Benchmarking made easier.** For similar reasons, it is easy to measure the performance of different `PPLP`s. This point will be explained more thoroughly in the next paragraph.
- **Common tools.** The design of tools is made easier. For instance, the creation of a visualization tool is easy, as we know the user positions will always be in the `head_pose` field. The implementation of these tools do not depend on the detection algorithms, but on the definition of the `PPL` only.

Benchmarking a `PeoplePoseList` Publisher (`PPLP`) We just introduced how all `PPLP`s were publishing a common message type. A benchmark application was developed taking advantage of this characteristic. It uses files containing both the input data (RGB and depth images) and the manually labeled user position (user mask). Three public academic datasets were considered for benchmarking.

1. The *DGait database* ([Igual et al., 2013]) contains video sequences of 55 users, both female and male, walking on a stage with varying light conditions. Some samples of the dataset are visible in Figure 3.6.
2. The *Gaming3D (G3D) dataset* ([Bloom et al., 2012]) contains a range of gaming actions captured with Microsoft Kinect. The dataset contains synchronized video, depth and skeleton data of 10 subjects performing 20 gaming actions: punch kick, tennis, etc.
3. Finally, the *LIRIS human activities dataset* ([Wolf et al., 2012]) contains (gray/rgb/depth) videos showing people performing various activities taken from daily life (discussing, telephone calls, giving an item etc.). The dataset is fully annotated, where the annotation not only contains information on the action class but also its spatial and temporal positions in the video.

An interface for each of these three datasets was written so that their data could be published into the ROS architecture and used by our `PPLP`s. However, *only the DGait database was finally used for benchmarking*: its high number of different actors and various points of views (front, back and side views) offered a diversity superior to the other datasets.

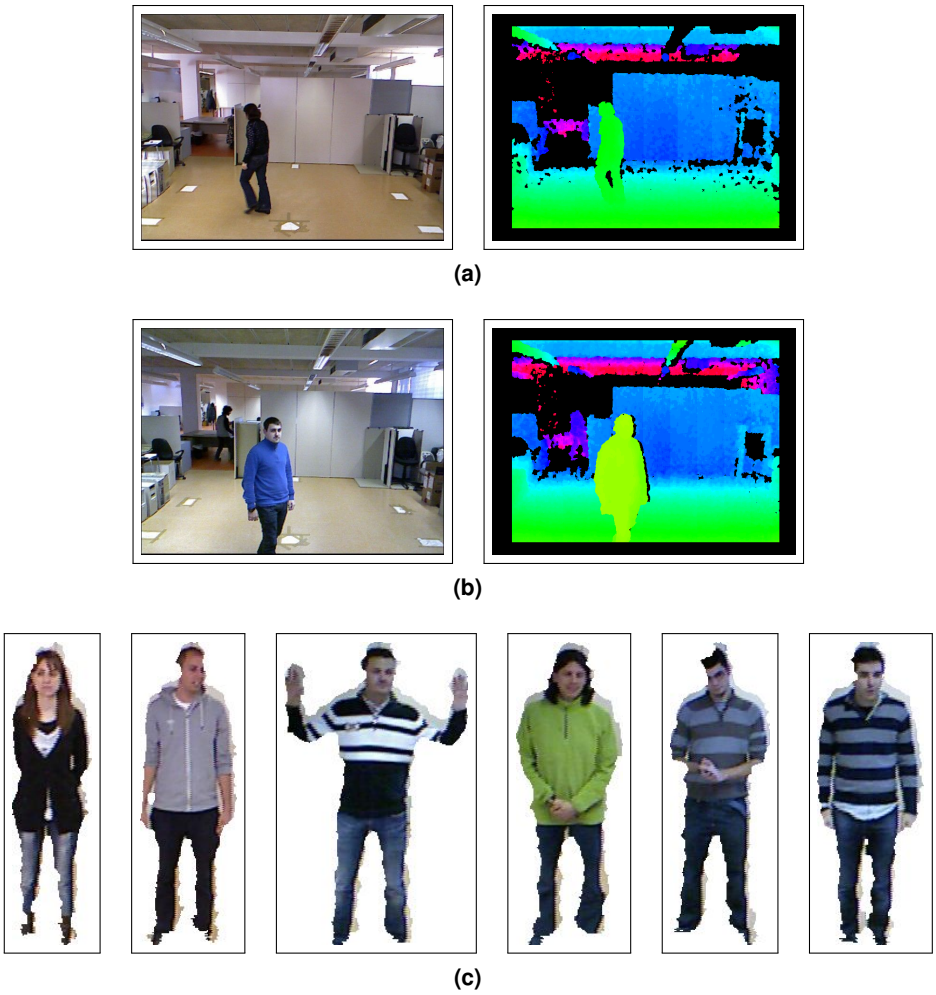


Figure 3.6: *Some samples of the DGait dataset.*
(a) and (b): Sample images from the videos of two different users. Depth images have been remapped to visible colors.
(c): The diversity of users in the dataset.

The benchmark of a PPLP is then made by running it on each frame of all video sequences of the DGait database and comparing its results with the ground truth: the true positive, true negative, false positive, and false negative rates are computed [6]. Note that the benchmark

[6] We will here briefly define these metrics through an example. For more information, please refer to [Olson and Delen, 2008]. Let us imagine we have built an algorithm that can detect the presence of dogs in an input image. A **positive** detection means that the algorithm detects a dog in the current input. Similarly, a **negative** detection means that the algorithm cannot find any dog in the current input. The true and false adjectives will then describe if this result is correct or not.

Thus, a **true positive**, also called **hit**, corresponds to a correct dog detection when the input indeed contains a dog. A **true negative** or **correct rejection** is a negative detection when the inputs does not contain any dog. A **false positive** or **false alarm** is a dog detection while the input contains no dog. This detection is an error of the algorithm,

application can in fact evaluate the performance of several PPLPs at the same time: each of them, structured as a ROS *node*, runs in a different thread. The benchmark subscribes to the output PPL topic of each detector.

Structure of the remainder of the chapter contributions We will now present several PPLPs based on different techniques: face detection and 3D reprojection is at the key of the PPLP of subsection 3.2.3; the Histogram of Oriented Gradients (HOG) detector previously seen powers the HOG PPLP presented in subsection 3.2.4; the NiTE-based PPLP in subsection 3.2.5 makes use of the NiTE middleware; the Polar-Perspective Map (PPM)-based PPLP in subsection 3.2.6 uses a polar transformation and an accumulation process on the ground plane; and finally the tabletop PPLP of subsection 3.2.7 uses point cloud manipulation techniques, detecting users as it would for objects on a table.

3.2.3 Robust three-dimensional (3D) face detection with depth information.

The classical two-dimensional (2D) Viola Jones and neural network detectors presented in subsection 3.1.1 only use the color (Red Green Blue) image data and indicate the position of the found faces in the image frame, by their bounding boxes, in pixels. As our robot uses a Kinect device, we also have depth data available. We can use the depth data to discard false positive detections, that is, zones of the image that are incorrectly classified as faces by the 2D detector. The underlying idea is the following: the 3D points corresponding to a face obey certain geometric constraints, especially in the width and height of their bounding box. Indeed, two points belonging to one given face cannot be away one of the other of, say, more than one meter.

As such, to determine if a zone of the image classified as a face by the Viola-Jones detector is really a face, we will sample a given number of 2D points from this zone, that we will reproject to 3D using the depth (distance) image. If the bounding box of these reprojected points does not comply with generic given geometric constraints, this detection is classified as a false positive and discarded. The pipeline is illustrated in Figure 3.7. A sample is visible in Figure 3.8.

Face-based PeoplePoseList **Publisher implementation and benchmarking** The face detection PeoplePoseList Publisher (PPLP) was implemented in the robot MOPI presented in subsubsection 1.3.1.ii, page 6. It was implemented in C++, using the Viola-Jones implementation supplied by OpenCV ([Bradski and Kaehler, 2008]). The Red Green Blue (RGB) and depth data are supplied by the Kinect drivers. The thresholds for the dimensions of an acceptable face bounding box were determined experimentally: the maximum bounding-box depth of the face points is set to 30 cm and its height to 40 cm. Note these thresholds are reconfigurable dynamically thanks to the mechanism of ROS *parameters*.

it could be triggered by, say, the presence of a cat. Finally, a **false negative** or **miss** is when the algorithm does not detect a dog that was in the input.

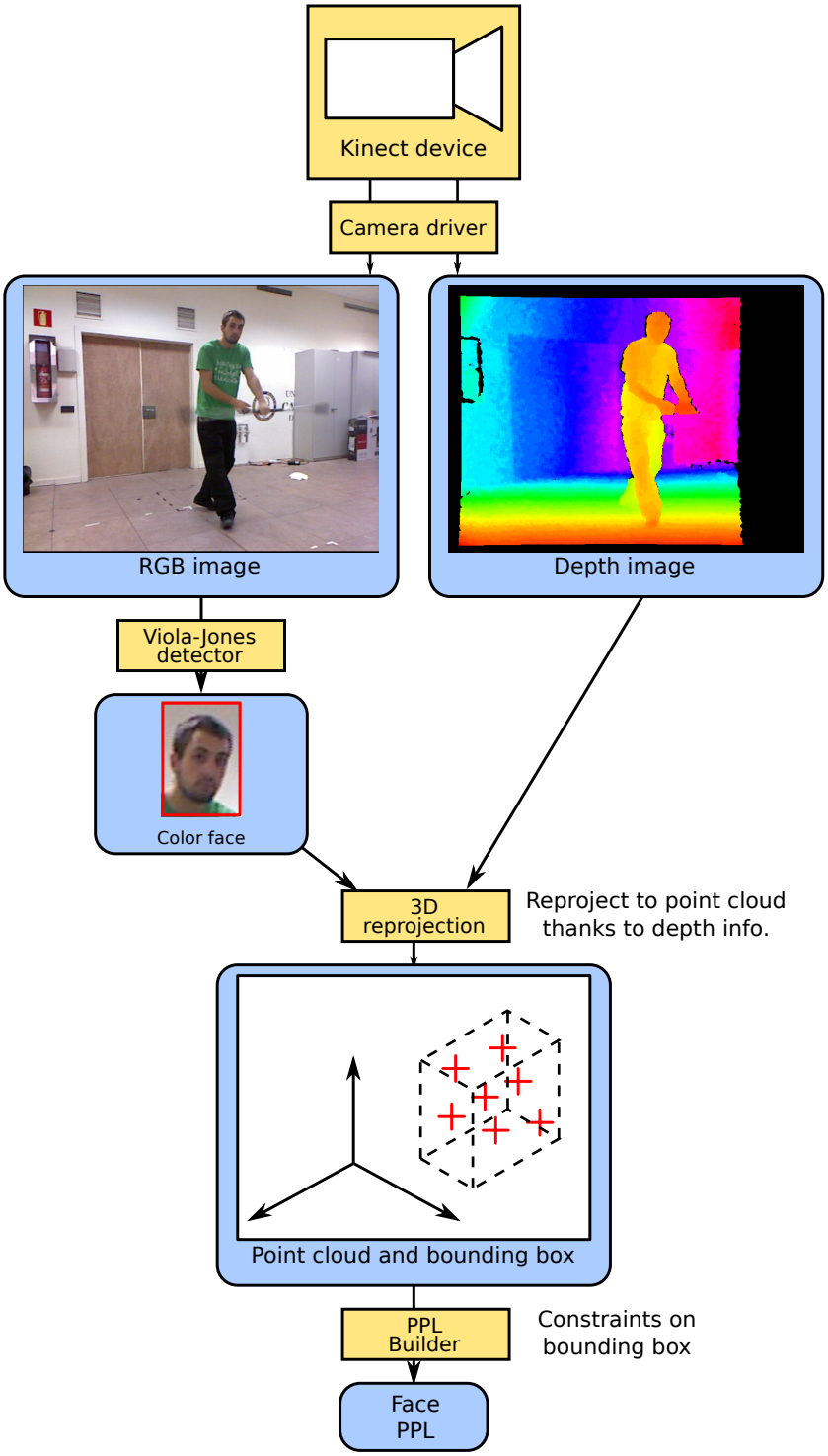


Figure 3.7: *Processing pipeline for the face detection algorithm.*

Figure 3.8: *Results of the face detection algorithm on a sample image. The red and green rectangles correspond to the faces as detected by the Viola-Jones detector. The inner yellow boxes are the zones where the sample 2D points are being reprojected in 3D. The 3D-points bounding boxes of the correct detections pass the geometric constraints of a normal 3D face, hence their green color. The 3D-points bounding box of the wall is too big in dimensions and hence is discarded.*

For each face detection, an output `PeoplePose` (`PP`) message is built: the RGB, depth, and user masks are obtained via a propagation from seed, as seen in subsection 3.2.1, page 39, using the 2D center of the face detection as seed. The 3D reprojection of this same seed is used as output user position.

The performance of the algorithm was measured thanks to the benchmark based on the DGait database presented insubsection 3.2.2, page 41. The results are gathered in table Table 3.1.

True positives	3655
True negatives	1606
False positives	36
False negatives	12457
Hit rate	22.7%
Accuracy	29.6%
Position error per true positive (m)	0.21

Table 3.1: *Benchmark results for the face detection`PeoplePoseList` Publisher (`PPLP`).*

Note that the false positive rate is very low: our contribution, the use of the depth information to discard false positives, indeed limits the detection of non existing faces.

However, the hit rate is under 25%: less than one fourth of the users were detected. We already commented that the Viola-Jones detector, on which this `PPLP` is based, only works well with frontal faces. The dataset is made of users walking on a stage, which means that, unless

they look in direction of the camera, their face is not visible. Furthermore, this detector only detects well faces that are reasonably large in the input image (the minimum size can be of course lowered, but this triggers an important increase of false positives). Both these reasons account for the low hit rate.

3.2.4 Improvement of the original Histogram of Oriented Gradients (HOG) detector and integration as a `PeoplePoseList` Publisher

The two-dimensional Histogram of Oriented Gradients detector, was introduced in subsection 3.1.2, page 34, and it was shown how it can be used for detecting people in front of the camera.

However, the detector is made for finding people in a Red Green Blue (RGB) image, and supplies a rough rectangular estimate of their position in that image. In our case, we are interested in their three-dimensional (3D) position, and we can make use of the depth image too.

In this section, we will see how we can overcome these limitations so as to transform the human shape HOG detector into a `PeoplePoseList` Publisher.

3.2.4.i HOG detection and `PeoplePoseList` Publisher

As mentioned before, the HOG detector we use takes as input a RGB image and find the human shapes in the image. Note that our current implementation is based on the OpenCV HOG C++ implementation [Bradski and Kaehler, 2008]. The search is performed with a scaling factor of 1.05 (increase the detection window size by 1.05 for each search scale). These human locations are in fact rectangles that give a rough bounding box of the user. This information is two-dimensional: the rectangles indicate where the user is in pixel coordinates.

We are interested in converting this information into an accurate user location (in meters) and a corresponding binary user mask. We could use the same method as the face detection `PeoplePoseList` Publisher: use the geometric center of each rectangle as a seed pixel and retrieve the associated blob, as seen in subsection 3.2.1, page 39. However, the rectangles returned by the 2D HOG detector are only rough estimates of the location of the users: it is possible that their center pixel do not belong to the user's mask, which is a prerequisite for the use of this method.

Another approach is to consider that the user represents most of the content of the rectangle. As such, for a given detection r of the HOG detector, if r is a positive detection, i.e., there is a user in r, then this user corresponds to the main 3D cluster inside of r. In other words, the user corresponds to the main 3D cluster in the point cloud generated by reprojecting each point of r. This gives us a roadmap for a successful conversion of the HOG detector into a `PeoplePoseList` Publisher. The processing pipeline is made of several stages, as seen on Figure 3.9. The details are given in the following algorithm 1.

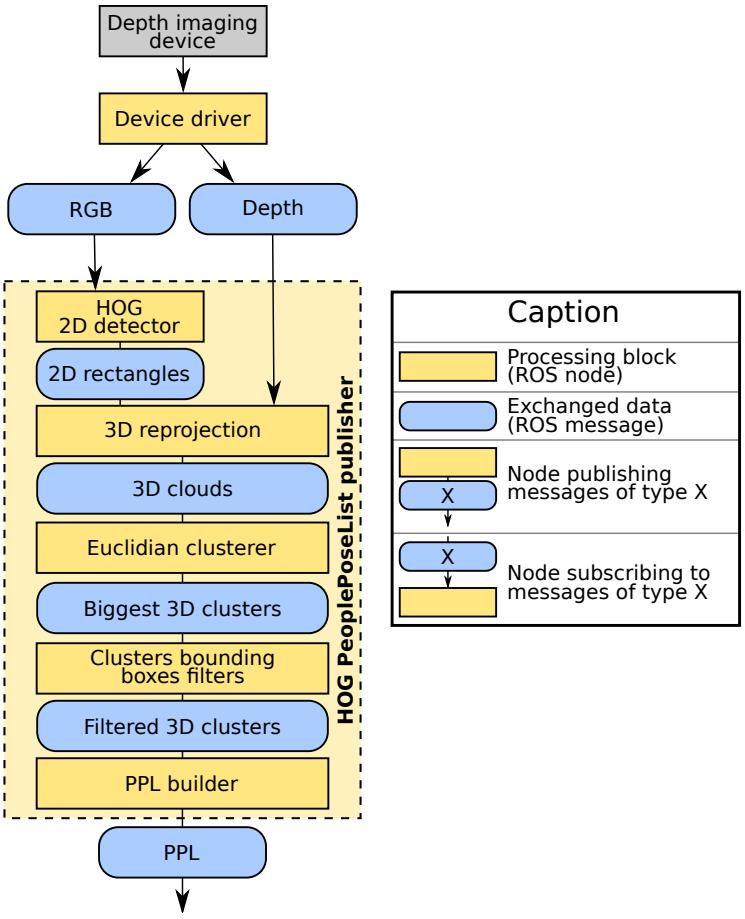


Figure 3.9: *The pipeline for the HOG-based* `PeoplePoseList` *Publisher.*

The key step of the algorithm is the clustering of the 3D point cloud c_r (generated by reprojecting in 3D the pixels in a HOG detection r). This is done thanks to the PCL (Point Cloud Library). [7] More details are available in [Rusu, 2009]. We used an Euclidean neighbor distance threshold of 10 cm.

The biggest cluster of the 3D point cloud c_r should then correspond to the user shape. Constraints on the 3D bounding box of this biggest cluster enable the filtering of false positives, in a way similar to the improved face detection of subsection 3.2.3, page 47. These constraints are actually permissive ranges on the values in meters of the height, width and depth of this 3D bounding box. If one of its dimensions is outside these thresholds, then the detection r is

[7] `http://www.pointclouds.org/`. The clustering makes use of a compact and fast representation of the cloud thanks to a Kd-tree, and fast access to the neighbor points of a query thanks to an octree. Very briefly, the algorithm goes as follows: for the first point of the cloud, creating a queue with its Euclidean neighbors. Then for each point of the queue that was not seen already, recursively add the neighbors of this point to the queue.

Data: RGB image rgb, depth image $depth$
Result: PPL message ppl
RectangleVector $R \leftarrow HOG\_detect(rgb)$;
initialize PointCloudVector U ;
for *Rectangle* $r \in R$ **do**
    `// reproject only ROI of the depth image`
    PointCloud $c_r \leftarrow reproject3D(depth(r))$;
    PointCloudVector $C \leftarrow euclidean\_cluster(c_r)$;
    PointCloud $u_r \leftarrow$ biggest cluster (in number of points) of C ;
    if $bounding\_box(_u r)$ *has human dimensions* **then**
        add u_r to U ;
for *PointCloud* $u \in U$ **do** `// build PPL`
    initialize `PeoplePose` (`PP`) pp ;
    $pp.head\_pose \leftarrow centroid3D(u)$;
    $mask \leftarrow reproject2D(u)$;
    $pp.user \leftarrow mask$;
    $pp.rgb \leftarrow rgb(mask)$;
    $pp.depth \leftarrow depth(mask)$;
    add pp to ppl ;
return ppl ;

Algorithm 1: The processing pipeline for the HOG PeoplePoseList generator

discarded. Otherwise, a `PP` message is built. The final 3D position of the user corresponds to the 3D centroid of the main cluster. A sample is visible in Figure 3.10.

3.2.4.ii Benchmarking of the HOG `PeoplePoseList` Publisher

We measured the performance of the HOG `PPLP` thanks to the benchmark based on the DGait database and presented in subsection 3.2.2, page 41, as we previously did for the face detection `PPLP`s. The results are visible in Table 3.2.

True positives	11415
True negatives	1422
False positives	846
False negatives	4713
Hit rate	70.8%
Accuracy	69.8%
Position error per true positive (m)	0.33

Table 3.2: *Benchmark results for the HOG-based`PeoplePoseList` Publisher (`PPLP`).*

Figure 3.10: *A sample true positive detection with the HOG* `PPLP`*. The red rectangle corresponds to the classical HOG detector. The red dots correspond to the 2D reprojection of the main 3D cluster.*

We can first be surprised that this widespread `PPLP` obtains an accuracy of roughly 70%. If we consider the fact that the detections are made using only the color information, the depth being used only for false positive removals, this is a fairly reliable algorithm: more than two thirds of the users will be detected.

The two `PPLP`s we just saw, face detection `PPLP` and HOG `PPLP`, were based on 2D algorithms, as they were originally developed for RGB images. The following algorithms we will now present were made for 3D information.

3.2.5 NiTE-based `PeoplePoseList` Publisher

The NiTE middleware for the Microsoft Kinect device seen in subsection 3.1.3, page 36 supplies a data structure that is straightforward to convert into a `PeoplePoseList` (`PPL`): the *users muti-mask*. We have seen that the unique values of the users muti-mask, which are strictly positive integers, can be seen as the users IDs. The NiTE middleware already includes tracking capabilities: the same physical user, whether she is visible in successive frames, is identified by the same ID in the resulting successive multi-masks.

Implementation The number of users visible by the camera is equal to the number of unique non-zero values (IDs) in the users muti-mask. For each ID of the users muti-mask, we define the *user mask* as the set of pixels in the users multi-mask that have a value equal to this ID. This is equivalent to setting all other IDs in the multi-mask to zero. Then, the three-dimensional (3D) position of the user with this ID is the Center of Mass (COM) of the reprojection in 3D of this user blob. The user Red Green Blue (RGB) image corresponds to the part of RGB image where the user mask is non null. Similarly, the depth image of the user is the intersection of the depth image and the user mask.

These properties are used for the conversion between the users multi-mask and the `PPL` message. First, we determine all the unique values (IDs) of the multi-mask, then for each ID, we compute the Center of Mass (COM) of its user mask. This COM is reprojected to 3D and stored in the `PPL` message.

The NiTE-based `PeoplePoseList` Publisher (`PPLP`) was implemented in robots Maggie and MOPI, both presented in the Introduction chapter, using their embedded Kinects. The `PPLP` is structured as a C++ ROS *node* that subscribes to the three image streams published by the NiTE middleware node: RGB, depth, and multi-mask. The full processing pipeline is visible in Figure 3.11. A sample of user detection and `PPL` building thanks to the NiTE middleware is visible in Figure 3.12.

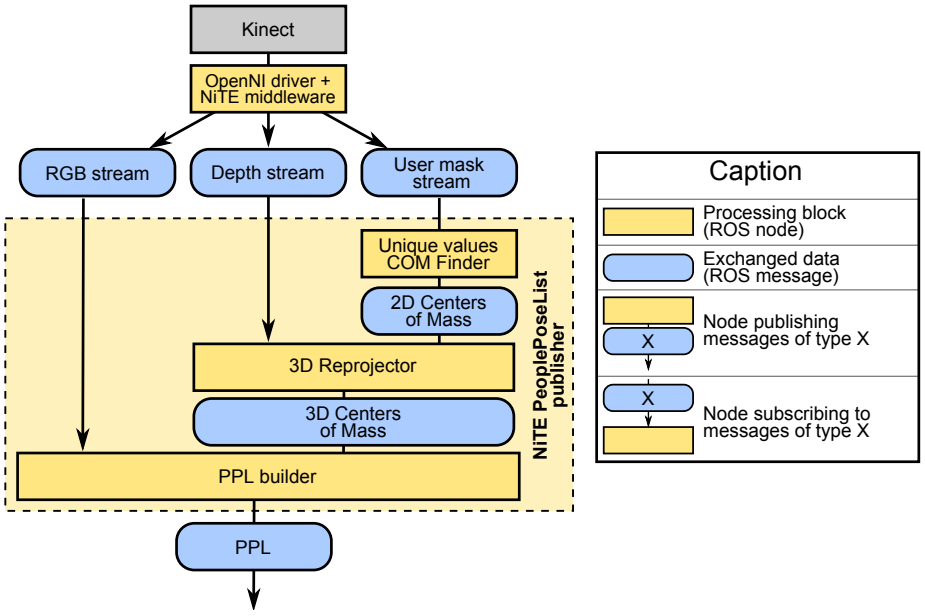


Figure 3.11: *The NiTE-based user detector pipeline.*

Limitations and benchmarking of the NiTE-based `PeoplePoseList` Publisher The benchmark presented earlier [Igual et al., 2013], using manually labeled videos of 55 users walking on a stage, was used to measure the performance of the algorithm. Note that the user mask

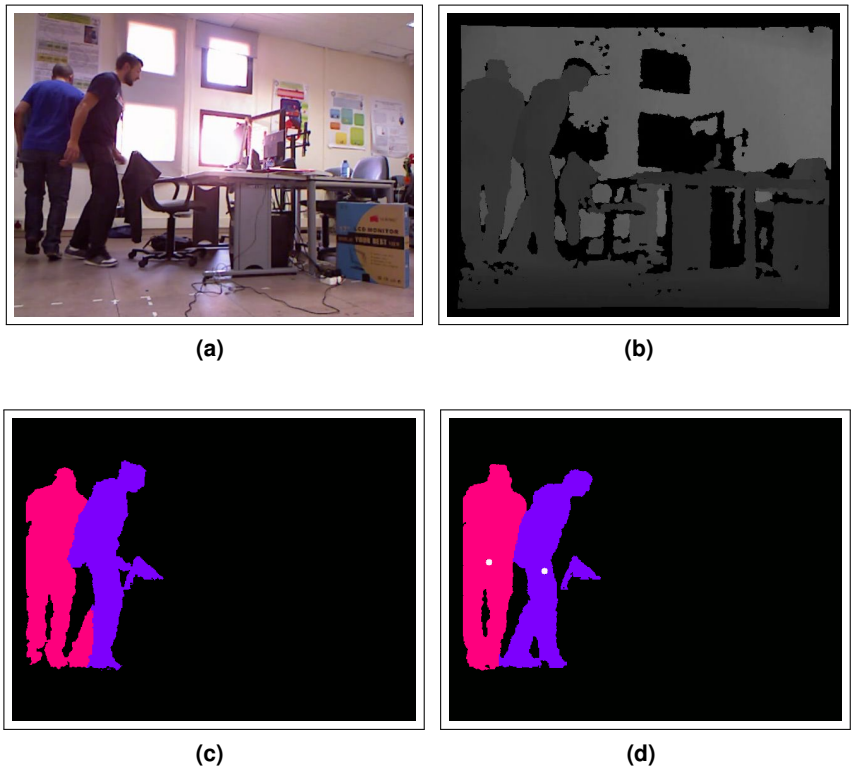


Figure 3.12: *Sample images of the NiTE-based* `PeoplePoseList` *Publisher.*
(a) , (b) , (c): the data supplied by the NiTE middleware, respectively the RGB image; the depth image, remapped to grayscale; the users multi-mask;
(d): the Center of Mass of each user (white circle);

supplied by the dataset was actually obtained by its authors using the NiTE middleware, so, thanks to this dataset we can measure the accuracy of our wrapper that converts the three image topics (RGB, depth, and user multi-mask) to `PPL`. The results are visible in Table 3.3.

True positives	16076
True negatives	1608
False positives	9
False negatives	1
Hit rate	100.0%
Accuracy	99.9%
Position error per true positive (m)	0.00

Table 3.3: *Benchmark results for the NiTE-based* `PeoplePoseList` *Publisher (PPLP).*

For the reasons we just explained, the accuracy and precision are roughly equal to one. This

does not mean that the detection never fails, it means that it fails as often as the labeling of the dataset: the labeling made by NiTE is faithfully converted by our wrapper, and so the failures in the labeling, triggered by NiTE failures, generate identical failures in the `PPLP`.

The limitations of the NiTE middleware are hard to demonstrate thanks to a dataset. The software is indeed built in such a way that it can only be used with a live stream coming from a Kinect device. The NiTE-based `PeoplePoseList` Publisher has the same limitations as the NiTE middleware itself, presented in section 3.1.3, page 37. Most seen errors are linked with ID swaps: when two walking users cross in front of the Kinect, the middleware will erroneously exchange their IDs. However, even if we here miss numerical data to measure it, the NiTE `PPLP` is a relatively stable and accurate `PPLP`.

3.2.6 Polar-Perspective Map (PPM)-based `PeoplePoseList` Publisher

This people detector uses the fact that a person appears as a three-dimensional (3D) cluster, i.e. a set of points tightly close one to another, in the 3D point cloud given by the range-imaging device (Microsoft Kinect). When projecting this 3D cloud on the ground plane, these clusters will be projected onto the same area, thus generating a sort of high-density blob on the ground plane. Standing persons can then easily be characterized by the size of the blob: their height will most likely belong to a span of characteristic human sizes, while their width and height correspond to the footprint of a human user.

PPM computation The so-called *Polar-Perspective Map (PPM)* is an occupancy map based on the polar coordinate system: it uses a regular grid based on the bearing of the points and their inverse distance to the device. As such, the resolution of closer points is higher than far points: unlike the Cartesian map, it can be tuned to match exactly the resolution of the device. The PPM was introduced for a pedestrian detection system by [Howard and Matthies, 2007] and was already explained in subsection 3.1.4.

We implemented the PPM-based user detector using the geometric transform from the Cartesian system of coordinates to the PPM indicated by [Howard and Matthies, 2007] and having the following definition:

$$f : \begin{cases} \mathbb{R}^2 & \mapsto & \mathbb{R}^2 \\ (x, y) & \rightarrow & (r, \theta) = \left(\sqrt{x^2 + y^2}, \tan^{-1} \frac{y}{x} \right) \end{cases}$$

The full processing pipeline is visible in Figure 3.13 and some samples in Figure 3.14.

First are chosen two resolutions: a spatial one (corresponding to steps in r) and an angular one (steps in θ). Each pixel of the point cloud is transformed into the PPM coordinates and the corresponding cell of the PPM is aggregated: this step is also called **Accumulation**. We maintain at the same time a *reverse map* that stores the list of 3D points that were projected onto a given cell of the PPM. A sample is visible in Figure 3.14 (c). Then the PPM is thresholded:

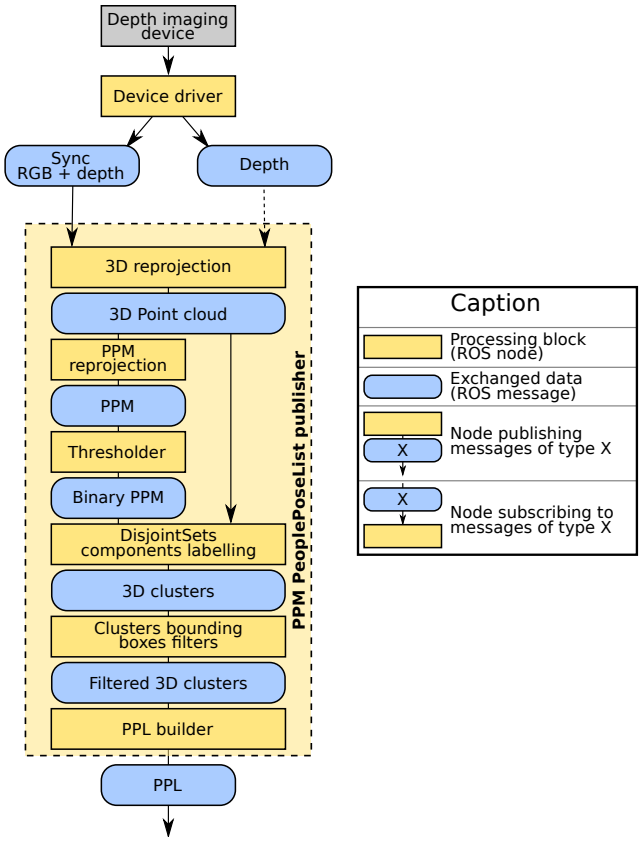


Figure 3.13: *The PPM-based user detector pipeline.*

all cells that have a minimum given of pixels are marked as 1, the others as 0, thus generating a binary mask, as visible in Figure 3.14 (d).

The objects in the scene correspond to the connected components in the image. In [Ramey, 2011], we present a lightweight and fast algorithm for fetching connected components in a monochrome image, such as our edge map. It is based on an efficient representation in memory of the connectivity between components thanks to a *disjoint-sets* forest. The disjoint-sets data structure was first presented in [Galler and Fisher, 1964] in 1964. In [Ramey, 2012], this algorithm was benchmarked against two widespread ones for components labelling, flood-fill and Chang's [Chang et al., 2010]. On the image collection used in [Ramey, 2012], the new algorithm turned out to be 30% faster than the two widespread ones.
We use this algorithm for fetching the different components of the PPM, as visible in Figure 3.14 (e). The 3D bounding box of each connected components is obtained by considering the cluster made by all the 3D points that belong to that component, which is done using the previously mentioned reverse map.

Then, we here focus on detecting standing humans: some empirical thresholds on the bounding boxes will discard most of the connected components, enabling to keep only the ones

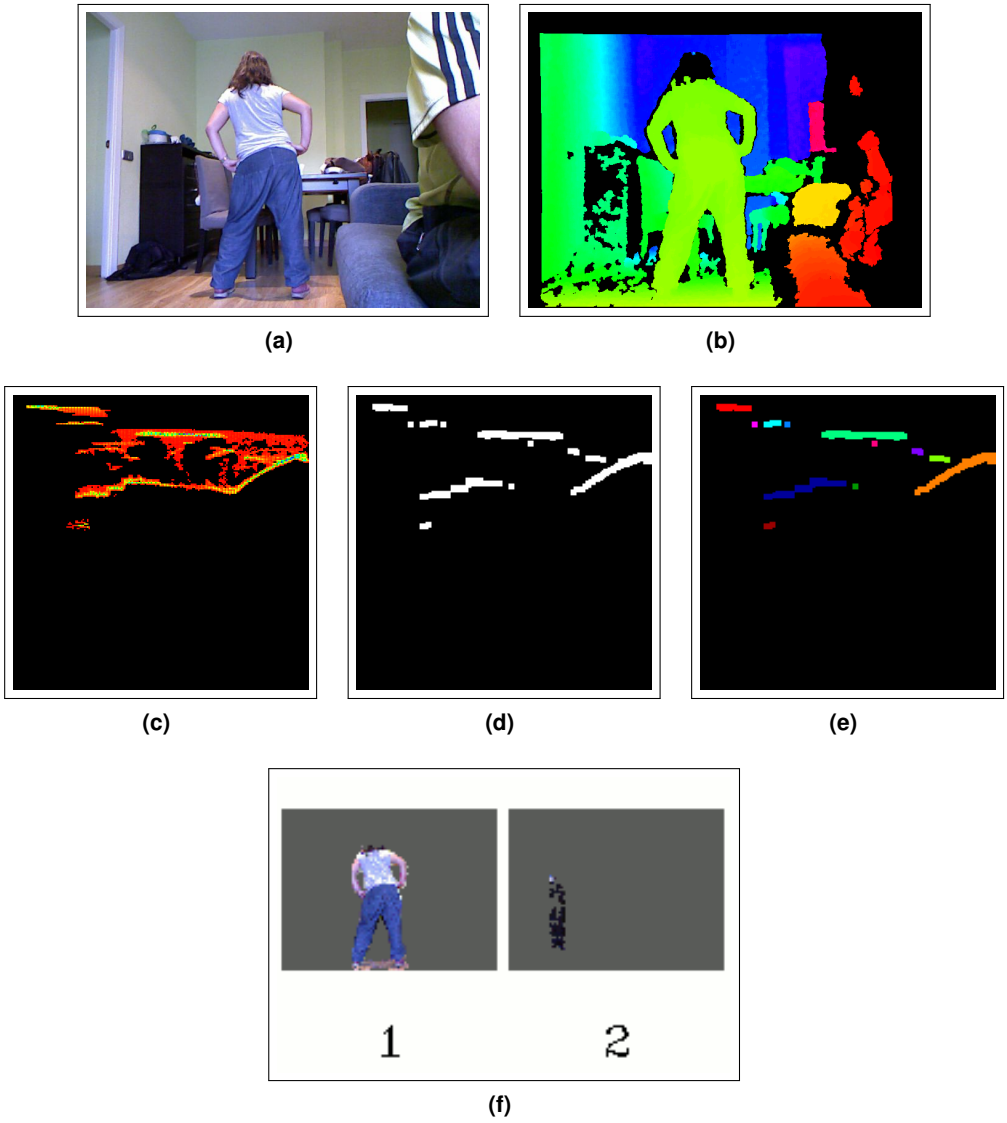


Figure 3.14: *User detection thanks to Polar Perspective Maps (PPM).*
(a): The input RGB image given by the range-imaging device (Kinect).
(b): The depth map, as supplied by the Kinect, and remapped to visible colors (Hue scale).
(c): The PPM, remapped to visible colors (Hue scale). Empty bins are shown as black pixels, red colors correspond to almost empty bins, while green bins correspond to fuller bins and blue the fullest.
(d): The previous PPM, thresholded and thus transformed into a binary map.
(e): The connected components labeled by color of the thresholded PPM.
(f): The remaining connected components after eliminating the ones that do not pass the tests on the bounding boxes. The person on the left is a true positive. Note that the cupboard in the right image is incorrectly identified as a person.

that are shaped as a standing person. In our implementation, the width must be between 0.3 and 1 meters, the height between 1.2 and 2.1 meters (thus also including most children aged over 6), and the depth between 0 and 1 meters. These values were chosen so that they include all human configurations, while discarding a reasonable amount of non-human clusters. In other words, we aim at having a false-negative rate of zero, while having a low false-positive rate. As can be seen on the sample in Figure 3.14 (f), we indeed have some false-negatives from time to time. All objects in the scene that have dimensions in these spans will indeed be labeled as users.

Finally, the 3D clusters that passed all tests are likely to be users. We then shape a `PeoplePoseList` (`PPL`) message for each cluster. The user mask corresponds to the 2D reprojection of the cluster in the camera frame. In this `PPL` are also stored the corresponding parts of the Red Green Blue (RGB) and depth images, and the 3D Center of Mass of the centroid.

PPM-based `PeoplePoseList` Publisher benchmarking The DGait database already presented was used to benchmark the PPM-based `PeoplePoseList` Publisher (`PPLP`). The 55 videos of different users walking on the stage were used to compute a PPM on each frame and determine the presence of a user. The results are gathered in Table 3.4.

True positives	12719
True negatives	1161
False positives	3857
False negatives	3353
Hit rate	79.1%
Accuracy	65.8%
Position error per true positive (m)	0.62

Table 3.4: *Benchmark results for the PPM-based`PeoplePoseList` Publisher (`PPLP`).*

The PPM-based `PPLP` has an accuracy of about two thirds. It is designed to work specifically with standing humans, which is no limitation for this benchmark, as all users are standing or walking in this dataset. Note that because the PPM has no maximum distance, all the remote objects (walls, cupboards, etc.) are used for its building and are erroneously recognized as users from time to time. This explains the numerical difference between hit rate and accuracy for this detector.

The detector that we will now present also handles depth information, but it will limit this erroneous recognition of remote objects by using a so-called depth *clamping*: the depth values greater than a given threshold are not taken into account.

3.2.7 Tabletop `PeoplePoseList` Publisher

People standing on the floor generate a point cloud that is similar to objects standing on a table top. Detecting objects on a planar surface such as a table top is a problem that has already been tackled by other authors, most notably for object grasping ([Blodow and Rusu, 2009, Marton et al., 2011]). We here chose to combine some of these techniques in an innovative way to find the users in front of the robot.

We implemented a straightforward `PeoplePoseList` Publisher using such techniques. It is based on detecting the ground plane thanks to a statistical method, then retrieving the pixel blobs of the objects being on top of that object. *We here make the strong hypothesis than all the blobs visible in a given range of distances are only users.* For instance, no furniture, box, or any object should enter in this range of distances. This is a very restrictive hypothesis, but it can be met in some cases: an uncluttered environment such as a wide living room or a lab fulfills this condition.

The computation is made of various stages, that will explained in the following paragraphs. The pipeline is illustrated in Figure 3.15, and a sample is visible in Figure 3.16. This sample will be referred to while we describe the processing pipeline.

Depth map clamping We supposed the users need to be detected only in a limited and static range of distances. This makes sense in a configuration where the camera is in a given position, say a fixed camera on a robot with no motion capabilities. We name $d_{min}, d_{max} \in \mathbb{R}$ the minimum and maximum distances where the user can appear. Parameter d_{min} would typically be around half a meter, and d_{max} slightly smaller than the closest wall. Then we clamp our depth map: we mark all pixels with a value smaller than d_{min} or greater than d_{max} as not valid. All pixels in $[d_{min}, d_{max}]$ belong either to users, or to the ground. See how such a filter affects the depth map on Figure 3.16 (c).

Ground plane detection The first step is the detection of the ground plane. In other words, we want to detect which points in the image belong to the ground, and which belong to objects on top of it. A three-dimensional (3D) plane can be represented as a 4-variables equation:

$$ax + by + cz + d = 0,\ a, b, c, d \in \mathbb{R}$$

We here use the classical world coordinates frame, where the ground plane is described by the x and y axes, while z corresponds to the height.

Two methods are available to compute the plane equation.

1. **Coordinates transform:** If the model of the camera is available, i.e. if we have calibrated our camera and both intrinsics and extrinsics parameters are available (cf [Bradski and Kaehler, 2008]), then we can directly transform the coordinates of any point from the camera frame to the world frame. The ground plane points are then $\{P \in \mathbb{R}^3;\ P_{world}.z \approx$

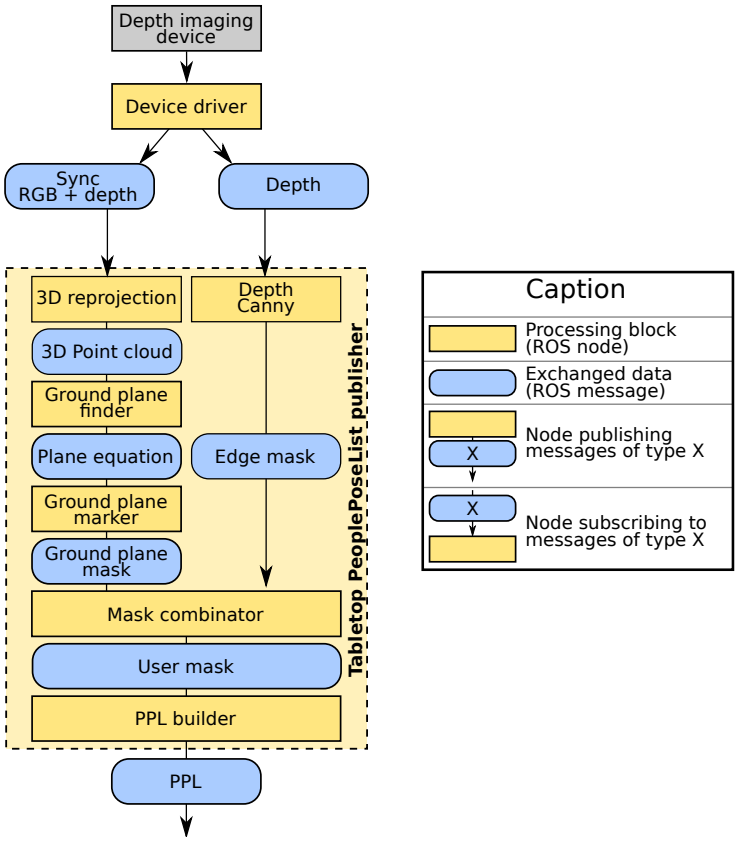


Figure 3.15: *The tabletop user detector pipeline.*

$0\}$.
However, this requires the calibration of the camera and the estimation of the camera in the world frame, which can turn out to be a tedious task.

2. **RANSAC plane estimator:** We saw in subsection 3.2.1 how we could use the RANSAC algorithm ([Fischler and Bolles, 1981, Rusu and Cousins, 2011]) on the 3D point cloud generated by the depth image to estimate the plane of the ground.

Blob marking Once we have obtained the equation of the ground, we can evaluate which pixels of the depth map belong to it. For each pixel, we evaluate the absolute distance between its 3D reprojection and the plane. If the distance is greater a given threshold, the point is evaluated as belonging to an object. Otherwise it belongs to the ground. This enables the generation of a ground plane mask, as seen in Figure 3.16 (d).

The user mask is obtained by combining the binary inverse of this first mask (thus removing all ground points) with the depth mask of the points in $[d_{min}, d_{max}]$ described before. In order to separate aligned users, we combine this user mask with a Canny filter on the depth map that will

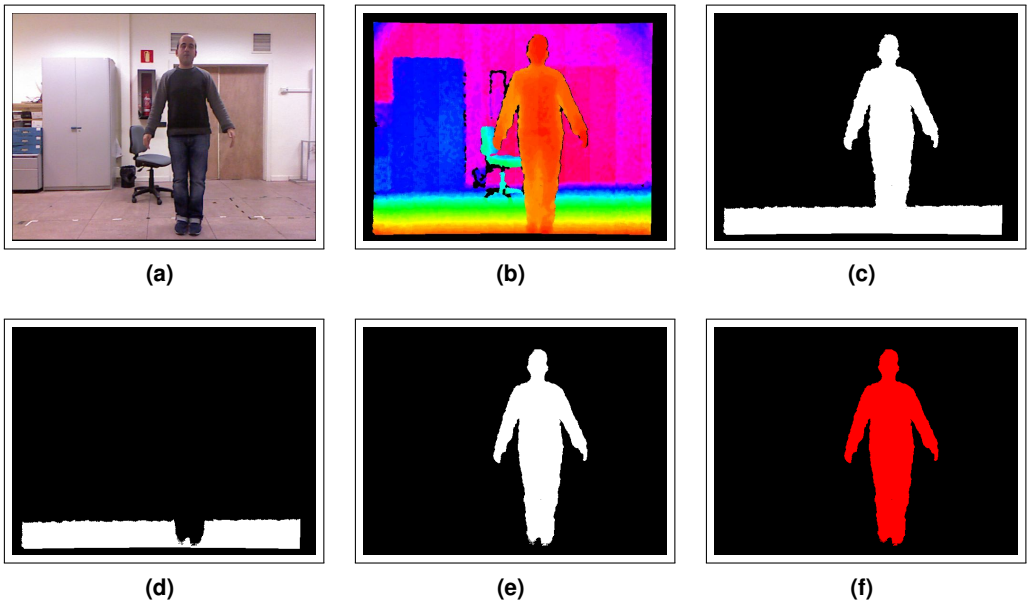


Figure 3.16: *User detection thanks to the tabletop* `PeoplePoseList` *Publisher.*
(a): The input RGB image given by the range-imaging device (Kinect).
(b): The depth map, as supplied by the Kinect, and remapped to visible colors (Hue scale).
(c): The valid measures of the depth map, belonging to the chosen range of depth $[1, 4]$ *(meters)*
(d): The mask of the pixels belonging to the computed ground plane.
(e): The mask of all visible objects.
(f): The person blobs. In this example, there is only one person, so one color blob.

mark the edges (depth disparities) as described in section 3.2.1, page 39.

This generates a binary map where objects are indicated by a positive value, the rest being zero. A sample is visible in Figure 3.16 (e).

Blob retrieval In section 3.2.6, we saw how to quickly retrieve the connected components in the binary map. We supposed all objects in $[d_{min}, d_{max}]$ can only be users. Consequently, all connected components of the binary map correspond to the user. A sample is visible in Figure 3.16 (f).

These components are then passed to a `PeoplePoseList` Publisher (`PPLP`), that shares this message with the rest of the system, in a way very similar to the Polar-Perspective Map (PPM)-baed `PeoplePoseList` Publisher seen in subsection 3.2.6.

Tabletop `PeoplePoseList` Publisher benchmarking In a way similar to the other `PPLP`s, the tabletop `PPLP` was benchmarked on the DGait database [Igual et al., 2013]. The results are gathered in Table 3.5.

True positives	16020
True negatives	1586
False positives	11
False negatives	45
Hit rate	99.7%
Accuracy	99.7%
Position error per true positive (m)	0.04

Table 3.5: *Benchmark results for the tabletop-based* `PeoplePoseList` *Publisher (PPLP).*

The accuracy of the algorithm is very close to one: the detector is almost always true in its predictions. We have to remember the strong assumption we made before though: we consider an uncluttered environment, with a maximum number of users known. The database is made of users walking on a stage, which corresponds exactly to these settings. This justifies the good results.

Note that the PPM `PPLP` presented before does not have such a depth range and obtains an accuracy of two thirds: the remote objects (walls, cupboards, etc.) are erroneously recognized as users. The depth clamping of the tabletop detector prevents this confusion. This accuracy would be much lower if there was an alien object on stage, such as a cupboard for instance.

3.2.8 Comparative performance of the different `PeoplePoseList` Publishers

The performance of the different `PeoplePoseList` Publishers (`PPLP`s) on the DGait dataset, presented in the previous subsections, was gathered in a chart visible in Figure 3.17.

The algorithms have very different performance on this dataset. The NiTE and tabletop `PPLP`s obtain excellent results, as both their accuracy and recall are very close to one. Polar-Perspective Map (PPM)-based `PPLP`s and Histogram of Oriented Gradients (HOG)-based `PPLP`s obtains both recall and accuracy around 70%, which makes them reliable too. On the other hand, face detection performs worse, as it only detects roughly one user out of five.

The face-detection-based `PPLP` has a poor performance on this benchmark: the users visible in the video stand several meters away from the camera and is often turning her back to it, which does not correspond to the domain of use of face detection, thought for interaction at short distance. On the other hand, as already mentioned when these individual results were given, the uncluttered environment is a key explanation for the outstanding performance of segmentation-based detectors. The introduction of any object shaped like an human user, say a coat rack, would generate false positives and thus alter the accuracy rate of these. The algorithms based on the color analysis (HOG and face detection) wouldn't be affected by this new object. For this reason, the HOG algorithm is to be noticed: its good performance, already seen in the literature review, is here experimentally verified.

We can already see here that each of these detectors has advantages and limitations: the

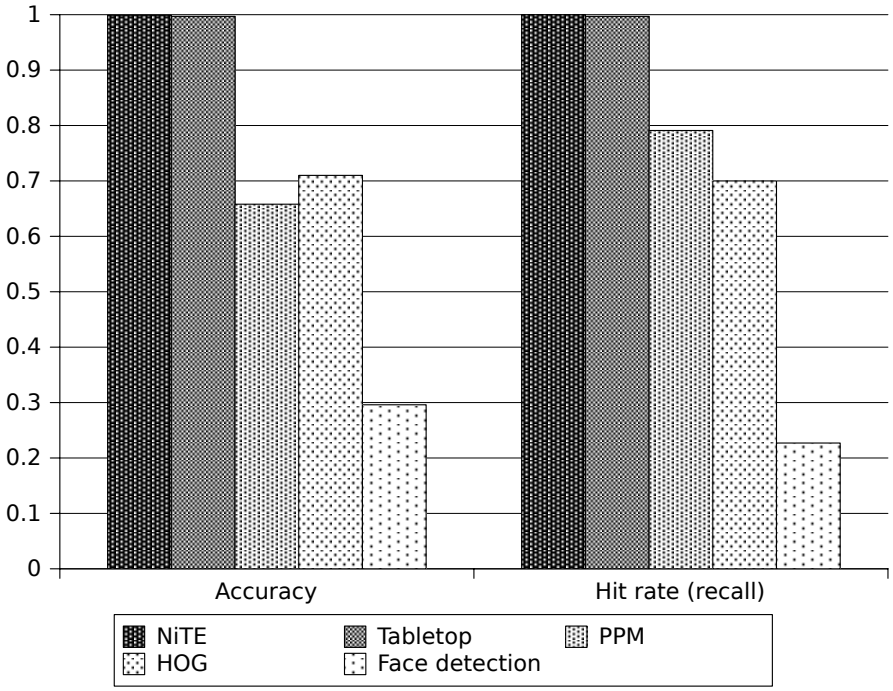


Figure 3.17: *Comparative performance of the different* `PeoplePoseList` *Publishers.*

face detector works well for facing users, while the tabletop PPLP detects well remote users but is easily fooled by objects with dimensions similar to humans. A fusion mechanism that would use all these algorithms and cleverly mix their results according to the context would improve the performance. This idea will be later discussed more thoroughly.

Summary of the chapter

In this chapter, we tackled the problem of detecting users around a social robot using image processing and computer vision techniques. A common data structure called `PeoplePoseList` (`PPL`) was designed to standardize the output of the different algorithms that were integrated. This results in numerous advantages: programming language abstraction (thanks to Robot Operating System (ROS) `msg` mechanisms), workload redistribution between several computers, agile integration of new algorithms, debugging and benchmarking made easier, use of common visualization tools. The algorithms integrated in our architecture and using `PPL`s are called `PeoplePoseList` Publishers (`PPLP`s).

Several state-of-the-art techniques exist for detecting human shapes in an image and were integrated as `PPLP`s. In addition, the use of a depth imaging device (Kinect) can often improve their performance by removing outliers. The HOG detector ([Dalal and Triggs, 2005]), for instance, and the Viola Jones face detector ([Viola and Jones, 2001]) were improved that way. Other state-of-the-art algorithms such as Polar-Perspective Map (PPM)-based detectors ([Howard and Matthies, 2007]), originally developed for pedestrian detectors in autonomous cars, tabletop detectors, and the Kinect official API, called NiTE ([Berliner and Hendel, 2007]) were also integrated. All algorithms were carefully tested and benchmarked.

Some algorithms turned out to perform notably better than others and all of them have a domain of use: the face detection `PPLP` works fine for close users, but has a high miss rate for remote ones, the tabletop and the PPM detectors are made for standing users and non clustered environments as they do not discriminate humans and other shapes with human-like sizes.

A clever fusion system would then take advantage of each detector when it is the most relevant, and discard its output when it is out of its domain of use. This point will be further studied in chapter 7. For now, the following chapter 4 will present techniques for user detection based on other fields than image processing and computer vision.

CHAPTER 4

Other techniques for person detection

Introduction

In the previous chapter, we have seen that many different approaches exist to detect humans around a robot using algorithms on the flow of images given by cameras. Even though the work presented in this PhD is mainly focused on methods using this visual information, other approaches, using different sensors than cameras, to detect users. These approaches are studied in this chapter. They do not represent the main component of the final work presented in this dissertation but are a good proof of the modularity of the proposed architecture concerning the hardware devices and the algorithms used.

For instance, microphones can be used to detect and localize human voices. Human users can also indicate their presence using special tags, may they be visual (like bar codes), or based on wireless chips. Information about the distance of the closest objects around the robot can also be used to try to find the users.

4.1 State of the art

The state of the art will focus on the fields listed in the introduction part, as they correspond to the ones we have explored more in depth to meet the goals of this PhD.

First, in subsection 4.1.1, we will review different techniques based on recognizing some information added on the user, also called tags. Second, in subsection 4.1.2, we will present

related work dealing with detecting and localizing human voices thanks to microphones. Finally, in subsection 4.1.3, we will review some innovative techniques based on detecting the legs of the users thanks to a laser range finder.

4.1.1 Tag based user detection

In this chapter, we aim at detecting the users around a robot. The previous chapter focused on achieving this goal using the visual information given by the cameras without needing any additional action of the user: no extra-device, no special clothing was required for her detection. In this subsection, we will present an alternative method: adding some extra-information that identifies the user uniquely, called **tags**. They are similar to barcodes for products in a shop, and can be based on a variety of mechanisms. We will here present both tags based on vision and on radiofrequency identification (RFID).

4.1.1.i Visual tags: ARToolkit, QR codes, etc.

This part focuses on visual tags, i.e., tags that can be detected by vision. In other words, the method consists in attaching an additional physical object to the user, shaped as a label. The content of this label can easily be recognized by some algorithm.

Barcodes Barcodes are widespread labelling tags mainly used for the recognition of products. The barcode technology was developed in the fifties, originally for recognizing items in a food chain, by Norman Joseph Woodland and Bernard Silver. The first item scanned was a packet of chewing gum in an Ohio supermarket in 1974.

A barcode is originally a pattern made of vertical lines. Lines of two different thicknesses encode a binary pattern. As there is no variation in the pattern vertically, this kind of barcode is one-dimensional (1D). Since its invention, the barcode has known many developments: many two-dimensional (2D) patterns (the pattern varies horizontally and vertically) now exist, such as QR-codes, AZTEC codes, and others. Smartphones now commonly sport a barcode reader using the embedded camera. Note that most barcodes are based on a binary encoding of the information, or in some cases on a low-dimensional encoding (base inferior to 5). Due to its simplicity, the barcode has been the main method for product identification over 40 years. They have also often been used for Human-Robot Interaction (HRI), as they make possible the detection of meaningful patterns easily by the robot. For instance, in [Katsuki et al., 2003], QR-codes are put on top of objects to help a manipulator robot finding the right object to grasp in home or office environment. Some samples of barcodes are visible in Figure 4.1.

However, barcodes reader are made for close range identification. Their use for people identification is challenging if the users are far away. On top of that, barcode readers cannot specify the spatial position of the label relative to their own frame. A label system, similar in concept with barcodes, was conceived to solve these issues, called *ARToolkit*.

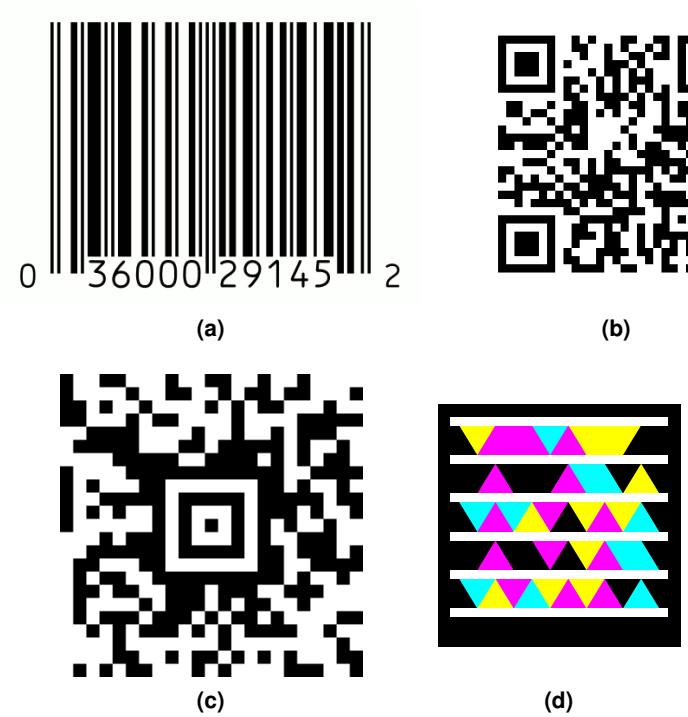
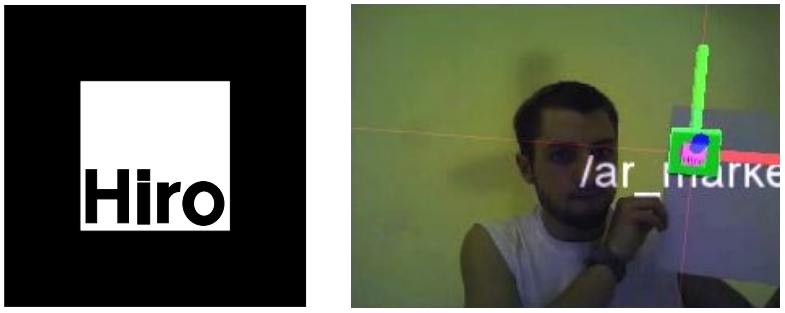


Figure 4.1: *A few samples of barcodes.*
(a) : a classical barcode;
(b) : a QR-code, the most widespread barcodes for smartphones;
(c) : an Aztec code, used by many railroads companies for their online bills, such as Deutsche Bahn, SNCF, Eurostar, Trenitalia and others;
(d) : a High Capacity Color Barcode, a barcode developed by Microsoft, and in this case, encoding information with a base 4 (black, cyan, yellow, magenta).

ARToolkit **ARToolkit** ([Kato and Billinghurst, 1999]) is an open-source cross-platform software suite aimed at creating Augmented Reality interfaces. It was initially developed by Dr. Hizokazu Kato from Osaka University in 1999 for Augmented Reality (AR) purposes, hence the name. It is now maintained by the Human Interface Technology Laboratory (HIT Lab) of the University of Washington. The latest version dates back from 2007.

The job performed by ARToolkit is to search pre-defined patterns in the input stream of images. These fiducial markers, called **Patterns**, are greyscale images of squared dimensions. Each marker is unique and asymmetric. A sample marker is visible in Figure 4.2 (a). The internal vision-based algorithms of ARToolkit locate these visual markers in each input image. For each detected marker, using the size and perspective of this markers, ARToolkit can estimate its three-dimensional (3D) position and orientation. As such, information about the position in the environment of the marked objects is obtained. On Figure 4.2 (b) the orientation of the pattern is represented overlaid on a sample of input frame.



(a) *A sample pattern.*

(b) *The input frame superposed with the relative orientation of the marker.*

Figure 4.2: *A sample ARToolkit pattern and a sample use case.*

ARToolkit is a stable library, who has been in constant development and improvement for many years. Furthermore, it is a very popular library, it has been downloaded more than 450.000 times since 2004. We led some tests to measure the accuracy of the library and they showed some impressive recognition rates and correctness concerning the position of the camera. Furthermore, the library has a very fast processing time: ARToolkit is efficient enough to enable real-time processing even with embedded processors.

There are many examples of articles using ARToolkit tags. We can notably cite [Dudek et al., 2007], in which a diver operator can operate an underwater robot using ARToolkit tags. This eases the task of locating the operator, which is otherwise challenging because of the light conditions and the water distortion. In [Anezaki, 2011], the same method is applied to control a ground robot, except that QR codes are used.

However, as pointed in [Fiala, 2005b], the binarization process of ARToolkit is made through a simple binary threshold, which leads to detection failures for challenging light conditions.

ARTag ARTag is another kind of fiducial markers ([Fiala, 2005a]). It also uses some planar markers to indicate planar surfaces. Its development started after ARToolkit, and it aims at solving some of the limitations of it.

It replaces the simple threshold technique by some four sides contour finding. This most notably solves some of the problems affecting ARToolkit, such as partial occlusions and inter-marker confusion. as shown in [Fiala, 2005b]. A system such as ARTag was proven robust to poor lightning condition.

Markerless AR techniques Both ARTags and ARToolkit rely on visible, predefined markers that are detected by the algorithm. These markers are explicit: they are also visible by a human observer watching the scene. There is another kind of marker that enables the markers to be ”hidden”: just as the invisible ink cannot be detected by an observer, these markers are integrated

into another image in an invisible way for the human eye. There is a variety of techniques, check for instance [Comport et al., 2003].

For the work of this PhD, such techniques could be interesting, because the produced markers are not as explicit and unaesthetic as the monochrome markers of ARToolkit and ARTag. This aspect could be interesting for helping the social acceptance of markers as a widespread recognition system. However, the scope of this work is a lab environment, and not to use markers among a wide population.

In the past few paragraphs, we have presented techniques for detecting the user thanks to visible tags. They rely on widespread image analysis libraries and give real-time 3D position of the tags, and hence of the users. However, theses tags need to be visible: in other words, any occlusions of the card will prevent the tag from being read, and then the user to be recognized.

4.1.1.ii Radiofrequency recognition tags: Radio-Frequency Identification (RFID)

Other technologies do not need to have a visible marker. We here present the Radio-Frequency Identification (RFID) technology. It enables the recognition of an object remotely. RFID labels do not need to be visible to be read, even more, they can be read under a thin layer of paint, snow, etc. Some labels can be written once and read multiple times, while others are writable several times.

There are two categories of labels: passive and active. Passive labels are activated by the energy transmitted by the radio waves. Active labels have their own energy source, such as a battery, that enable them to have a greater range.

RFID tags are often used in automation and robotics for object and user recognition. The amount of data stored on the RFID tag, although not huge, is enough to store a unique ID for a great population of objects or users. Thus, [Oberli et al., 2010] discusses the possibility of using RFID for passenger recognition and public transport. [Vogt, 2002] uses RFID tags for multiple object detections at once. Ultra High Frequency RFID tags are used in [Deyle et al., 2014] for navigation, by using the signal strength to guide the robot.

Note that, unlike ARToolkit tags seen before, RFID tags are limited in the sense that they only give the presence or not of the tag when detected, but not their spatial (3D) position . As such, they can be used for user detection and recognition, but not for mapping. Furthermore, they need an additional device, the RFID reader, that can be fairly bulky and needs additional power, which can lower the energy autonomy of the robot. And finally, the reach of RFID readers is limited: most RFID passive tags can be read to a distance up to one meter. Note, however, that a passive ultrahigh-frequency (UHF) transponder can be read from three to six meters feet in free air (that is, with the tag not on an object).

Overall, RFID tag-reading can be a way of detecting and recognizing users, and is handy from a hardware point of view, as our robots are already equipped with a RFID reader, but it is limited by nature. We did not implement a RFID tag-based user detector.

4.1.2 Voice detection

Human voice detection The previous chapter focused on detecting users around the robot thanks to their visual appearance, but there is another sense that the humans use extensively to locate other people: hearing. This aspect was not dealt with extensively in the work presented in this thesis, as some other researchers are tackling it in our team. We will however briefly review here some of the techniques.

Voice activity, which is the presence of voice in an audio stream, can be achieved using a set of different features and metrics. Some simple systems are based on a volume threshold activation and shutdown: if the volume of the microphone goes over a given threshold, voice analysis starts, until the volume goes below that threshold for a given shutdown time (usually between half a second and two seconds). This system is straightforward to implement, but error-prone: no distinction is made among the perceived sounds.

To help discriminating human voice from other sounds, we can make use of **Audio features**. These are numerical values characterizing some property of the audio stream during a given time lapse. A simple example is the sound pressure level, which is the logarithmic measure of the effective sound pressure of a sound relative to a reference value (this reference being the threshold of human hearing), and is expressed in decibels.

These audio features help us discriminating voices among other noises. Even if it shows a great diversity among humans, the human voice can be characterized with different properties. A work of our team, [Alonso-Martin and Castro-González, 2013], listed a range of meaningful audio features for voice detection. The used features are *Pitch computed using Fast Fourier Transform* (frequency domain); *Pitch computed using Haar Discrete Wavelet Transform* (time-frequency domain); *Zero-crossing rate (ZCR)* (time domain). *Centroid* (frequency domain); All of these features are obtained thanks to mathematical transformations of the input audio stream. Statistical analysis of dataset of both voice and non-voice samples helped determining ranges of values for these features that characterize human voice.

Sound localization A first method to localize the spatial source of a sound is based on the amplitude of the signal received by all microphones: the microphone which is oriented in the direction of the source should receive a signal with a more important amplitude than the others.

In [Blauth et al., 2012], a microphone array gives away the sound source location. It makes use of the Steered Response Power with the phase transform (SRP-PHAT) method. A HMM determines if there is silence, and another if a user is speaking. Face detection and tracking gives the estimated number of speakers and the three-dimensional (3D) position of their face center. The fusion is fairly simple: the final probability of the user being at one of the discretized positions is the estimate by the audio HMM, weighted by the distance between this position and the output positions of face tracking.

However, this theory is compromised by the nature of the sound: the sound wave is omni-directional, and its multiple rebounds on the walls, the ceiling and the floor will induce higher

volumes for the other microphones. Furthermore, each microphone has a different sensibility to noise, even for identical brands, models and series. The determination of the transformation on the amplitude that gives identical measures for a given output, called **Calibration**, is cumbersome and difficult to achieve.

Another method is based on the phase analysis of the audio signal: all the microphones receive more or less the same signal, but with a delay induced by the sound traveling speed. As such, the microphone that received first the sound phase profile is the closest to the user ([Brandstein and Ward, 2001, Benesty et al., 2008]). However, the positioning of the microphones is challenging: if the microphones are too close, they receive almost the same signal. If they are too remote, the phase profile might differ too much to compute its delay.

4.1.3 Leg pattern detection

Viewing people face or hearing their voice is a way to detect their presence. However, some other sensors of the robot can be used for this goal. The embedded cameras of the robot give indeed visual (color) information, but no information about the distance of the objects, in other words, the depth of the scene [1]. On the other hand, a class of devices known as **Laser range finders** gives us information about the spatial structure of the scene. It is characterized by a two-dimensional measurement plane and gives, for a range of angles, the distance to the closest object in this plane in each direction, in meters. The device is limited by its angular range, called Field of View (FOV) and its distance limit. Some devices, such as the Sick LMS200, are made for outdoors laser measuring and have a maximum range of 80m and a FOV of 180 degrees, while smaller devices such as the Hokuyo URG-04LX have a wider FOV of 240 degrees but a range of only four meters. Most commonly, the laser range finder is mounted on the mobile platform so that its scan plane is horizontal. That gives us a representation of the closest objects in all directions in this horizontal plane.

Laser range finders are one of the most widespread sensors in robotics for decades. Their accuracy is most often millimetric and their processing is easy, as it is structured as a one-dimensional (1D) array. It gives a two-dimensional (2D) perception of the world though: each distance value of this array is implicitly associated with an angular value, which results in a 2D point. Laser scans have been used in many fields in robotics, such as navigation and obstacle avoidance. Using them for people detection is challenging though: recognizing human shapes from an horizontal slice of the environment is not easy. However, the works presented in [Bellotto and Hu, 2009] tackle this problem. The robotic platform they use is a mobile platform that is about half a meter high, so that the laser range finder sees is about the height of human knees. They have developed a people detector based on the analysis of laser scans: typical leg patterns are searched in the scans are recognized within the scans. The information is then merged

[1] Some sensors have been able to give this information for many years. Stereo-vision for instance supplies depth maps of a good quality, but is limited by a challenging calibration, the need for textured environments, high computational needs, and its cost. Time-of-flight cameras or structured-light devices, such as the Kinect sensor, can give depth information.

with some other algorithms (face detection) using some Kalman filtering, a method that will be presented in chapter 7. An illustration of this method, taken from the original article, is visible in Figure 4.3. Note that this technique is relatively straightforward, gives relatively reliable results according to the article, and makes use of a sensor that otherwise is hard to use for people detection. These reasons explain our interest in this method, this is why we will explain later on in this chapter how we integrated this method in our architecture. Note that several subsequent papers re-use this method, see for instance [Mozos et al., 2010, Varvadoukas et al., 2012].

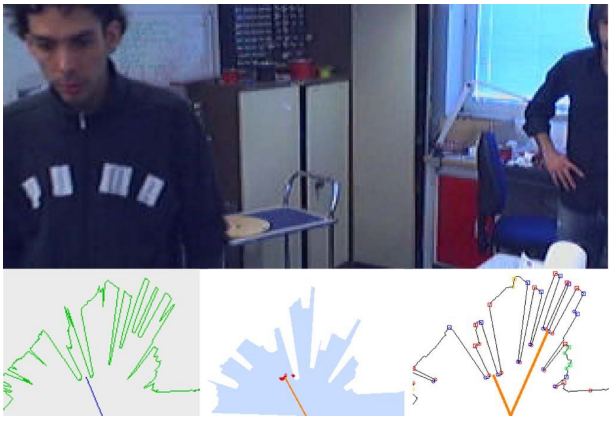


Figure 4.3: *Human leg pattern detection from laser scans, image from [Bellotto and Hu, 2009]. On top, the color image as viewed by the webcam. Three different algorithms are compared here: minima on the left, motion in the middle, and leg patterns on the right. Note that the leg patterns algorithm, the one proposed by the authors, is the only one able to detect both users.*

4.2 Research contribution

We have seen in the previous section some state-of-the art techniques concerning user detection using algorithms not based on the visual appearance of the user, namely tag-based user detection, voice detection and leg patterns in laser scans recognition. Our contribution in that field consists in experimenting these different fields thanks to our robotic platforms, and in integrating them into our architecture thanks to . the common framework for people detectors called `PeoplePoseList` Publisher (`PPLP`) presented in the previous chapter. In a first part, in subsection 4.2.1, we will present how one of the tag-based detection and localization algorithms, namely ARToolkit ([Kato and Billinghurst, 1999]), was integrated and benchmarked. Then, we will present two detection mechanisms that do not require extra device from the user: in subsection 4.2.2, we will present some experiments we led on voice localization using the `PPLP` framework, and in subsection 4.2.3, we will present the integration of the leg pattern detector from laser scans of [Bellotto and Hu, 2009] in our architecture.

4.2.1 ARToolkit `PeoplePoseList` Publisher

We presented in subsection 4.1.1 different methods for recognizing human users thanks to ad-hoc methods. Among others, the ARToolkit tags ([Kato and Billinghurst, 1999]) had several advantages: they are easy to produce (printed pattern), they only require a standard RGB camera, and the computational cost needed by the algorithm is low.

This is why we chose to convert the ARToolkit Robot Operating System (ROS) wrapper, supplied by Gautier Dumonteil [2], into a `PeoplePoseList` Publisher.

Customized tag building we saw in subsection 4.1.1 that ARToolkit searches in each frame pre-defined markers, called patterns, in the input stream of images. Patterns are encoded with so-called **Pattern files**. A pattern file is a text file, representing the visual appearance of the pattern, seen from above. However, ARToolkit does not document how to produce such pattern files. It is only shipped with a variety of predefined patterns, and with an executable to build new patterns from the camera input. This executable detects black rectangles in the input image and converts them into pattern files.

By examining the produced files, we could determine the format of a pattern file. It contains a stream of space-separated integer values, representing the brightness of of each pixel. Most patterns are of size 256×256, so such a pattern file contains 256×256 space-separated integers between 0 and 256. ARToolkit also works with color patterns, in that case the pattern files have three streams of values. The analysis of the patterns produced by the camera application lead us to the conclusion that patterns produced that way are blurry and imperfect due to the rectification of the detected pattern from the perspective to the square point of view. This imperfect conversion results in a poor detection rate when we later use this pattern.

For this reason, we analyzed the format of the pattern files and wrote a small program that converts any square image into a pattern file using OpenCV. It reads the image file, and converts it into space-separated integers files that are compatible with OpenCV. This program is also compatible with color images. As thus, as our custom patterns are generated by printing a custom pattern image (in our case, designed with the drawing program Inkscape [3]), we can directly use this source image to build an accurate pattern file that does not include any deformation or blur.

Such a tool, even if is simple, is of key interest for generating meaningful markers: instead of using cumbersome monochrome square matrices, we can generate explicit markers that can be understood by users. A sample custom pattern is visible in Figure 4.4. In Figure 4.5 (b), we can see two custom ARTags that can be understood by human users: one contains the text ”Alice”, the other ”Bob”. In other words, we can generate for each user a pattern that is customized for that specific user.

Once we have this key functionality of accurate custom pattern generation, we can focus on

[2] `http://wiki.ros.org/artoolkit`
[3] `http://www.inkscape.org/`

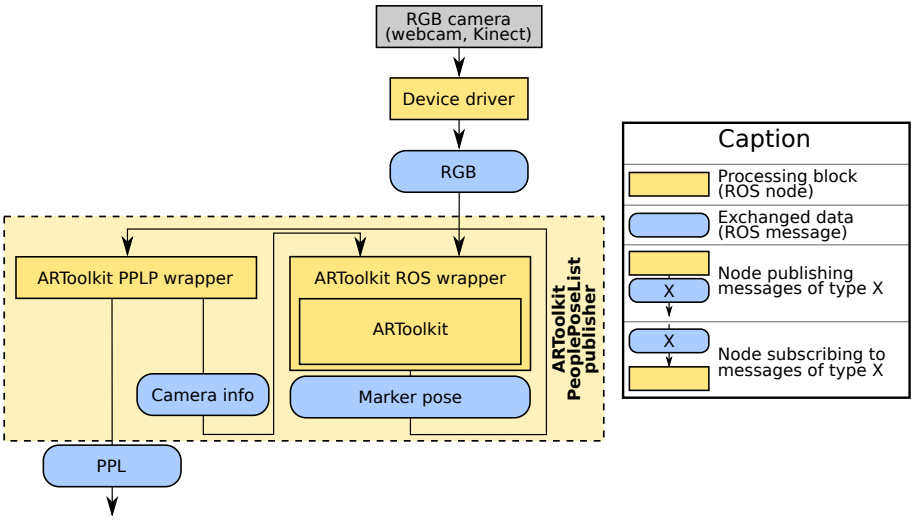
Figure 4.4: *A custom ARToolkit tag for the use Alice.*

integrating the original ARToolkit pattern detector into our use awareness architecture: this is done by wrapping it into a `PeoplePoseList` Publisher (`PPLP`).

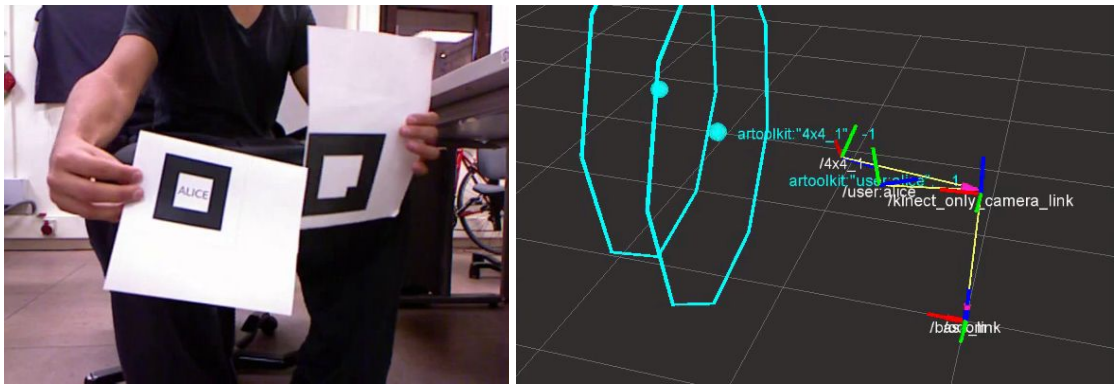
`PeoplePoseList` **Publisher wrapping:** ARToolkit is a standalone library. The original ROS wrapper of ARToolkit previously mentioned subscribes to an image stream and publishes the detection results: this message contains the name of the labels that were recognized by the algorithm (i.e. the pattern filenames), along with their 3D position. There is little work needed to convert this information into a proper `PeoplePoseList` (`PPL`). The ARToolkit `PPLP`, at startup, reads from a data file the correspondence map between pattern filenames and unique users ID, and subscribes to the ARToolkit ROS wrapper output. Then, upon reception of a new ARToolkit message, it converts it into a `PPL` using this map. The processing pipeline of the system is visible in Figure 4.5 (a). A sample capture of the simultaneous detection of several tags is visible in Figure 4.5 (b).

This system offers several advantages. First, it provides our architecture a reliable estimate of the three-dimensional (3D) position of the different users and of their identity thanks to the matching between pattern and person ID, while needing only a two-dimensional (2D) color stream, and without any depth information. It then outperforms the other 2D detectors we saw in the previous chapter, namely the face detector and the Histogram of Oriented Gradients (HOG) detector: these `PPLP`s, if feeded with a 2D image feed, only provided an estimate of the 2D position of the users. Second, the computational needs of this `PPLP` node is very limited, and ARToolkit itself is a highly optimized library. For both these reasons, the whole ARToolkit `PPLP` is very lightweight.

There are two drawbacks to this method. First, it requires the users to change their behavior to be recognized: they need to wear the patterns and have them visible, which is a counter-intuitive and somewhat cumbersome process: if the user forgets her pattern, she cannot be recognized by the robot. Second, ARToolkit obviously cannot detect tags that are not visible or very deformed, as seen from sideways. As such, if the user wears her logo on the torso and is not facing the robot, the pattern cannot be detected, so neither the user. Note that this problem could be overcome by using several tags worn by the robot from several points of view.



(a)



(b)

Figure 4.5: *ARToolkit* `PPLP` *using custom tags.*
(a): the pipeline of the ARToolkit `PeoplePoseList` *Publisher.*
(b): simultaneous detection of several tags.

Conclusion the ARToolkit `PPLP` is a lightweight user detector and recognizer. It uses the color image stream, which makes it compatible with a robot having no depth imaging sensor. It gives a reliable estimate of the 3D position of the users and their identity. However, it requires the users to wear a cumbersome pattern file and to have it always visible if they want to be recognized.

4.2.2 Voice localization `PeoplePoseList` Publisher

Whereas the previous chapter focused on detecting human activity thanks to the visual information of the scene, humans can also be naturally detected thanks to the sounds they produce, and especially the sound of their voice. Indeed, a human user willing to interact with a robot platform will most probably greet it or start talking to it. Compared with the tags approach seen in the previous section, voice activity is a more natural approach in terms of Human-Robot Interaction (HRI), as it corresponds to the way a user would interact with another person. Voice Activity Detection (VAD), i.e. determining with certainty if a user is speaking in the sound stream, is a problem that has been tackled and solved by many authors, as seen in subsection 4.1.2, page 72.

We do not tend to tackle the problem of VAD here. Readers interested in this topic can refer to other works published by members of the team, such as [Alonso-Martin and Castro-González, 2013]. Here, on the other hand, we focus on the next stage of the scene sound analysis: once we know a user a speaking, the following problem is to localize her spatially. *We here consider the simplified case where there is one user speaking, so that the source of the loudest sound corresponds to this user*: it is adapted to situations where there is only one user and she is speaking, or, in multi-user scenarios, this hypothesis ensures the detection of the speaking user only. This voice localization problem is related to a field of signal processing called *sound source localization*. As we saw in the review of the state-of-the-art, some algorithms exist that can locate a sound source thanks to the analysis of a single microphone. However, most solutions use systems of several microphones.

Our robot Maggie, presented in subsubsection 1.3.1.i, page 5 is equipped with 8 directional microphones located around the base. They are positioned in a uniform way, so that there is a microphone every 45 degrees. This is visible in Figure 4.6.

Naive method: average sound pressure-based algorithm As presented in subsection 4.1.2, page 72, we can use the amplitude, also called the sound pressure, of each microphone to know where the sound comes from. Thanks to ALSA audio drivers, we can acquire in real time the pressure of each microphone. The average sound pressure over a given short time lapse (half a second) estimates the volume from each microphone. The microphone which obtains the highest volume during several successive iterations is then considered as being the closest to the sound source.

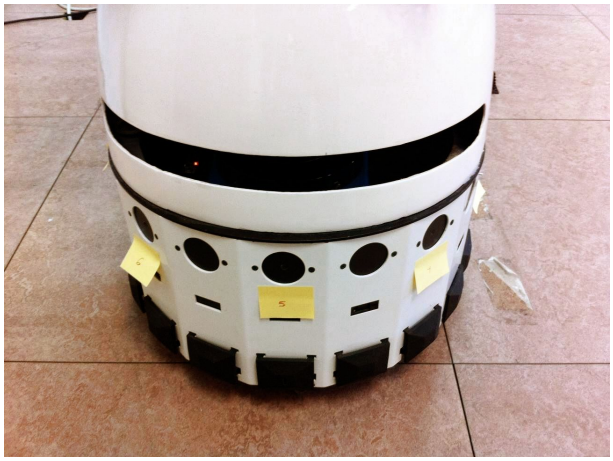


Figure 4.6: *The microphones situated in the base of Maggie.*

Machine-learning based techniques The method based on the average sound-pressure is limited by the different effects explained in subsection 4.1.2, page 72: the rebounds of the sound wave on the floor, the ceiling and the walls will induce artificially high volumes on all microphones. Furthermore, the difficult calibration of the microphones does not ensure that a microphone having a higher amplitude than its neighbor is closer from the input.

This problem can be tackled with Machine Learning (ML) algorithms: collect a lot of data samples, where each data sample consists of the output of all microphones, together with the ground truth user position; then train a regression algorithm on this data to predict the user position given the microphones output.

Formally speaking, if the robot has n microphones, $n \in \mathbb{N}$, then the input of the ML is a vector v of size $n \times 1$, and the output is the position of the user. It can be represented as a three-dimensional (3D) point $x \in \mathbb{R}^3$. In our case though, we prefer using the polar coordinates to describe it, $(r, \theta), r \in \mathbb{R}^+, \theta \in [0, 2\pi[$, and we only try to estimate the angular position θ. [4]

ML solves the calibration problem: the different responses of the microphones to the volume can be learned by the algorithm, which will give more weight to the microphones that are less reactive. For instance, if a microphone has an amplitude response twice as weak as another, the importance given to the former is likely to be twice as a strong as the latter.

We used two different ML regression algorithms for training and prediction: RandomTrees ([Breiman, 2001]) and Support Vector Machines (SVMs) ([Cortes and Vapnik, 1995]). Note that these algorithms can be trained in parallel using the same data without any interference. The pipeline of both training and prediction processes are indicated in Figure 4.7.

For training, the ground truth position of the user can be obtained thanks to two different algorithms. The first method is based on the NiTE `PeoplePoseList` Publisher presented in

[4] Indeed, the distance of the user r is hard to estimate with microphones: a first user and another further from the microphones but speaking louder will generate a similar signal.

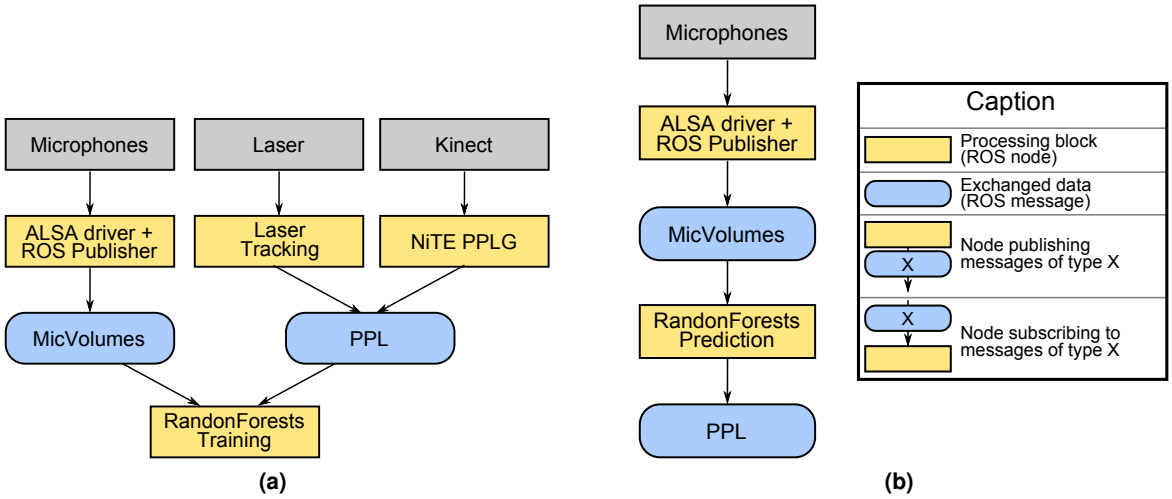


Figure 4.7: *Sound localization with ML techniques. Pipeline for (a), training, and (b), prediction.*

subsection 3.2.5, page 53. The results of the user estimation are exact, but limited by the very narrow field of view of the Kinect, around 60 degrees. This problem is overcome by the second method: we used a cluster-tracking algorithm based on the two-dimensional (2D) laser scans. They are provided by the SICK laser range finder, with a field of view of roughly 270 degrees. The user initializes the blob-tracking algorithm when she stands next to it, then the algorithm tracks this blob when she moves.
Synchronized ground truth position (shaped in a `PeoplePoseList` message, see subsection 3.2.2, page 41) and output of all volumes are sent to the ML algorithm, namely SVM or RandomTrees.

For prediction, the trained ML algorithm subscribes to the output of all microphones and gives an estimation of the position of the user, shaped as a `PeoplePoseList` (`PPL`).

Benchmarking For training, we collected samples of synchronized (user position, vector of 8 microphones outputs). To generate them, the robot Maggie was located in a quiet lab ambient noise. A unique user walked around it, and generated noise either by speaking at different volumes, or by singing. The user stayed in the field of view of the sensor, as presented before. 136 samples where saved, which represents roughly two minutes of recording. The synchronized ground truth position and volume arrays were saved in a text file. The θ values span from -80 to $+97$ degrees (mean of 8 degrees, standard deviation of 50 degrees), i.e. they almost span the full visible FOV of 180 degrees. The ML algorithm was trained on this data.

For testing, another independent set of values was collected using the same process. This test set contains 138 values of synchronized ground truth position and volumes array. Once trained, both ML algorithms, RandomTrees and SVMs, were run on this test set. The naive method, which sets the microphone with the highest volume as the source of the sound, was

also used. We also run all these algorithms on the training set: for the naive method, as there is no training associated, it gives us more test data, and for both regression algorithms, we can thus ensure the regression was coherent. The results of this benchmark for each algorithm are collected in Table 4.1.

Algorithm	Naive	SVM	RandomTrees
Prediction angular error on training set	0.87 rad = 49°	0.10 rad = 6°	0.31 rad = 18°
Prediction angular error on test set	0.91 rad = 52°	0.47 rad = 27°	0.38 rad = 22°

Table 4.1: *Benchmark results for voice localization using machine-learning methods.*

We can first observe that the naive algorithm, based on the maximal volume, is not a very reliable method to determine the sound source: the angular error on both training and test sets is over 45 degrees. This angular error is considerably reduced by using a regression algorithm. Note that the SVM fits very accurately its training set, but the error on the test set is somewhat higher. On the other hand, RandomTrees obtain the most accurate prediction law for the test set, with an error about 20 degrees.

Conclusions: we built a `PeoplePoseList` Publisher (`PPLP`) based on voice localization. Voice Activity Detection, presented in [Alonso-Martin and Castro-González, 2013], indicates when the user is speaking. We focus on the case where there is one user speaking, and we want to determine her position. The robot Maggie is equipped with 8 microphones pointing at different directions, which provide simultaneous volume arrays. Sound localization is obtained thanks to machine learning techniques: first, we obtain a great number of volume samples synchronized with the ground truth position, which is obtained thanks to another `PPLP` (based on laser scan analysis). Second, two ML algorithms are trained on this data: RandomForests and SVMs. Third, their accuracy is evaluated on an independent test set. The naive method, which sets the sound source to the direction of the microphone with the loudest volume, has an angular error over 45 degrees. On the other hand, the angular error of the SVM is under 30 degrees and of the RandomForest under 25 degrees: these regression algorithms are reliable `PPLP`s. Furthermore, they illustrate the modularity of the `PPLP`s system, mainly focused at vision-based detectors but perfectly adapted for other classes of algorithms.

4.2.3 Integration of the leg pattern detector and benchmark

We saw previously, in subsection 4.1.3, how the laser scans provided by two-dimensional laser range finders can offer some advantages concerning people detection that overcome limitations of vision-based techniques. Indeed, the information being structured and associated to a metric measure, we can have a direct understanding of the scene in front of the robot in the laser plane. Furthermore, their wide Field of View (FOV) ensures a perception of the environment to a greater extent than cameras, that typically have a FOV of roughly 70 degrees. Most laser range finders

being typically mounted at the level of the legs of the users, i.e. about forty centimeters high, user detection is made through the detection of their legs.

Many leg pattern detection algorithms exist. We chose the one described on [Bellotto and Hu, 2009] for its simplicity and the overall good performance claimed by its authors. It was integrated as a `PeoplePoseList` Publisher (`PPLP`). As an input, it subscribes to the laser scans acquired from the laser range finder. It then performs the leg pattern detection described in the original article, that we reimplemented in C++ using the original description. Each found pattern is then converted into a `PeoplePose` (`PP`), using the (x, y) coordinates of the pattern and setting a constant z height of 170 cm. All the `PP`s are then grouped in a `PeoplePoseList` (`PPL`) that is published to the rest of the architecture. This pipeline is visible in Figure 4.8.

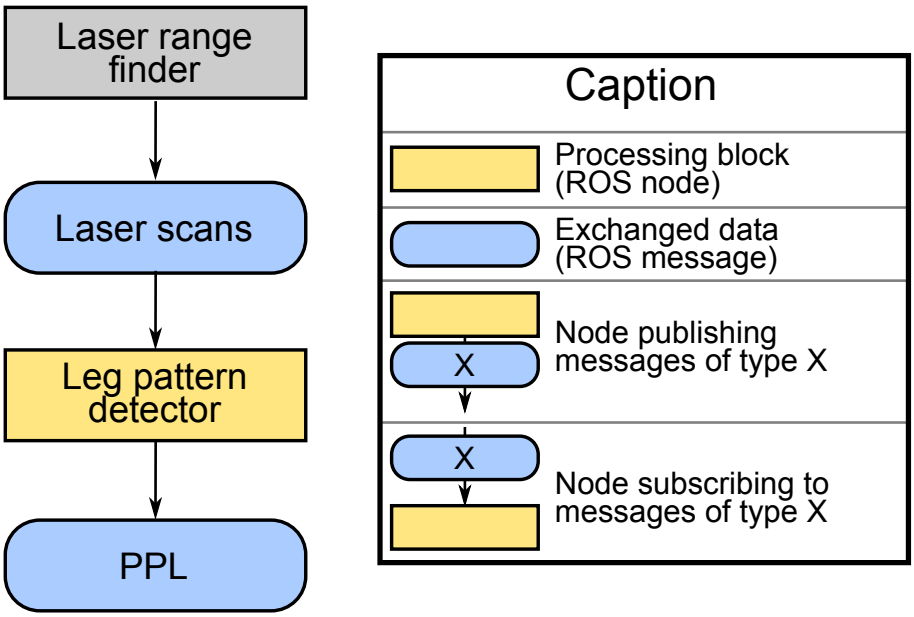


Figure 4.8: *The pipeline of the ARToolkit `PeoplePoseList` Publisher.*

Benchmarking: To evaluate the performance of the leg pattern-based `PPLP`, we needed a dataset of test laser scans. The DGait dataset ([Igual et al., 2013]) used for benchmarking the vision-based detectors do not include laser scans, only raw data from the Kinect. Although there are techniques that allow the conversion of a depth image into a laser scan, the position of the camera, mounted on top of a cupboard, and the narrow field of view of the Kinect, do not result in data that are similar to a laser range finder mounted on top of a robot.

For this reason, we acquired real laser scans from a Hokuyo laser range finder mounted on top of one of our vehicle. During the acquisition, a user walks randomly around the robot, using the whole 270 degrees of field of view. In average, we kept 3.5 scans per second that we saved in a CSV file. In total, 139 scans were saved. They were then manually annotated: the ground-truth user 2D position was marked on each scan thanks to a Graphical User Interface (GUI). The

final CSV file contains both the scans and the positions of the user, when visible. This dataset is licensed under the terms of the GNU General Public License Version 2 as published by the Free Software Foundation, and freely available to download on the author's website, along with images and videos [5].

The leg-pattern detector is then evaluated on this dataset: each scan is sequentially read from the file by a CSV parser and published to the ROS architecture. The leg-pattern `PPLP` subscribes to this data stream and perform the detection as described above. The resulting `PPL` is then compared against the ground-truth annotation, also available in the CSV file. The results are gathered in Table 4.2.

True positives	99
True negatives	0
False positives	703
False negatives	0
Hit rate	100%
Accuracy	12.3%
Position error per true positive (m)	0.306

Table 4.2: *Benchmark results for the leg detection`PeoplePoseList` Publisher (`PPLP`).*

We can observe that the leg-detector `PPLP` always detects people in its surroundings: the hit rate is 100%, which means that all ground-truth user occurrences were correctly detected. However, it has a low accuracy: the number of false positive is very high, which means that many leg patterns are detected in the background objects, resulting in a poor accuracy. Indeed, we can understand that it is challenging to detect users only by the processing of an horizontal layer of their three-dimensional (3D) shape: legs can easily be confused with other geometrical shapes of similar appearances such as trash bins, lamps and so on.

Conclusions on the leg pattern-based `PPLP`: To conclude, we integrated into our user awareness architecture a `PPLP` based on the analysis of laser scans, in which it tries to find leg patterns. This leg-detector based `PPLP` has been evaluated through a benchmark on data we acquired and that we make available to the community. It is characterized by a very high hit rate, but a poor accuracy. This performance can be improved by coupling this detector with other detectors having a better accuracy: this will be considered in following chapters.

[5] `https://sites.google.com/site/rameyarnaud/research/phd/legs-in-laser-scans-datset`

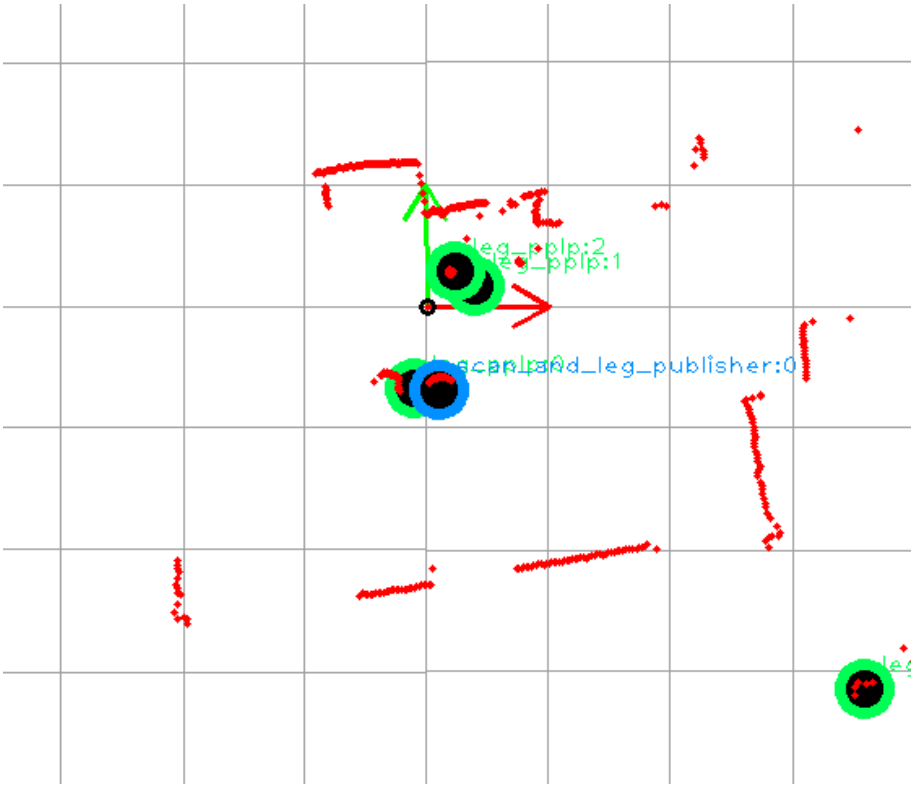


Figure 4.9: *A capture of the benchmark of the leg detector* `PeoplePoseList` *Publisher. The arrows correspond to the position of the robot (the origin of the coordinates frame). In red, the current laser scan. In blue, the ground truth position of the user. In green, the users detections: four detections are visible, the one in the middle being a true positive and the three remaining false positives.*

Summary of the chapter

This PhD tackles the problem of giving user awareness to social robots. This local mapping of the users can be split into three sub-tasks: user detection, user recognition and user mapping. This chapter focuses on the first sub-task, user detection, using methods that are not based on image processing, dealt with in the previous chapter. Indeed, there are a variety of sensors and algorithms that can help detecting users. Our contribution in that field is made of three elements.

First, we explored the field of tag-based user detection. Among the different tag possibilities (barcodes, Radio-Frequency Identification (RFID) tags, IR tags, etc), we chose a specific class of visual tags, called ARToolkit tags. They are monochrome tags, each of them different from the others. They can easily be detected by an algorithm, which also returns the three-dimensional position and orientation of the detected tags. Although the detection and recognition accuracy are good, the system is limited by definition: the markers turn out to be cumbersome to wear in a realistic setting and they need to be visible by the camera, which is hardly compatible with user motion. We integrated the original algorithm into our architecture, shaping it as a `PPLP` (defined in the previous chapter).

Second, we re-used the work done by other members of the team in voice detection. Voice Activity Detection (VAD), which consists in determining if a user is speaking or not thanks to the audio stream, was already tackled by previous articles of the group. We built on top of this feature a voice localizer, that determines the angular position of the speaking user relative to the robot center using an array of eight microphones located around the robot. The algorithm is shaped as a `PPLP` and is based on machine learning techniques, namely RandomForests and Support Vector Machines (SVMs), which do not need explicit laws between each mic volume and the position of the user, and is modular by approach: after a change in the microphone configuration, the process of collecting data and training the algorithm with this new configuration is fast. The voice localizer was tested on both test datasets and live in the robot Maggie. The precision of the estimation of the angular position is good. Perhaps more interestingly, this algorithm demonstrates the flexibility of the `PPLP` interface: it fits as well audio based algorithms as well as vision-based ones.

Finally, a user detection algorithm based on laser scans was implemented. This kind of data is two-dimensional (2D) and by definition, more ambiguous: recognizing users from this kind of data structure is challenging. After a careful review of the state of the art, we selected [Bellotto and Hu, 2009] as a base for a laser-based `PPLP`. The detection is based on the research of several given leg patterns in the laser scan. These patterns correspond to how a pair of user legs are seen from the laser point of view, according to the user orientation. The algorithm was implemented, and tested. It is characterized by a good hit rate (most users are detected), but a high false alarm rate, as static objects are often detected as legs too.

Overall, this chapter demonstrates the modularity of the `PPLP` system, focused on vision-based techniques, but also perfectly adapted for other sensors and classes of algorithms. It also gives contributions in the field of user detection that integrate well with the other capabilities of the robot and make use of the variety of sensors that equip it. Coupled with the previous chapter

based on user detection thanks to vision-based techniques, they provide a range of methods, all respecting the PPLP interface, that suits well the variety of sensors and configurations that can be found in robotic architectures.

Part II

User recognition

CHAPTER 5

Vision-based user recognition

Introduction

It was shown in the previous chapters how to perform user detection for social robots using different methods. These methods can be based for instance on vision techniques, such as face detection, or other devices, for instance, detection of the legs with a laser range finder.

However, for a meaningful Human-Robot Interaction (HRI), detecting the users around the robot is not enough. **User recognition** consists of determining that a given user visible by the robot is in fact a given known user (uniquely identified, for instance with her name, with all her characteristics). User recognition is of key importance for HRI: it enables having a personalized interaction flow with the user based on her preferences, her experience with the robot, etc. User recognition methods have an intrinsic life scope associated: for instance, an algorithm based on the color of the clothes is valid for short term estimations (a few hours), while one based on the visual appearance of the face of the user or her voice will last for months or even years.

Some valuable hints for the recognition of a user can be obtained via **Gender recognition**, which goal is to estimate automatically and robustly the gender of the user. Although it is not a proper recognition technique, because it does not estimate the identity of the user, its information is useful: knowing the gender of an unknown user prunes the possible match [1]. It may also have a direct interest in HRI as it allows the customization of the interaction according to the user genders without resolving their full identity.

[1] For instance, if they are as many women as men in the user dataset, the possible matches are pruned by a factor of two.

This chapter first aims at making a comprehensive review of the existing vision-based techniques for user recognition and gender recognition, in section 5.1, page 90. The topics covered include recognizing the user according to the visual appearance of its face, anatomy-based techniques, and histograms.
Then, in section 5.2, page 102, the different contributions we achieved in this field and their implementations will be presented. They include the building of an image dataset for gender-from-face using innovative techniques; details on the implementation of gender-from-face recognition for the robot MOPI using this dataset; a novel algorithm for height detection, that can be used either for user recognition or for gender estimation; another novel method estimating the gender of the users according to the shape of their torso; and an innovative user recognition algorithm based on structured histograms of the color of their clothes.

5.1 State of the art

User recognition using vision and image processing is a wide field. In this section we will only present the most relevant and interesting techniques for our goals, defined in section 1.2, page 3.

Face and gender-from-face recognition techniques are first presented in subsection 5.1.1, page 91. The former aims at recognizing a new face among known users thanks to a dataset of labeled faces of these users. The latter guesses the gender of a new face thanks to a training on a dataset of both male and female faces. This gender can then be used as a hint for pruning the possible matches of a user recognition algorithm.

However, to obtain a natural interaction with the human user, it is important that the robot gathers information about her. The queries can be explicit, such as questions asked to her or an interactive display, but the range of applications widens if some features can be inferred with no user action required. These features are called **Soft biometrics**. Three important families of soft biometrics will be further presented: physical traits, behavioral traits, and adhered human characteristics.

In subsection 5.1.2, page 94 we will present some soft biometrics techniques for user recognition based on the anatomy of the user: first, her height, which can be challenging to compute when she is not standing straight, and second the shape of her body, which consists of a useful feature for recognizing her.

Then, we will consider how histograms can help us for user recognition. Histograms are a powerful way to represent how data spreads among a range of values, and it comes especially handy for representing color distributions. They can hence be used for user recognition algorithms according to their color distribution, as explained in subsection 5.1.3, page 96.

5.1.1 Face and gender-from-face recognition

In this part, we will present different state-of-the-art algorithms for face and gender-from-face recognition. **Face recognition** consists of recognizing the user thanks to a picture of her face. More accurately, face recognition determines the most likely user label for a given face picture, given a set of face pictures annotated with user labels.

In subsection 3.2.3, page 47, we saw how it was possible to robustly detect faces in the robot's surroundings. We now aspire at recognizing these detected faces against known ones.

In subsubsection 5.1.1.i, we will review the most popular techniques for face recognition. In subsubsection 5.1.1.ii, gender-from-face recognition will be presented as a special case of face recognition.

5.1.1.i Face recognition basics

A facial image can be seen as a point in a high-dimensional space. Indeed, let us consider greyscale images of $p \times q$ pixels. Then each face corresponds to a point in a pq-dimensional space. However, for typical dimensions, such as $p = q = 100$, this very highly dimensional space makes the matching problem very difficult.

The three main face recognition algorithms will be further explained: *Eigenfaces*, *Fisherfaces* and *Local Binary Patterns Histograms (LBPH)*.

Eigenfaces The idea behind **Eigenfaces** ([Turk and Pentland, 1991]) is to determine the dimensions that matter the most, that is, that convey the most discriminative information between faces. To achieve this, the theory of Principal Component Analysis (PCA) is used. An extensive review of PCA is available in [Duda et al., 1995]. PCA helps to transform a set of numerous variables possibly correlated into a smaller set of variables that are uncorrelated, and which are the most meaningful for the description of the set. The PCA method finds the directions in the set with the greatest variance in the data, called *principal components*.

The Eigenfaces approach is based on PCA. Here is a brief summary on how to compute Eigenfaces for face recognition:

1. A small set of face pictures is used to train the classifier, which learns how to differentiate these pictures thanks to the data distribution of these faces.
   More exactly, the classifier extracts eigenvalues and eigenvectors from the covariance matrix of the distribution of these training faces. Then, only the most discriminative eigenvectors are kept, i.e. the ones with the eigenvalues with the highest norm. The number of eigenvectors that should be kept heavily depends from the data, but some rules of thumb are available [Zhao et al., 2003].
2. Faces are then represented as linear combinations of these eigenvectors, called *Eigenfaces*. The Eigenface subspace is then defined as the subspace spanned by these

Eigenfaces. This performs the dimensionality reduction that we sought.

3. Face recognition is then made by projecting a new image into the Eigenfaces subspace and classifying this new image position in this subspace with relation to the labeled training sample images. The closest neighbor is the most probable recognized face.

Fisherfaces The idea of PCA, used in Eigenfaces, is simple: to find linear combinations of components in the training sample that maximize the total variance in the data. However, the images of the training set are clustered by their labels: each image is associated to a physical user, so the set of labels is the set of users, and there can be several images per label. This label is also called **Class** of the image.

Eigenfaces does not take into consideration the classes (labels) of the training set, and it might happen that the found components are not relevant to discriminate between these classes of objects. This is the case when most of the variance in the data is not brought by the class itself, but for instance light conditions. In such a case, images projected into the Eigenfaces subspace do not form distinct clusters, and a successful classification becomes challenging.

On the other hand, **Fisherfaces** also performs a dimensionality reduction, but with respect to the classes of the training samples. The components it computes maximize the inter-class variance, while minimizing the variance within samples of the same class. As such, it learns a class-specific transformation matrix: the facial features it will find are the most useful to discriminate between the different classes (users), which is bound to increase the accuracy of the classification, and hence the precision of the user recognition.

This strategy, called *Linear Discriminant Analysis*, was presented by the first time by the statistician Sir R. A. Fisher to classify flowers [Fisher, 1936]. The first published use of this technique for face recognition was in [Belhumeur, 1997], and the details of the computations of Fisherfaces are available there.

Local Binary Patterns Histograms (LBPH) Both Eigenfaces and Fisherfaces follow the same idea: a face picture can be seen as a point in a high dimensional space. Similar faces are seen as neighbors in this high dimension space. However, such a high dimension makes it impossible to process in practice. This is why both perform a dimensionality reduction which tries to discriminate between faces, Fisherfaces making this reduction with respect to the class of the objects. However, both of them remain very sensible to variance sources that do not come from the classes, such as light conditions, orientation, pose, etc.

On the other hand, Local Binary Pattern (LBP) is a class of features that describes the image locally. LBP finds it roots in texture analysis. The basic idea of LBP is to describe the nature of each pixel (for instance if it is a corner or an edge) with relation to its neighbor pixels. To do that, consider a pixel of the image and threshold its 8 immediate neighbor pixels with its value: if a neighbor value is higher, denote this neighbor with 1, else 0. This gives a pattern of 8 binary values, called *LBP code*, which can be converted into a decimal value. This operator hence

transforms an image into a LBP image. By definition, the LBP operator is invariant to linear transforms on the image.

LBP images can then be used for face recognition: the technique presented in [Ahonen et al., 2004] divides an LBP image into n chunks. For each of them, a histogram is computed. The long histogram, obtained by concatenation of these n histograms, constitutes a descriptive feature of the image. Using histogram distances on these long histograms enable a robust and efficient recognition of the class of the image.

Most efficient face recognition algorithm These three algorithms (Eigenfaces, Fisherfaces, LBPH) have different approaches to tackle the same problem and it comes naturally to determine which performs better. Several articles discussed which is the most accurate face recognition algorithm, among others [Belhumeur, 1997, Zhao et al., 2003]. It seems that the performance of each algorithm depends vastly on the dataset used for both training and testing. Later on, we will see how we performed our own benchmark in subsubsection 5.2.2.ii.

5.1.1.ii Gender-from-face recognition

As seen in the introduction of the chapter, determining the gender of the user is a valuable indication, as it helps pruning the possible match for other user recognition algorithms, as the face recognition algorithms that were explained in the previous section.

In this part, we contemplate determining the gender of a (possibly never seen before) user thanks to a picture of her face. This problem can be seen as a special case of face recognition: there are two classes only (male and female).

In previous subsection 3.2.3, page 47, we saw how to robustly detect faces in the Red Green Blue (RGB) and depth stream of the robot. We now target classifying these faces according to their gender (male or female).

As underlined in [Moghaddam and Yang, 2002], a very similar approach to face recognition can be used. Face recognition consists of determining if a given face image corresponds to a given user label. These labels come from a so-called *training set* consisting of images and the associated labels, for instance 10 pictures of user A, 15 of user B, etc. As such, gender-from-face recognition can be seen as a specific case of face recognition with only two ”meta-users”, i.e. two classes: user *Female* and user *Male*. The training set would then be a collection of pictures of male faces, and another one of female faces.

In subsection 5.2.1, page 103, we will see how a gender faces dataset can be quickly constituted thanks to automatic retrieval techniques and in subsection 5.2.2, page 104 how a 2-class face recognition classifier can be trained using this dataset.

5.1.2 Height and other anatomy-based techniques

Soft biometrics were previously defined as the gathering of data about the user's morphology without explicitly querying them to the user. They can be divided into three families.

- **Physical traits** are bound with the user body, such as the height, the weight, the color of the skin, the shape of the body, facial hair, etc.
- **Adhered human characteristics** concern the more temporary appearance of the user, for instance the color of her clothes, accessories, etc.
- Finally, **Behavioral traits** are bound with the user motion and actions, such as her gait.

User height: *The height of the user* is a very useful tool, as it helps to discriminate quickly between the different users, and so prune the search tree for a faster matching. It can also be used for a gender estimation, as men tend to be taller than women.

On a side note, Human-Robot Interaction (HRI) studies have demonstrated that the role of the robot is perceived differently according to the difference of size between the user and the robot ([Rae et al., 2013]). Detecting the size of the user and adapting to it allows to shape her relationship with the robot.

However, determining the height of the user is challenging. First, we cannot expect the user to always stand straight. On the contrary, it is likely that during an interaction process, she will relax and then be slightly stooped, bend over to check something, etc. Second, she might lift her arms higher than her heads for a variety of reasons: greeting, scratching her head, etc. For both these reasons, the metric difference between the lowest and the highest three-dimensional (3D) point of the user is only a rough estimate of the user height.

And yet, to the best knowledge of the author, related work use this method. One of the earliest multi-modal user tracking systems [Darrell et al., 2000] used the user height and the color of the skin to obtain a robust tracking of the users around a multimedia kiosk (interactive public screen display). Height is obtained by computing the median value of the highest point of a user silhouette in 3-D. The same method is used in [Mittal and Davis, 2003]. While working in most conditions, this is not robust to a user lifting her arm for instance.

User gait: The gait motion of the users can also be used for estimating their gender: men and women tend to have a somewhat different gait that can be used for differentiating one from the other, as explained in [Igual et al., 2013]. Gait analysis has even been extended for people recognition in [Bouchrika and Nixon, 2006].

Other metrics bound with physical traits will now be further detailed: the proportions of the user's body and the shape of her torso.

5.1.2.i Anatomy proportions

Artists have used the human body proportions for centuries for accurate drawing and sculpting of the human body. The relative length of parts of the body are used to build an accurate representation of a person. For instance, for an adult male, if the height of the head is defined as a unitary length, the user height is roughly equal to eight heads, its navel is located at five heads from the feet, etc. This technique is widely used by artists. More details are found in [Loomis, 1971] and examples are visible in Figure 5.1.

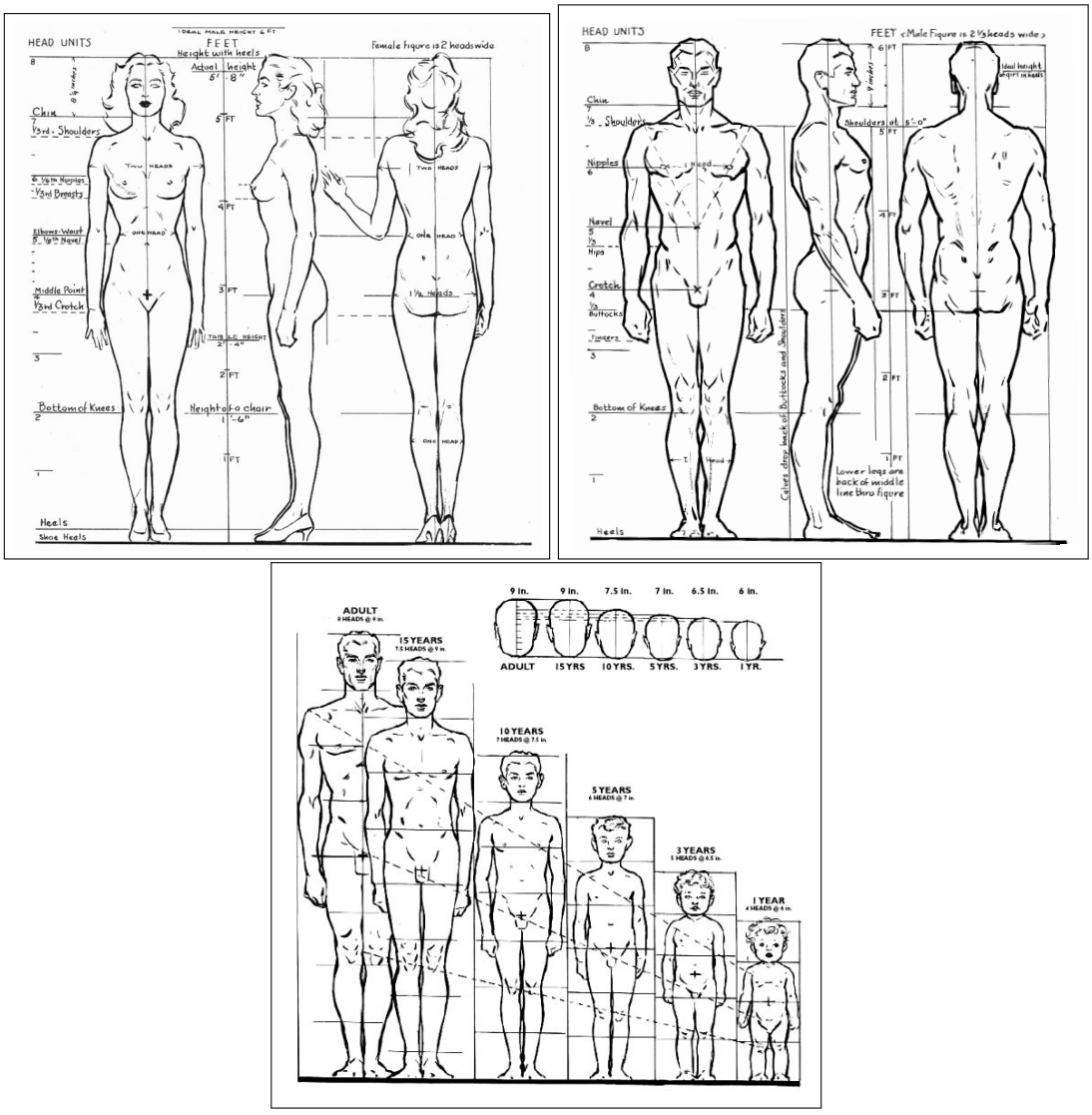


Figure 5.1: *Anatomy and proportions of the human body for artists. The size of each part of the body can be related to the others, independently of the total height of the person. The proportions of children are different from the adults. The three charts are from [Loomis, 1971].*

These proportions can then be used in computer vision algorithms. For instance, an accurate face detector could give an estimate of the size of the user head, which could help to approximate the user height, even though the full body is not visible. Another example is found in [Cai et al., 2010], which aims at determining the gender of a person according to the shape of his or her torso (more details in subsection 5.2.4). The authors use the body proportions to locate the area of the breast and prune the surface to be examined.

5.1.2.ii Breast detection

Considering the shape of the breast of a person is a natural way of estimating its gender [Laws and Cai, 2006]. Human proportions are used to find the area of the breast in the user scan then a feature shape is fitted to that area. The scale of that shape determines the gender of the user.

However, this method turned out to perform poorly in real time and with challenging body shapes. Furthermore, as further underlined in [Cai et al., 2010], the users are concerned about the privacy of this detection and might consider it intrusive.

We will see in subsection 5.2.4 how we tackled these limitations.

5.1.3 Histogram-based user recognition

Another class of soft biometrics consists of features that describe the temporary visual appearance of the user. Color is one of the most important visual cues.

Human vision system is highly trained for perceiving color (thanks to the cone cells located in the eye retina, and then the different parts of the cortex in charge of color perception). We heavily use color information for recognizing objects and people. As such, it makes sense to use color information to recognize users. Histograms are a compact and robust way to represent color distribution for an object or a user.

In subsubsection 5.1.3.i, some basic notions about histograms will be taught, the use of histograms for images will be presented. and existing techniques and metrics for histograms comparison will be reviewed.

In subsubsection 5.1.3.ii, we will present how histograms were used for user recognition in previous works by other authors.

5.1.3.i Histograms basics

Definition Histograms are powerful tools for representing data in a more compact way. For a given set of numerical data, a histogram is a set of values that represent the way this data is spread. It is made of a series of so called *bins*, which can be seen as a cell contain one numerical value. Each of these cells represents a range of possible values for the data, such as the range of a cell begins where the range of the previous cell ends. The number of bins is

constant. For instance, let us say we want to represent the height of the population of a country, in centimeters. We could represent it with a histogram of five bins, with range $[150, 160[$ for the first bin, $[160, 170[$ for the second bin, etc. As such, the whole range $[150, 200[$ is represented by this histogram.

As seen in this simple example, minimum and maximum values for the histogram must be defined. It can be done either by the data own constraints (finite number of values, for instance, percentage of accuracy of different methods, between 0 and 100), or by choosing reasonable bounds: human ages can be safely bound in $[0, 130]$. In our example, we chose to represent a given span that represents most of the population, but not all of it. We could have chosen to have bins between 0 and 300 centimeters, thus including all possible data but resulting in a less compact feature.

The single numerical value of a given bin represents the amount of the data set which has values in the range of this bin. Let us now consider the data of our population is the 10 following values: $177; 166; 185; 177; 166; 163; 196; 154; 142; 185$. The first bin, accounting for $[150, 160[$, contains the number 1, the second, for $[160, 170[$, contains 3, the third, for $[170, 180[$, contains 2, the fourth, for $[180, 190[$, contains 2, the fifth and last, for $[190, 200[$, contains 1. [2] The histogram can thus be represented as such:

Index	1	2	3	4	5
Value	1	3	2	2	1

How to choose the number of bins? There is a trade-off between precise representation, and data compactness; and is highly dependent on the size of the data. For instance, our previous example data has ten entries and the histogram range was $[150, 200[$. Using only three bins in a poor description of the data: knowing most of the data is in the $[166, 183[$ span is a hint, but it could be refined. On the other hand, having ten bins with most of them empty does not describe well the data either. Five bins then appears as a good tradeoff.

Normalization: A histogram represents the way data is dealt over a given span. As such, what matters is not really its absolute values, but the ratio between them. If a cell has a high value compared with the others, it means the span it represents is statistically more frequent. Multiplying all bin values by a constant factor does then not change the meaning it contains. If for instance, much of the data fits into one bin, multiplying all bins by 2 will not change this trend.

Norms A norm is a function that maps a set of numerical values to another numerical value. A norm is linear, which means the norm of a set where each value is multiplied by a constant factor is equal to that constant multiplied by the norm of the set. There are numerous kind of norms, the most common being the L_1, L_2 and L_∞ norms.

L_1 norm The L_1 norm is defined as the sum of the absolute values:

$$L_1 : \left\{ \begin{array}{ccc} \mathbb{R}^n & \mapsto & \mathbb{R} \\ X = x_1, \ldots, x_n & \rightarrow & \|X\|_{L_1} = \mid x_1 \mid + \cdots + \mid x_n \mid \end{array} \right.$$

[2] We can see that one of the entries is discarded: 142 is outside of the data range.

L_2 **norm** The L_2 norm is defined as the square root of the sum of the squared values:

$$L_2 : \begin{cases} \mathbb{R}^n & \mapsto & \mathbb{R} \\ X = x_1, \ldots, x_n & \rightarrow & \|X\|_{L_2} = \sqrt{x_1^2 + \cdots + x_n^2} \end{cases}$$

L_∞ **norm** The L_∞ norm is defined as the maximum of the absolute values:

$$L_\infty : \begin{cases} \mathbb{R}^n & \mapsto & \mathbb{R} \\ X = x_1, \ldots, x_n & \rightarrow & \|X\|_{L_\infty} = \max \mid x_1 \mid, \ldots, \mid x_n \mid \end{cases}$$

Histogram normalization consists of multiplying the histogram by a specific numerical value. This value depends on the data of the histogram and of the chosen norm. It is chosen so that, once multiplied, the norm of the histogram is equal to one.

We consider the previous example of the heights in a given population:

Index	1	2	3	4	5
Value	1	3	2	2	1

The L_1 norm of this example is $1 + 3 + 2 + 2 + 1 = 9$.
The L_2 norm is $\sqrt{1^2 + 3^2 + 2^2 + 2^2 + 1^2} = 4.36$.
The L_∞ norm is 3.
And so we get the following normalized histograms:

Index	1	2	3	4	5
Value of H_{L_1}	$\frac{1}{9}$	$\frac{3}{9}$	$\frac{2}{9}$	$\frac{2}{9}$	$\frac{1}{9}$
Value of H_{L_2}	0.23	0.69	0.45	0.45	0.23
Value of H_{L_∞}	$\frac{1}{3}$	1	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{1}{3}$

Histograms for images Histograms are a powerful tool of data crunching, used in a wide range of fields, among other statistics of course, but also economy, biology, physics, and many others. They are also vastly used in image processing to represent the way images values are dealt.

For instance, the values of a greyscale image can be used to generate a histogram. The bounding values of the histogram are well defined: greyscale images have their values between 0 and a maximum, usually 255. The histogram will show how the image brightness is dealt over the image. In Figure 5.2 (d), we can see a sample of a value histogram for a black and white image.

In fact, we can use any one-dimensional value for building the histogram. **A color space** is a way of representing visible colors into numerical values. Several methods remap the three-dimensional (3D) Red Green Blue (RGB) values to a one-dimensional (1D) color space and can be used for building histograms. For instance **HSV (Hue Saturation Value)** is a color space, where the Hue values span from 0 to 360 degrees, the Saturation and the Value values span

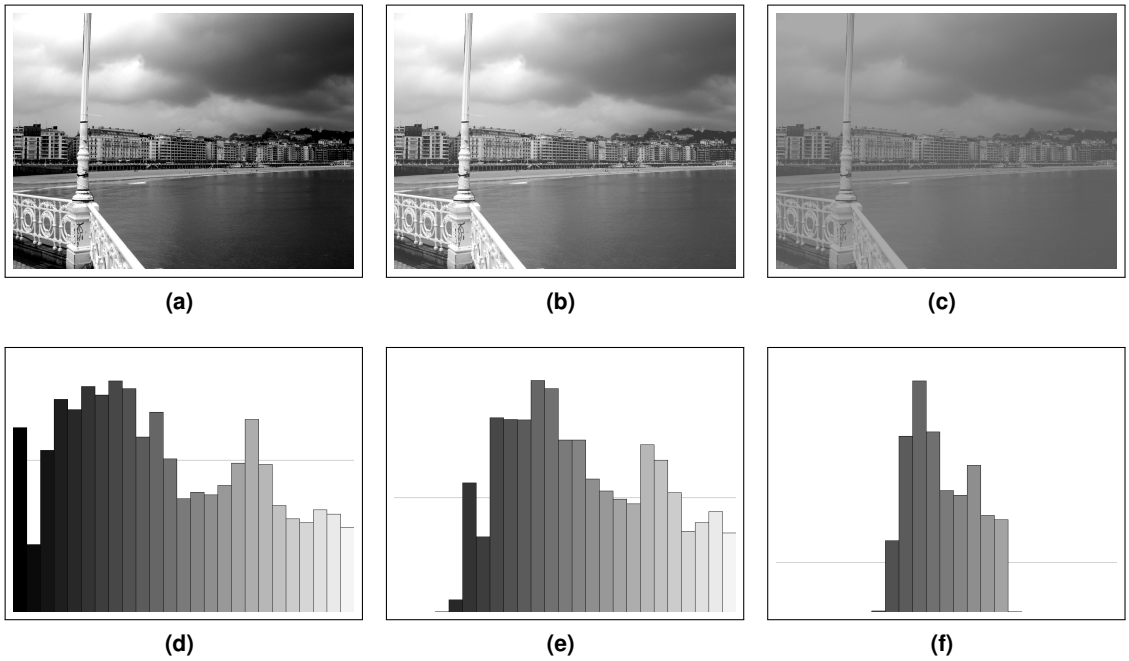


Figure 5.2: *A greyscale picture: (a) is the original image, (d) is its histogram, made between* 0 *and* 255 *and* 25 *bins. Values stretch in a balanced way between the minimal value* (0) *and the maximum value* (255)*.*
(b) is a version with more lightness (i.e., all the image values have a constant positive shift), Note how its histogram (e) is shifted to the right, which corresponds to brighter pixels.
(c) is a version with less contrast (i.e., the span of the image values is shrunk down). Its histogram (f) is shrunk down the same way: the values do not span from 0 to 255. .

from 0 to 1. More details about HSV come in [Bradski and Kaehler, 2008]. The Hue channel of an image in the HSV color space can for instance be used for computing a histogram. The Hue scale is visible in Figure 5.3.

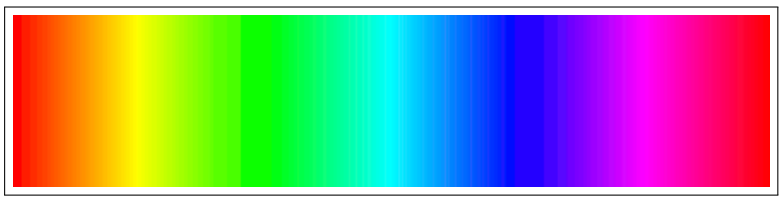


Figure 5.3: *The Hue scale. The left edge corresponds to a value of* $0°$*, the right edge to a value of* $360°$*.*

The key interest of the Hue channel is that it is a 1D feature that describes the color of objects. Furthermore, it has been proved to be fairly independent to light conditions: changes of brightness will result in different values in the saturation or value channels, leaving the hue component relatively intact. This differs vastly from the Red Green Blue (RGB) color space,

where the value of each component directly depends on the amount of light hitting the object, and therefore is correlated with the two others. As such, a Hue histogram results in a powerful and relatively light-independent descriptor of the color of an object. This is illustrated in Figure 5.4.

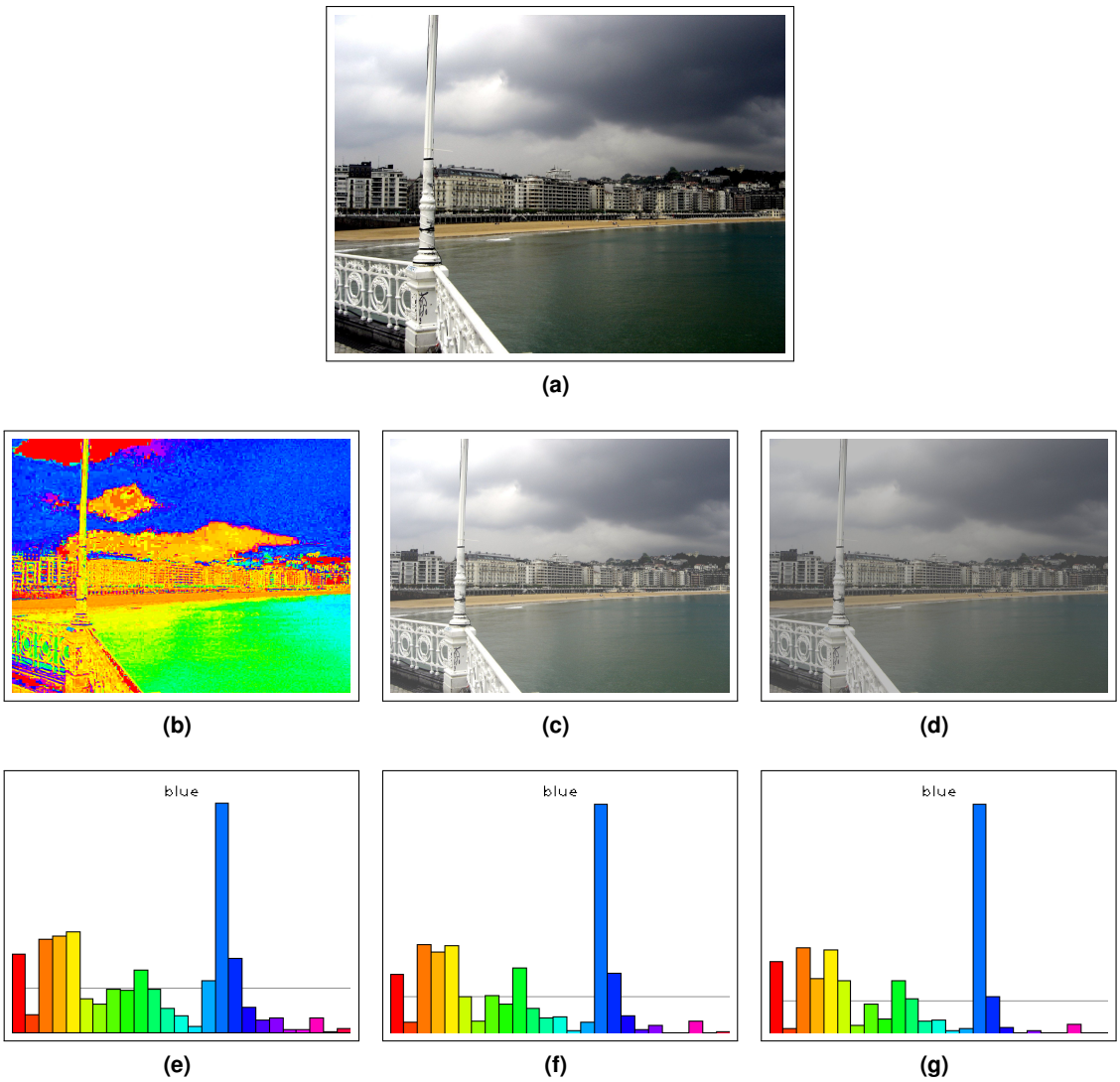


Figure 5.4: *Effects of lightness and contrast on Hue image histograms: (a) is the original image, (b) is the Hue component (i.e., with the Saturation and Value at their maximum), (e) is its histogram, made between* 0 *and* 360 *and* 25 *bins.*
(c) is a version with more lightness and (d) with less contrast. Note how the obtained histograms for the three versions (c), (f) and (g) are almost identical.

Histogram comparison We want to estimate to what extent how two histograms represent the same distribution of, for instance, color. Various metrics exist that enable the comparison of

histograms. All of them suppose the histograms to have the same number of bins. OpenCV[3], the Computer Vision library that we used, implement five different popular metrics for histogram comparison, namely the intersection method; the correlation method; the chi-square method; the Bhattacharyya distance; the Earth Mover's Distance method.

More details are available in [Schiele and Crowley, 1996] and [Bradski and Kaehler, 2008]. These metrics differ in their formula, but all of them can be remapped thanks to a linear transformation so that: (i) a perfect match between two histograms result in a distance of 0; (ii) a maximal mismatch gives a distance of 1. For instance for correlation, a perfect match is 1 and a maximal mismatch is -1; a value of 0 indicates no correlation. Using the transformation $d' = -0.5 \times d + 0.5$, we remap it so that $d'(H, H) = 0$ and d' for a maximal mismatch equals 1.

5.1.3.ii Histograms for user recognition

Face color matching has been used in many tracking systems.

Many examples of adhered human characteristics, especially the use of color histograms of clothes, exist.

For instance, in [Darrell et al., 2000], already seen in subsection 5.1.2, a histogram of the color of the clothes is used along with depth blobs obtained thanks to stereo-vision, face detection, and the estimated height of the user. The unique color histogram, kept for each user, was reported to be discriminative enough for big datasets if used with the other features, but not alone.

In [Krumm et al., 2000], the authors tackle the problem of person tracking in an intelligent environment, equipped with several stereo cameras. Although the detection of the persons is made thanks to the depth information of the stereo cameras, people tracking and identity coherency is ensured thanks to histograms of the color of the users. Each of the RGB axes is split in four wide domains, which gives a $4 \times 4 \times 4$ color cube and a 64 bin histogram. The system works well with up to three persons, with more, occlusions triggers poor clustering and tracks are not maintained.

The color of the clothes of the users are also used in [Saldivar-Piñon, 2012]: during a learning phase, the color model of the upper body of the user is learnt, then thanks to segmentation, its torso and arms are detected.

Another algorithm for extracting features of clothing is presented in [Tian and Yuan, 2010]: the matching between clothes items is evaluated to help color blind people. Two features are used for describing the clothes patterns: color and texture. A normalized color histogram in HSL space is first computed, then a discrete set of its dominant colors is used for the matching. The algorithm is reported to recognize correctly clothes pattern in a vast dataset. It is unclear to what extent the algorithm is robust to scale variations and how it can be used in real-time. However, unlike Hue values, RGB values are prone to change widely if light conditions change, which

[3] `http://opencv.org/`

was solved by continuously updating the features. Such a solution is hard to use in a multi-user environment.

These articles were about making a global histogram for the user. However, it makes sense of using the user shape for computing different parts according to its morphology. A single color model for the whole person is indeed be able to capture the vertical variation in color.

An interesting example is found in [Niinuma et al., 2010], where color histograms of the user's clothes and face enable her to identify herself in a computer, then they are used to continuously check that it is the same user who uses this computer. Two RGB histograms, one of the colors of the clothes and one of the face of the user, are registered during a login phase, then continuously matched with the current ones. The algorithm is reported to identify correctly each user in a 20 users dataset, and with varying light conditions.

In [Mittal and Davis, 2003], depth information is available thanks to stereo cameras, and several histograms are computed: the colored blob of the user is segmented by height slices of equal length. The similarity between histogram sets is made thanks to kernel estimators.

However, using slices of fixed length suffers from a drawback: the number of slices for small and tall people will be different. In other word, the index of a slice has no meaning, and we cannot guarantee that two slices of same index correspond to the same body part. For this reason, in subsection 5.2.5, page 122 we will present our own algorithm for user recognition based on structured Hue histograms, by computing first the ehgiht

Histograms summary

We have seen briefly what are histograms, how they can be applied to color images to generate hue histograms, and how to compare them thanks to different metrics. This knowledge will be used in one of our contributions: the use of structured histogram sets for user recognition, later explained in subsection 5.2.5, page 122.

5.2 Research contribution

As presented before, there is already a wide range of algorithms that exist and have been presented to perform user recognition based on computer vision techniques. We also mentioned how obtaining information about the user gender can be useful for its recognition.

The contributions of this PhD dissertation to this field take advantage of the existing state of the art.

In subsection 5.2.1, we explain how a face dataset with both genders can be generated in an innovative way, thanks to image search engines.

Next, subsection 5.2.2 presents how a gender-from-face recognizer trained with this data was implemented in one of our robots, MOPI.

In subsection 5.2.3 is presented a novel algorithm for the detection of the height of the user, even when she doesn't stand straight. This height information can be used for user recognition, or for some hints on the gender, as women tend to be smaller than men ([Nagamine and Suzuki, 1964]).

Then, subsection 5.2.4 shows how morphological features of the users can help the recognition of their genders, more specifically the shape of their breast.

Finally, subsection 5.2.5 presents a novel user recognition algorithm based on a description of the color of the user clothing, thanks to structured Hue histograms.

5.2.1 Building a dataset of training sample images for gender-from-face recognition

We have seen in subsection 5.1.3, page 96 how to perform gender-from-face recognition thanks to efficient face recognition algorithms, coupled with datasets of both male and female pictures.

A wide range of face images are available for academic use. For instance, one of the earliest and most used datasets is the ORL face dataset from AT&T Laboratories, Cambridge University, consisting of images of 40 subjects (36 men and 4 women, 10 images per subject) [Samaria and Harter, 1994]. Nowadays, an important effort has been made for building extensive face datasets, including both an important number of subjects and a variety of pictures per subjects, with varying poses, lighting positions, face expressions, etc. For instance, the Yale dataset [Lee et al., 2005] contains 5760 single light source images of 10 subjects each seen under 576 viewing conditions (9 poses x 64 illumination conditions).

However, in the scope of the implementation in MOPI presented in the next part, we focus only on gender-from-face recognition and not in specific face recognition. This is why we decide to constitute a gender-from-face images dataset using innovative techniques.

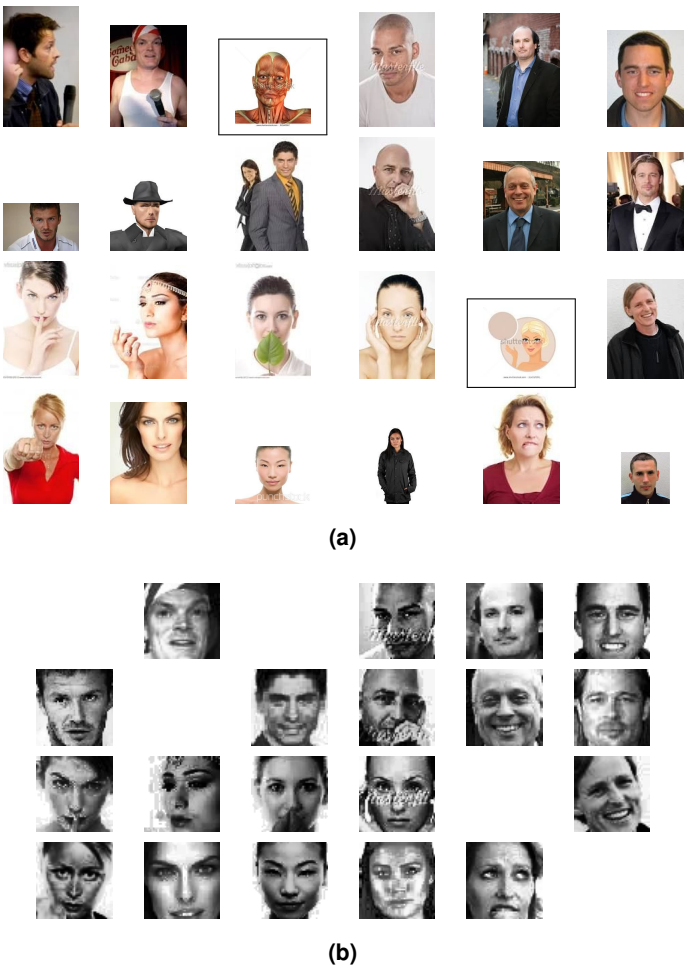
A first raw dataset of 550 images is obtained thanks to an image search engine. We collect the 225 first images returned by *Google Images Search (GIS)* [4] with the search query *man face* on the one hand, with the query *woman face* on the other hand.

A first manual review of the so-obtained 550 images allow us to remove the most evident outliers from these raw results, such as pictures not containing faces. The number of images remaining after this quick manual filter is 520.

Then, a second, more accurate filter is made by applying the Viola-Jones face detection algorithm described in section 3.1.1, page 33. The pictures which do not pass this filter might still contain a face, but at least all pictures which pass it do contain a usable face. Furthermore, the results of the filter supplies the bounding box of the face in the image. As such, filtered pictures can be used for an exact training of the gender classifier described in subsubsection 5.1.1.ii, page 93. The number of images remaining after this final filter is 467.

[4] `http://images.google.com/`

A few samples of the dataset are visible in Figure 5.5. A pie chart about the kept images is in Figure 5.6.



(a)

(b)

Figure 5.5: *Gender images dataset constitution thanks to GIS.*
(a): Raw image results from GIS, with queries man face *(two first rows) and* woman face *(two last rows). The images with a black frame were manually filtered, as they do not contain faces or inappropriate ones.*
(b): Remaining images in the dataset after applying the Viola Jones detector ([Viola and Jones, 2001]) and pre-processing the images. The missing pictures correspond to images that were either discriminated by the manual filtering, or the Viola Jones face detector.

5.2.2 Implementation and benchmarking of gender-from-face recognition for the social robot MOPI

We have seen in subsubsection 5.1.1.ii, page 93 how to build an algorithm to estimate the gender of a person according to the visual appearance of her face. It is indeed a sub-problem

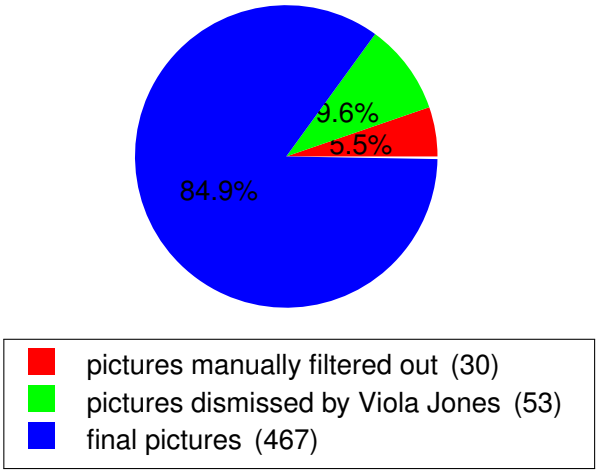


Figure 5.6: *Number of pictures in the GIS gender dataset after each step.*

of face recognition with only two classes: *female* and *male*. Three popular algorithms exist for performing face recognition: *Eigenfaces*, *Fisherfaces* and *LBPH*, seen in subsubsection 5.1.1.i, page 91.

In this part, we will explain how we implemented a real-time gender-from-face recognition system into the social robot *MOPI*, already presented in subsubsection 1.3.1.ii, page 6.

5.2.2.i Implementation of gender-from-face recognition into MOPI

Both face detection (seen in subsection 3.2.3, page 47) and robust gender-from-face recognition (seen in subsubsection 5.1.1.ii, page 93) have been implemented into the social robot MOPI.

They are structured as two different Automatic-Deliberative (AD) skills running at the same time as background skills into the software architecture of MOPI. The face detection skill subscribes to the image and depth stream coming from the Kinect, and publishes the results of the face detection, containing the rectangular cutouts of the image corresponding to the faces.

The gender-from-face recognition skill subscribes to this information and performs gender recognition on the cutouts. The resulting estimated number of male and female users around the robot is then published, and any other skill of the robot can then subscribe to it. A sample is visible in Figure 5.7.

The required times for data processing are showed in Figure 5.8. A typical VGA image, of size $640 \times 480 = 3k$ pixels, requires about 70 milliseconds to be procesed. The most costful step is the face detection, the gender classifier running in less than one millisecond.

The platform can also be used to benchmark the different face recognition algorithm. Further explanations follow.

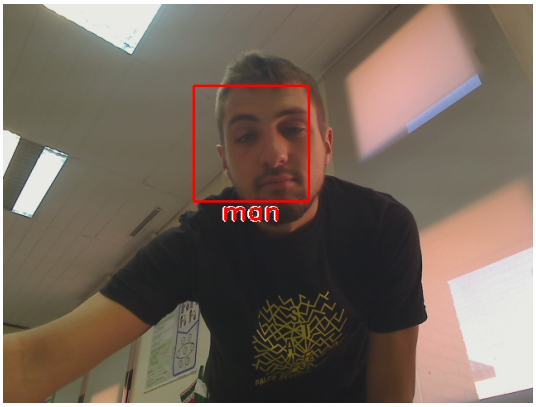


Figure 5.7: *A sample image of the gender-from-face recognition skill.*

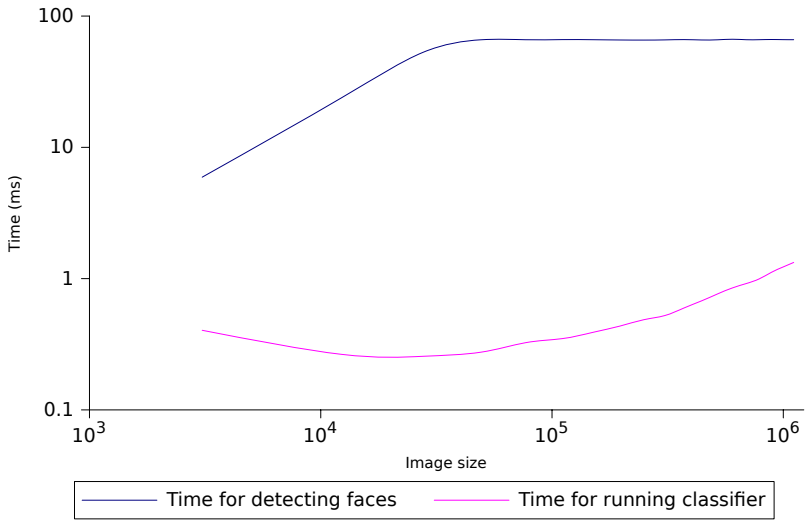


Figure 5.8: *Times needed for finding faces and estimating their gender on a sample image according to the size of the image (number of pixels).*

5.2.2.ii Comparative between face recognition methods

Classifier trained on Google Images Search (GIS), benchmarked on YaleB We presented in subsubsection 5.1.1.i, page 91 three different methods for face recognition: Eigenfaces, Fisherfaces and LBPH.

To evaluate to what extent which method performs better, we trained separately three classifiers with the same training set of images from GIS, as presented in subsection 5.2.1. Then, each of these classifiers was used to predict the genders of all faces of an annotated dataset and we compared the results of each classifier with the ground truth.

We made use of the YaleB dataset, an extension of the Yale dataset presented in subsection 5.2.1 ([Lee et al., 2005]). This dataset contains 16.380 face pictures, 12.870 being of male subjects and 3.510 of female. A sample of this dataset is visible in Figure 5.9. It was designed for benchmarking user recognition, this is why we manually labeled the gender of these pictures for our benchmark. The results are visible in Table 5.1.

Figure 5.9: *Some samples of the YaleB dataset.*

	Eigenfaces	Fisherfaces	LBPH
Training set size	408		
Test set size	16.380		
Number of detected faces	12.3367		
Precision	46.8 %	66.0 %	56.8 %

Table 5.1: *Benchmark results for the gender-from-face recognition trained on GIS and tested on YaleB*

Overall, the performance is not outstanding. This might be explained by the fact that most images from our dataset offer well-lit, well contrasted images, as seen in Figure 5.5 (a), while most of the YaleB dataset pictures offer challenging light conditions, as seen in Figure 5.9. We should obtain better results if we train with similar images to the test images: this point will be adressed below.

We can also point out the very different performance of the three classifiers. Fisherfaces perform much better tan Eigenfaces. This makes sense, as the dimensionality reduction made by Fisherfaces is made with respect to the classes of the objects (here the gender), while the one of Eigenfaces only tries to maximize the total variance of the data. As such, the found components may or may not be appropriate to discriminate between the genders. However, the poor performance of LBPH is harder to justify. It might be related to the previously mentioned important differences of contrast between the two sets of images. In any case, Fisherface outperforms both other algorithms.

Classifier trained on YaleB subset, benchmarked on YaleB subset The subset is obtained from the original 16.361 images, where only roughly 10% are randomly kept. It then has a size of 1669 images. It is split in two rough halves: a training set of 814 images, and a test set of 855. As such, an image cannot belong to both training and test sets.

The results of the benchmark are visible in Table 5.2. We observe a very clear improvement compared with the classifier trained on GIS, as Fisherfaces performs more than 20% better when trained on YaleB. This is explained by the fact that the training set and the test set are very similar (although the images are distinct), which helps greatly the classifier.

As in the previous benchmark, Fisherfaces outperforms the other algorithms. For this reason, it is the one used in the final application.

	Eigenfaces	Fisherfaces	LBPH
Training set size	814		
Test set size	855		
Number of detected faces	634		
Precision	87.5 %	88.4 %	88.9 %

Table 5.2: *Benchmark results for the gender-from-face recognition trained on half of YaleB and tested on the other half of YaleB*

5.2.3 Height detection for user recognition and gender estimation

As seen in subsection 5.1.2, page 94, the height of the users is a good metrics for two goals: first, for a known user, it helps recognizing him from others; second, the height of unknown users helps determining their gender, as men tend to be taller than women. However, related articles determine the height of the users by the highest three-dimensional (3D) point of the user bounding box. This supposes that the user stand straight, while she can relax and then be slightly stooped, or lift an arm for greeting.

We here present a novel method for estimating the height of the user, that overcomes these difficulties. It requires the depth image and user mask image, and supplies a estimate in meters of the height of the user.

The next subsubsection 5.2.3.i, will first define how to perform a so-called **Morphological thinning** on a binary image, that generates the **Skeleton** of the image. In subsubsection 5.2.3.ii, we will see how to detect the probable position of the head in the user mask. Both the skeleton and the head position will be used in subsubsection 5.2.3.iii for determining the height of a user given its depth image and user mask. Finally, the accuracy and performance of the algorithm will be discussed in subsubsection 5.2.3.iv.

5.2.3.i Preliminary: fast morphological thinning for binary images

This new height detection algorithm is based on finding a line that goes from the head of the user to her feet, going through the middle of her body shape, i.e. more or less following her dorsal spine and then a leg. The length of this line gives an estimation of the user height.

A field of computer vision comes of special interest for this application: **Morphological thinning**. It consists of transforming a shape into a simplified version of it, but topologically equivalent, called **Topological skeleton**. This topological skeleton is equidistant to the border of the original shape. [Jang and Chin, 1990] sums up the properties that can be expected of a good algorithm for the computation of topological skeletons: 1. the algorithm must converge; 2. the generated skeleton must be one pixel wide; 3. it should retain the connectivity of the original shape; 4. it should preserve the ”legs” of the shape, which are its convex parts.

Baseline algorithms used With respect to these criteria, we chose two thinning algorithms complying with these criteria and with an available implementation in C++: [Guo and Hall, 1989] and [Zhang and Suen, 1984]. Both are iterative algorithms that initialize the skeleton as the original shape that we want to thin. Then, they will iteratively remove points on the contour of the skeleton. At each iteration and for each pixel of the skeleton, a so-called *deletion test* will determine if the pixel needs to be removed according to its neighbors. The two papers differ in the definition of their deletion test. The algorithms end when no further pixel is removed from the skeleton during an iteration. Both algorithms running on different samples and at different iterations and at convergence are shown in Figure 5.10.

Implementation in OpenCV [Guo and Hall, 1989] and [Zhang and Suen, 1984], both reference algorithms for image thinning mentioned above, are iterative algorithms. At each step, the eliminated pixels are only on the contours of the non-zero pixels blobs of the current mask.

As such, we can accelerate each iteration by only considering the pixels at the border of the mask and not all of them. We then have created a data structure called **contour images**, which encodes, for each pixel, if it is outside of the blobs (pixel value of 0); if it belongs to the contour (pixel value greater or equal to 1 and at the edge); or if it belongs to the inner part of the blobs (pixel value greater or equal to 1 and surrounded by others non null pixels).

The basic actions on such a structure are twofold:

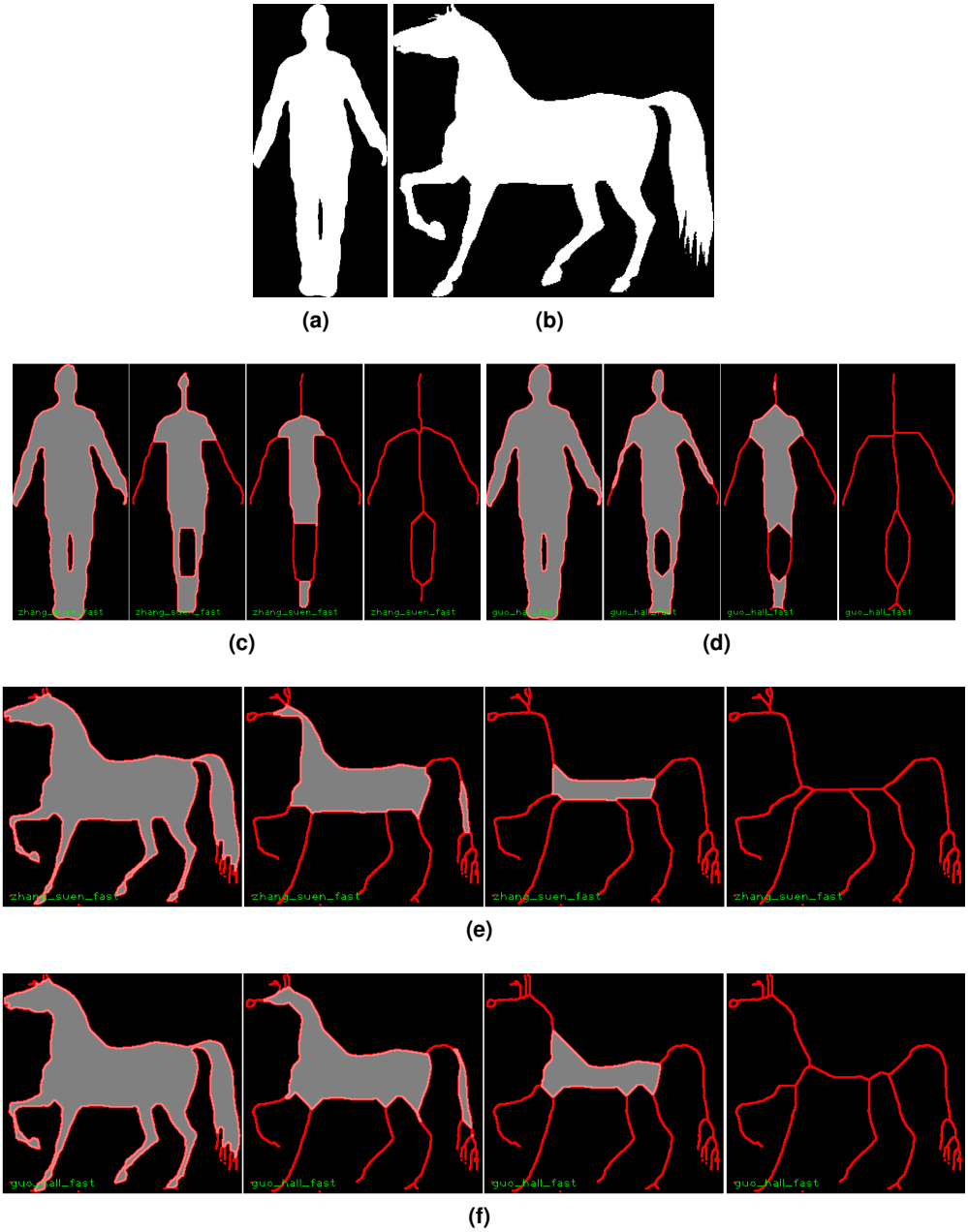


Figure 5.10: *Results of the thinning on different samples.*
(a) and (b) are the query images, (c) and (e) correspond to thinning thanks to [Zhang and Suen, 1984], (d) and (f) thanks to [Guo and Hall, 1989].

1. determine if a pixel belongs to the contour;
2. switch a pixel status from border to outside.

These actions trigger the update of the underlying data structure, as shown in Figure 5.11. At each step of [Guo and Hall, 1989] and [Zhang and Suen, 1984], the elimination test, which is run in the original algorithm on each pixel, can be run on the contour pixels only, thanks to the use of these contour images. Retrieving the value of a pixel in a binary mask being significantly faster than running the elimination test on it, resulting in a significant speedup for the algorithm convergence. Numerical results will be given later on.

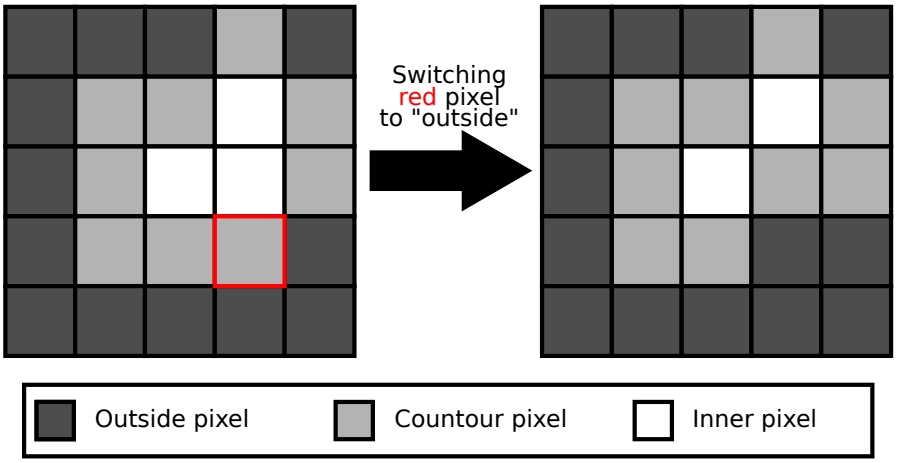


Figure 5.11: *The two basic actions for a contour image*

5.2.3.ii Head detection for user binary images

The height detection for a given user mask needs a reasonably accurate detection of the position of the head of the user in the user mask.

This could be obtained thanks to a face detection, as seen in subsection 3.2.3, page 47. However, in this case, it turns out to be not the best solution for several reasons: it is computationaly expensive; it only works with frontal detections; and it requires the Red Green Blue (RGB) image.

This is why we created another detector, using only the user mask. In most cases of HRI, the user stands straight in front of the robot. As such, her head is located in the blob at the upper end of the mask. This point corresponds roughly to the head. However, the upper pixel, as used in [Darrell et al., 2000] or [Mittal and Davis, 2003], is not appropriate, as it could belong to the user's arms if she is lifting them. Our method to overcome this challenge is hence to get rid of the blob corresponding to the arms using a morphological erosion filter on the mask [Bradski and Kaehler, 2008].

For the eroding, we use a kernel size proportional to the width of the whole mask. As such, a close user generate a wide mask, and consequently an important eroding; while a remote user is hardly eroded. A skeleton is also computed in that same mask, as seen in subsubsection 5.2.3.i. All ends of this skeleton, i.e. leaf points at the end of a segment, such as hands, feet, etc, are computed. An ellipse is fitted to the eroded user mask ([Fitzgibbon and Fisher, 1995]), and the upper end of the long axis of this ellipse is considered to be a first estimate. Then, a more precise estimate is obtained thanks to the end of the skeleton closest to that first estimate. Some samples are visible in Figure 5.12.

5.2.3.iii Use of the skeleton for height computation

We saw first in subsubsection 5.2.3.i how to get the skeleton of the user binary mask, and we have just seen in subsubsection 5.2.3.ii how to determine the approximate position of the head. As seen for instance in Figure 5.10 (c), the skeleton we obtain goes indeed from the head position to the feet position. If the user is not occluded, the height in pixels of the user can now be seen as the shortest path *along the skeleton* between the head position and the feet position [5].

First we will focus on the general problem of how to find the shortest path in a binary mask, then we will see how to use this method for the height computation of the user in pixels, then in meters.

Shortest path computation in a binary mask A **Binary mask** is an image of finite size where the value of each pixel is either 0 or positive. Pixels with value 0 as seen as *forbidden*, while positive pixels are seen as *allowed*. Given a mask M, we note M^+ the allowed pixels in M, and M^0 the forbidden pixels. For instance, our childhood mazes can be described as a binary mask where the hallways are positive, i.e., white, and the walls zeros, i.e., black, as shown in Figure 5.13.

Given a mask M of pixels, where M^+ describes the allowed pixels, and two points $a, b \in M^+$, we search the shortest path starting in a and ending in b that only goes through points of M^+. This problem could be efficiently solved using graph shortest path solvers such as Dijkstra's algorithm [Dijkstra, 1959], which complexity is roughly logarithmic in the size of the graph $\Theta(\log |M^+|)$.

However, we implemented a much simpler algorithm inspired by floodfill propagation and performing a BFS (Breadth-first search). Starting from a, we recursively add all the neighbor pixels in a queue till reaching b. Details of the algorithm are in algorithm 2.
Time complexity is roughly linear $O(|M^+|)$ and space complexity is constant $\Theta(|M|)$. As the skeleton obtained previously is only one pixel thick, $|M^+|$ is of the same order of magnitude

[5] In case of a partial occlusion, if this occlusion does not hide this shortest path, the shortest path might still be the height of the user.

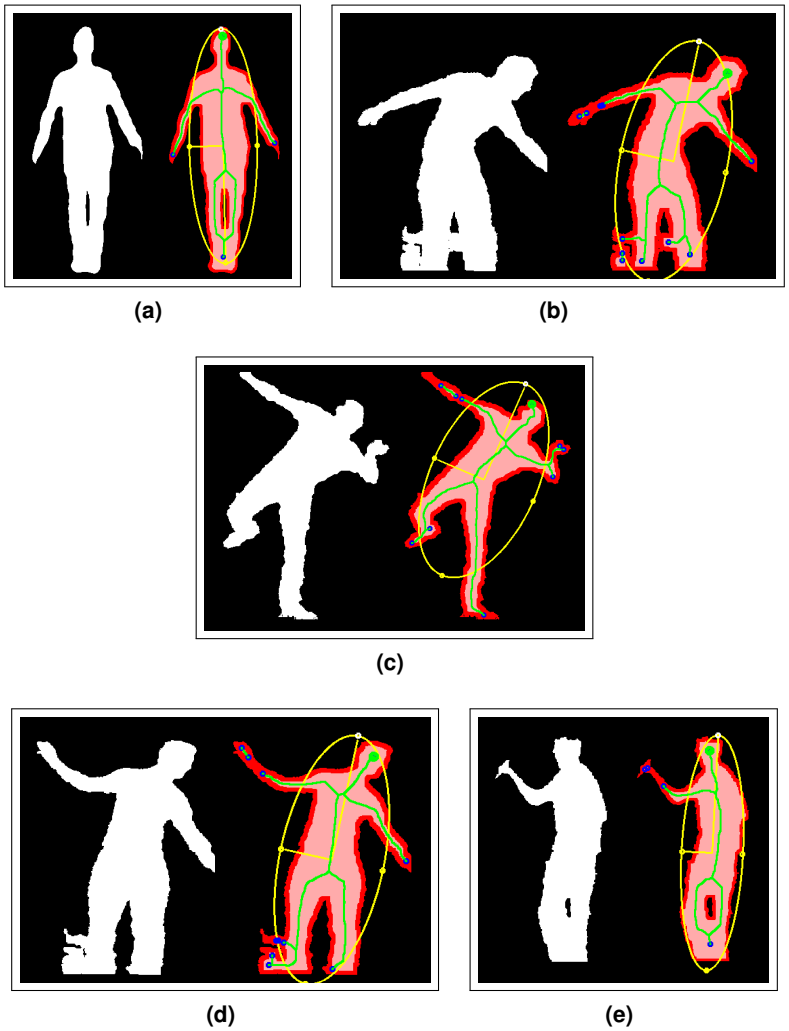


Figure 5.12: *Samples of head detection in a user mask. The white mask on the left of each image is the user mask. On the right, the dark red is the original mask, the light red the eroded one. The yellow ellipse is the fitted ellipse and the yellow lines its axes. The white point is the first estimate, i.e. the highest ellipse long axis. The blue circles are the ends of the morphological skeleton in the eroded mask. The green circle corresponds to the final estimated head position. .*

than the width or height of M. As such, a linear complexity will not generate an expensive computation overhead compared with Dijkstra's logarithmic one.

Application and conversion between distance in pixels and in meters We have seen before how to obtain the morphological skeleton of the user mask M, and an approximate location h for the user head in the user mask. Re-using previous definitions, a corresponds to

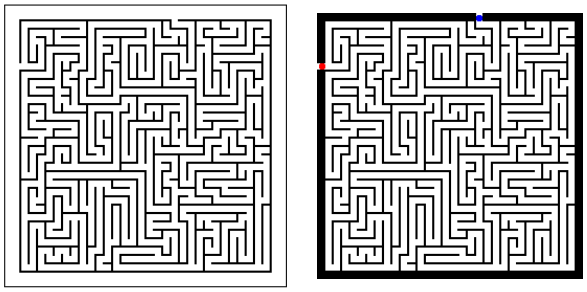


Figure 5.13: *Mazes as binary masks. (a) represents the maze as we are used to see it, (b) its representation as a binary mask, where the trivial paths have been forbidden and both beginning and end points indicated. .*

Data: mask M, seed a, goal b
Result: path P from a to b in M^+
initialize cost matrix C such as $C(M^+) = 0, C(M^0) = -1$;
initialize queue Q with a;
$C(a) = 1$;
while Q *not empty* **do**
  $q \leftarrow$ first element of Q;
  remove first element of Q;
  if $q = b$ **then** `// path found`
    `// get path back with decreasing cost values from` b `to` a
    $p \leftarrow q$;
    while $p \neq a$ **do**
      add p to path P;
      $p \leftarrow$ unique neighbor of p with a cost $< C(p)$;
    return P; `// done`
  else
    `// add un-seen neighbors of` q `with an increased cost`
    for q' *C4 neighbor of* q **do**
      if $C(q') == 0$ **then** `// unseen and in` M^+
        add q' to Q;
        $C(q') \leftarrow C(q) + 1$;
return; `// No path exists`

Algorithm 2: The BFS algorithm used for computing the shortest path in a binary mask

the closest point of h in the skeleton, and b the point of the skeleton in the lowest row, which corresponds to the position of the feet. An example is visible in Figure 5.14, where a and b are

marked with green circles.

Then the shortest path from a to b in M corresponds to a path that would follow the neck; then the spine; then the leg bone of the user; to her feet. This path can be computed thanks to the algorithm we just explained. Its length then gives an approximation of the size of the user, in pixels.

Thanks to the depth image, we can then convert the distance in pixels into a distance in meters: the average distance of the user to the optical center is equal to the average depth of the pixels that belong to the user mask. The camera pin-hole model then is used to convert the height in pixels at the distance into a height in meters.

The description of the height detection algorithm is now complete, we will now evaluate its performance through a thorough benchmarking.

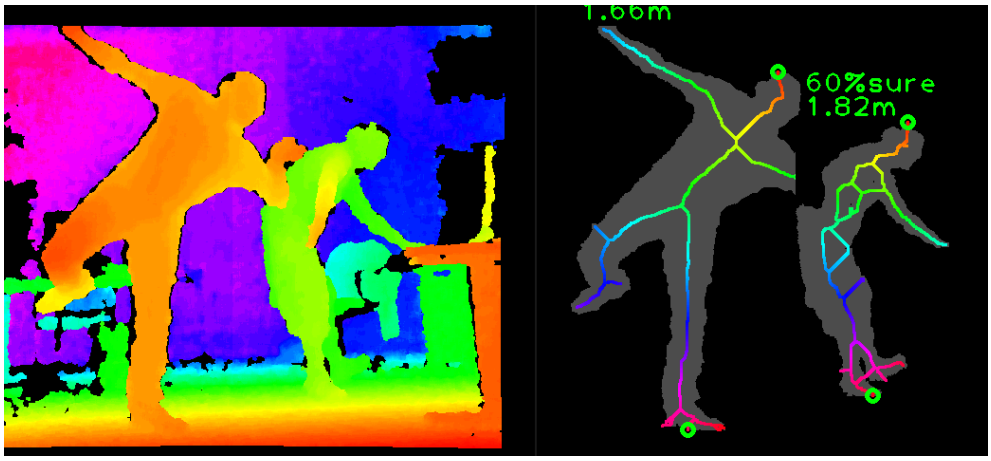


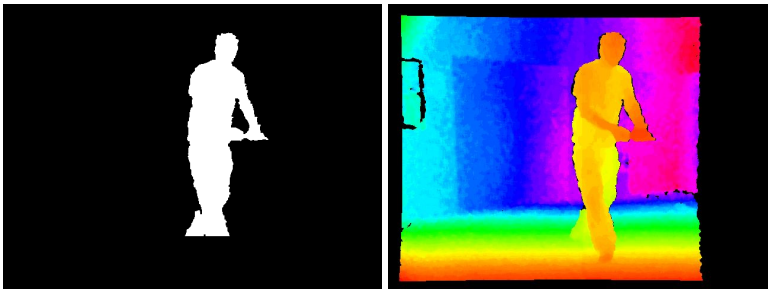
Figure 5.14: *Sample of height computation for several users at once. The heights are both indicated in pixels and in meters.*

5.2.3.iv Benchmarking of the height detector

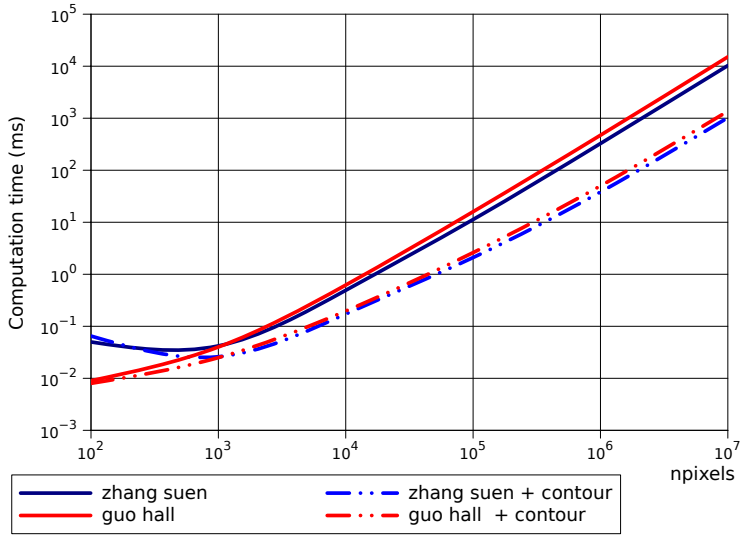
Benchmarking of the image contour speedup for thinning algorithms The four following thinning algorithms were benchmarked: 1. [Guo and Hall, 1989], 2. [Zhang and Suen, 1984], 3. [Guo and Hall, 1989] using contour images, 4. [Zhang and Suen, 1984] using contour images.

Note that, as explained before, the use of contour images does not alter the resulting thinned image and only results in a speedup in its computation.

We used one given test image. The test image was chosen to be representative of a user skeleton computation scenario. It is visible in Figure 5.15 (a). This high resolution user mask was scaled down to a range of given widths, and the needed time to run the algorithm 1000 times consecutively was measured for each method. The benchmark results are visible in Figure 5.15. The use of the contour images trigger a speedup of almost 10 times for both reference algorithms.



(a) (b)



(c)

Figure 5.15: *Benchmark results for different thinning algorithms.*
(a) The query image, that is resampled to different widths and heights. It corresponds to a typical user mask.
(b) The RGB image associated with the query image. This image was not not used in this benchmark.
(c) The use of contour images results in a computation speedup of almost one order of magnitude.

Skeleton-based height computation against the classical approach To measure the precision of our skeleton-based height estimation algorithm, we benchmarked it against another method, labeled ”the classical approach”. It corresponds to the algorithm used, among others, in [Darrell et al., 2000] and [Mittal and Davis, 2003]. It consists in reprojecting in 3D the mask of the user thanks to the depth information, then computing the 3D bounding box of this reprojected point cloud. Then the height of the bounding box is the height estimation. In other words, the height of the user is equal to the height difference between the user's lowest point (usually the feet) and the highest point (the head).

Although it has numerous user samples, we cannot use the DGait database ([Igual et al., 2013]) that we previously used. Indeed, the database authors do not supply the exact metric height of each user. We also considered the Kinect Tracking Precision (KTP) dataset ([Munaro et al., 2012, Munaro and Menegatti, 2014]), but it suffers from the same limitation. For this reason, we gathered a number of our own user images.

We made a quick implementation of the classical approach, previously presented. We gathered roughly 25 images, containing users of various sizes, on which we ran the benchmark. The results are gathered in Table 5.3. Some samples are visible in Figure 5.16.

In average, the average error of the classical method is over ten centimeters, while our method error is about five centimeters. In most cases, the user's head is indeed her highest point and both approaches are similar, thus obtain similar results. However, in cases where for instance the user is bending or lifting arms, our approach correctly locates the position of the user's head and obtains a better estimate.

	Classical approach	Our approach
Images in test set	25	
Unique users in test set	5	
Range of unique users heights (cm)	$[152, 187]$	
Average error (cm)	5.3	10.5

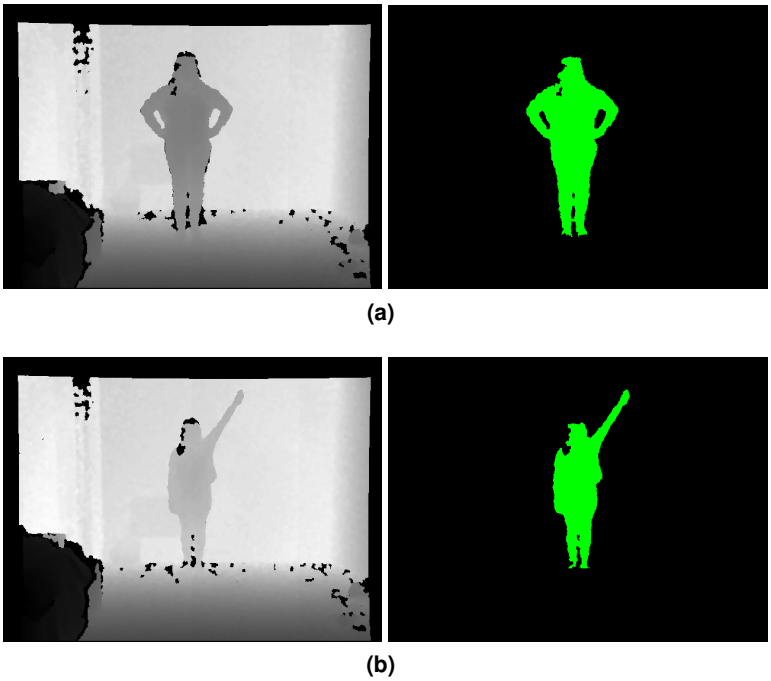
Table 5.3: *Benchmark results for height estimation algorithms.*

5.2.3.v Height detection for user recognition

Now that we have seen how to determine the height of the user from the input data, we can apply it to user recognition. At each `PeoplePoseList` (`PPL`) supplied by a user detector (for instance, any of the detectors presented in chapter 3, page 31), the height detector will compute the height of each user present in the message. Then, for matching known tracked users to detections, the height differences between both sets generates a similarity matrix. This will be later presented in section 7.2.3, page 159.

5.2.3.vi Height detection for gender recognition

We previously presented how the height of the users around the robot could be inferred using their visual appearance in a reliable way. The height of the users can, in some cases, give a hint about their gender. Indeed, the average female population is slightly shorter than the male one in Western countries ([Garcia and Quintana-Domeque, 2007]). Hence, the taller users tend to be male, while the smaller ones female. Although no definite conclusion can be drawn from this observation, it can help the system to discriminate between two possible recognition matches of opposite gender.



(a)

(b)

Figure 5.16: *Benchmark results for our homebrew height algorithm vs. the classical approach.*
(a) Ground truth: 152 cm, our approach: 152 cm (error: 0 cm), naive approach: 151 cm (error:1 cm). In this case, the user stands straight: the head is the highest 3D point of the user, and so both methods measure accurately the user height.
(b) Ground truth: 152 cm, our approach: 150 cm (error: 2 cm), naive approach: 185 cm (error:33 cm). In this case, the user lifts an arm above her head: the classical approach measures the distance between the hand and the feet, and so overestimates the height. Our approach finds the head accurately and estimates the height correctly.

This possibility has unfortunately not been explored further. We instead focused our efforts on another method for estimating the gender of the detected users thanks to their visual appearance: breast detection.

5.2.4 Breast detection for gender estimation

We just saw how the height of the users can be a metric for recognizing them and how to compute it.

But other morphological features can be used. This part will focus on the shape of the breast of the users. We have seen in subsubsection 5.1.2.ii, page 96 different techniques for gender detection thanks to breast detection. The work presented in this part aims at determining the gender of the user using similar techniques, but overcoming the limitations of the previous work.

5.2.4.i Description of the system

A structured light sensing device (Kinect) supplies a continuous stream of color and depth images to the robot Maggie. A full description of the robot is available in the subsubsection 1.3.1.i, page 5.

User detection and mask computation The input needed for our system is a three-dimensional (3D) (colorless) point cloud of the user. This can be obtained through a range of input devices, such as a structured light depth sensor such as Microsoft Kinect, or a stereo vision device. In any case, the point cloud given by the device needs to be cut down to keep only the points of the user. Our system sports two sources for this user mask. A device-specific solution is to use the one directly given by the device Software Development Kit (SDK) (Kinect NiTE SDK presented in subsection 3.1.3, page 36). The other way is to obtain a seed pixel belonging to the user thanks to face detection (subsection 3.2.3, page 47), detect the edges in the depth mask thanks to a Canny filter (subsection 3.2.1, page 39); the final user mask is obtained by performing a propagative floodfill in the edge image from that seed (subsection 3.2.1, page 39). In both cases, the point cloud is obtained by reprojecting to world coordinates the pixels of the depth map that also belong to the user mask.

Breast feature computation Laws' algorithm ([Laws and Cai, 2006]), which is the most recent and similar algorithm, is based on extracting several horizontal slices of the point cloud of the user for a span of height values that corresponds to the breast location in a body. The contour of each slice is matched against the pattern of a breast shape using non-linear regression [Shanno, 1970]. Metrics on the pattern that fits best one of the slices are then used to evaluate the gender of the user. However, female users with a moderate breast and over-weight males generate similar horizontal slices.

Our algorithm is instead based on computing the vertical contour of the torso of the user. The pipeline is illustrated in Figure 5.17.

First, the point cloud of the user is aligned with the three axes, to correct small tilts due to the user bending over. Then the orientation of the user is determined by fitting an ellipse to the shadow generated by the projection of the point cloud onto the ground plane ([Fitzgibbon and Fisher, 1995]). The shortest axis of the ellipse corresponds to the front direction of the user. The projection plane is chosen to be vertical, going through the user's Center of Mass and parallel to this short axis. The points of the cloud with a height corresponding to the breast, according to morphological rules seen in subsubsection 5.1.2.i, page 95, and that are close to that plane are projected onto it. The outline of the user thus generated is approximated by a fixed length polygonal line, which consists of the final feature.

Gender determination A training set with pictures of users (depth images) and manually labeled genders must be gathered to generate training features and labels. This data, once

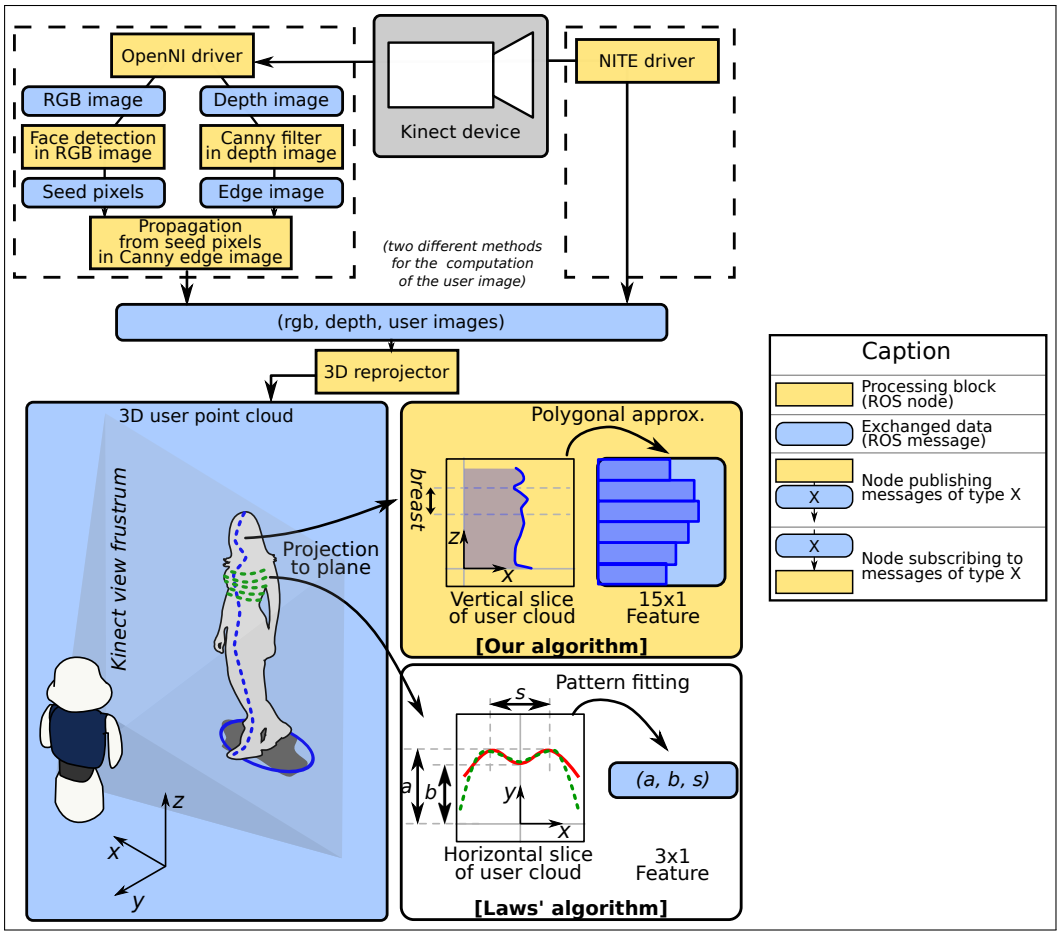


Figure 5.17: *Algorithms pipelines for both [Laws and Cai, 2006] and our algorithm. The robot Maggie is the blue shape on the left. The black shadow is the projection mentioned in subsubsection 5.2.4.i, and the ellipse the best fit to that shadow. The projection planes for both methods are indicated in dashed lines.*

gathered, can then be fed as training set to a two-class Support Vector Machine (SVM) classifier ([Cortes and Vapnik, 1995]) to learn the non-linear separation between male and female shapes. Once trained, the prediction output of the SVM for a test shape determines the gender of the user. We will now present the data we used and our experimental results.

5.2.4.ii Implementation and benchmarking of the algorithm

The algorithm was integrated into the robot Maggie. A full description of the robot is available in the subsubsection 1.3.1.i, page 5. It is written in C++ and is based on the Robot Operating System (ROS) architecture (see subsubsection 1.4.1.i, page 9) and the Automatic-Deliberative (AD) paradigm (see subsubsection 1.4.1.ii, page 12). It supplies in real-time an estimation of the gender of the surrounding users.

A user dataset was used to benchmark our algorithm against Laws' ([Laws and Cai, 2006]). The DGait dataset [Igual et al., 2013] contains video sequences of 55 users, both female and male, walking on a stage with varying light conditions. Both our and Laws' detectors were trained on 1100 images (20 per video), and tested on 550 (10 per video). The results are gathered in Table 5.4.

In his original article, Laws algorithm was tested on a dataset of medical scans. with full-figured, standing straight, subjects, and logically performs poorly on real-life user samples. Our algorithm correctly estimates the gender of the user in almost nine times out of ten.

	Laws	Our
Accuracy rate	63%	89%
Average computation time (ms)	83	11

Table 5.4: *Benchmark results for gender recognition based on breast detection*

5.2.4.iii Privacy concerns

The gender recognition algorithm runs in the background within the robot and supplies estimations of the gender of the surrounding users in real time, even though the user was never seen before.

Discussions with potential users have given interesting results. On the one hand, users acknowledge the usefulness of detecting their gender for a personalized interaction. On the other hand, once told the robot considers seamless the shape of their torso to guess their gender, some users, especially women, have shown concerns about their privacy and how measuring their chest shape is an intrusive method, thus meeting the results of [Cai et al., 2010]. However, once shown the output of the algorithm, some of these users have reported to have worse expectations than reality: the sensor and the algorithm are not as intrusive as infra-red cameras. These first conclusions now give way to deeper research using a proper experimental framework and a bigger set of users.

5.2.4.iv Conclusions on the breast detector

The proposed algorithm turns out to be a useful metric as it helps to determine the gender of new users. It gives a generally liable estimation of it, and its lightweight nature makes it perfectly suitable for social robots.

Possible extensions of this detector would be to study more in details the a-priori and a-posteriori privacy concerns of the users. Determining how the use other techniques such as gender-from-face recognition improves the accuracy of the gender estimation would also be an interesting topic.

5.2.5 *PersonHistogramSet*: user recognition based on structured Hue histograms

In this part, we will achieve recognizing users thanks to their visual appearance, more specifically to the color of their clothes.

As seen in subsection 5.1.3, page 96, color histograms have been used extensively for user recognition. However, they often do not take into account the fact that this color data in a person is structured. For instance, its position can be divided in three parts: head, upper body (torso and arms), and lower body. Some articles represent the user's color distribution as a set of histograms, but with a constant height step [Mittal and Davis, 2003], therefore slices do not correspond to physical body parts (head, torso, limbs, etc.). The work presented here aims at generating a set of Hue histograms structured so as to represented a natural segmentation of the human body. This set of Hue histograms is called a **PersonHistogramSet** (and abbreviated PHS).

5.2.5.i Preliminary: Masks and vector of histograms

Histograms can be computed over any one-channel image, as seen in subsection 5.1.3, page 96. A histogram is usually computed over all the values of the image, or in some cases over a defined region of the image. However, if the object we want to describe has a known structure, it makes sense to describe it thanks to several histograms, each of them describing a distinct part of this structure.

For instance, a user of the robot can be characterized by the color of its clothes thanks to the following structure: head, torso (shirt, sweater, bare torso...), and legs (trousers, skirt...). This would result in three histograms.

To be computed from an image, such a vector of histograms needs a *"mask"* describing to what part of the structure, i.e. to what histogram, each pixel belongs. The previously described description of human clothing would then need a mask consisting of 4 values: head pixels, torso pixels, leg pixels, and pixels not belonging to the user.

It is then possible to combine several masks into one so-called **Multi-mask**, that will generate several histograms at once. It is an array of integers, of the same size than the input image, where the value of a pixel corresponds to the histogram to which the input image pixel value should contribute [6]. A value of 0 means the value of the pixel will not be used. The number of histograms then corresponds to the number of unique values in the multi-mask, 0 excluded. As such, if the pixel $(10, 10)$ has a value of 2 in the multi-mask, then its hue value in the input image should be used for the computation of the second histogram.

The following example Figure 5.18 shows how a mask or a multi-mask can influence the

[6] Readers with a good memory will remember that the concept of multi-mask was already met concerning the data supplied by the Kinect, in subsection 3.1.3, page 36. In that part, it was also referring to an integer image, the value of each pixel indicating to what user it belongs (or 0 if it does not correspond to a user). Both defintions then match, as they refer to the same data structure type, and have the same purpose: give a structure to the input images.

histogram obtained on an image.

5.2.5.ii Computation of a PersonHistogramSet (PHS)

We have seen how to compute a vector of histograms based on a single color image and a multi-mask. Several algorithms seen in chapter 3, such as the NiTE API subsection 3.1.3, page 36, or the algorithm based on a seed and the depth mask seen in subsection 3.2.1, page 39, supply a so-called *user mask*, i.e. a binary image that indicates all pixels belonging to the user.

A user is naturally divisible into several "morphological parts", such as head, torso, legs, arms and feet. We here choose to divide the user in three parts: **1:** head; **2:** torso; **3:** legs.

If we manage to segment this user mask into different parts that correspond to the morphological parts of the user, we can then generate one histogram per part of the user. Our goal is then to generate a user mask that contains four different values: 0 where the user is not visible (outside of the mask), 1 for the head, 2 for the torso, 3 for the legs.

On the one hand, thanks to the anatomy rules seen in subsubsection 5.1.2.i, page 95, the different parts of the body relate one to another in size. As such, if we know the user mask, the position of the head of the user, and the height of her head, we can define the approximate position of the different parts of the body. For instance, the torso is between one head and four heads of distance from the top of the head of the user.
We have previously seen in subsubsection 5.2.3.ii, page 111 how to locate the head of the user in a binary mask, and in subsection 5.2.3, page 108 how to determine its height. So, we know explicitly the range of pixel distance from the top of the head that corresponds for instance to the torso. For example, if the user height is 400 pixels, then the user head is about $\frac{400}{8} = 50$ pixels, and so from the top of the head, the torso is located between 50 and $50 + 50 \times 3 = 200$ pixels.

On the other hand, we had seen in subsection 3.2.1, page 39 how to perform a floodfill propagation from the depth image and a seed pixel belonging to the user. We here use a similar algorithm, using the top of the head as the seed. The difference is that the mask is considered to be already computed here, as mentioned in the previous paragraph.

Coupling the floodfill propagation and the anatomy rules will generate a multi-mask, containing the three values mentioned before (head, torso, legs). We leave some gaps between the parts, to avoid ambiguous zones such as the neck or the belt. The gaps between each of these three parts can be easily re-configured. Coupled with the Hue image of the user, the multi-mask is then used for computing the three Hue histograms of the user at once. The triplet of histograms thus generated consists of a descriptor of the user, that we call **PersonHistogramSet** (PHS). This descriptor can then be used for user recognition, as explained in the next paragraph. A sample of user histograms computation is shown in Figure 5.19.

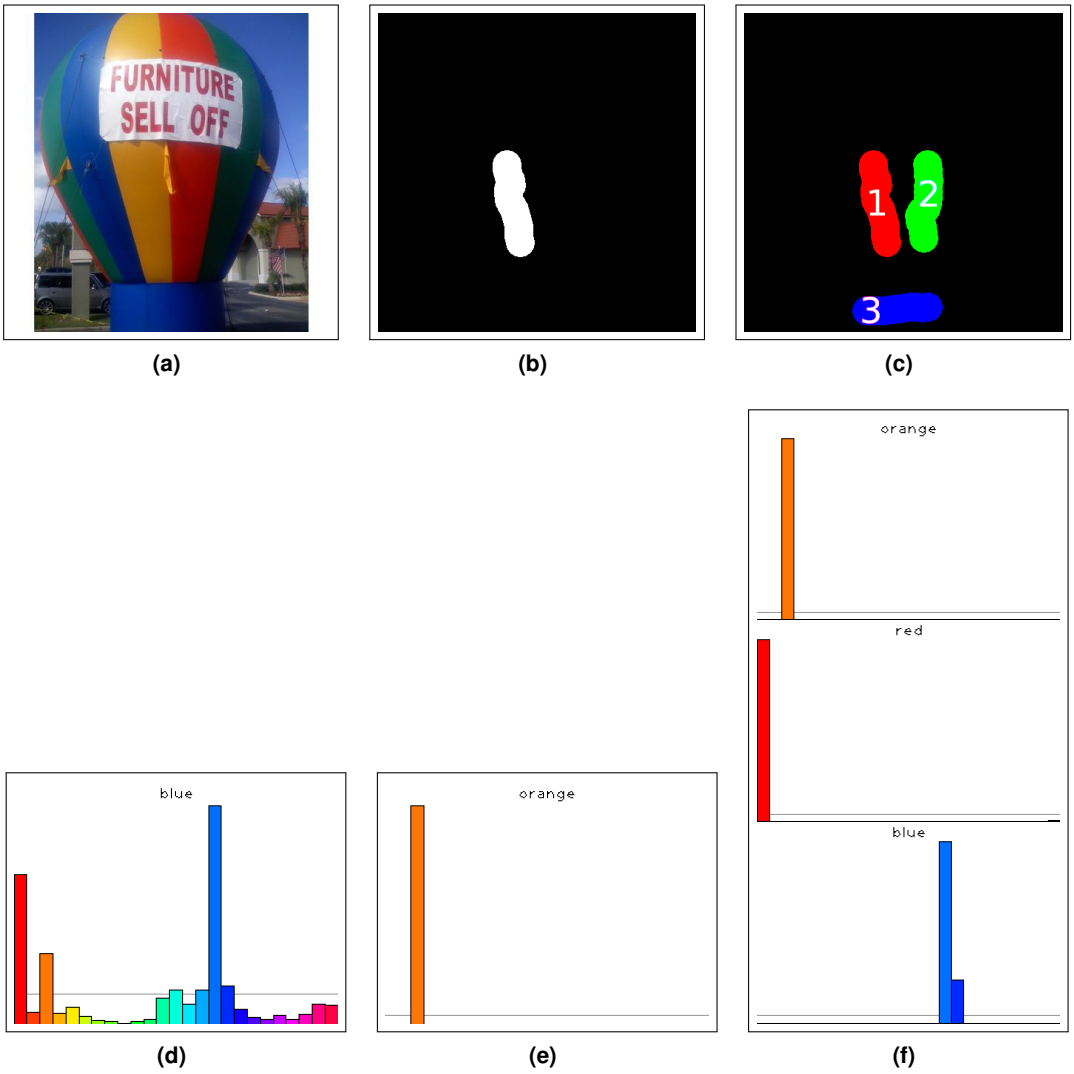


Figure 5.18: *The effects of masks on histogram computation.*
(a) is the original image, and (d) is its hue histogram.
(b) is a binary mask on the original image (a) and (e) the histogram of this image using this mask. Note how the histogram of the mask is different from (d) and only represents the content of the image corresponding to the mask, in that case the yellow-orange stripe of the balloon.
(c) is a multi-mask on the original image (a), combining several values that will result in several histograms, and (f) the resulting histogram collage on the original image (a). Note how each histogram describes the corresponding zone of the multi-mask.

Figure 5.19: *Multi-mask generation and histograms computation. The inputs are the user RGB image (a) and the user binary mask (b).*
(c) illustrates the floodfill made from the head seed. The head seed is drawn as a circle. (d) corresponds to the generated multi-mask by applying the anatomy rules. Note the gaps between the three parts, to get rid of the ambiguous zones such as the belt and the neck. Finally (e) shows the generated histograms for this input data. The title of each histogram corresponds to the dominant color of this histogram, therefore it is the dominant color the associated body part: in this example, we can verify that the shirt is mostly green.

5.2.5.iii User recognition with PHSs

As explained before, a PHS is a color descriptor of a given view of a given user. It encodes a powerful description of the visual appearance of the user, that is structured in space. It is a constant length feature, where the length is the sum of the number of bins used for each histogram. Furthermore, this feature is somewhat compact: our implementation uses 15 bins per histogram, which results in a feature size of 45 floating numbers, i.e. $45 \times 4 = 180$ bytes of information.

PHS distance: d_{PHS} The similarity between two PHS can be evaluated thanks to the histograms distances seen in section 5.1.3.i, page 100. As a PHS is a set of histograms, we define the PHS distance d_{PHS} as the sum of corresponding pairs of histograms. That is, given two PHS P and P', writing $P = (h_1, h_2, h_3)$ and $P' = (h'_1, h'_2, h'_3)$, the distance between P and P' is:

$$d_{PHS}(P, P') = \frac{1}{3} \sum_{i=1}^{3} d(h_i, h'_i)$$

where $d(\cdot, \cdot)$ is one of the histogram distances, for instance the intersection distance. Two similar PHS will then get a d_{PHS} distance close to zero, and two very different ones will have a distance close to one. This distance now gives use the possibility to recognize users.

User recognition We just saw how to evaluate the similarity of two PHS using a specific distance called d_{PHS}. We now use this distance for user recognition.

We suppose here we have a system of labels that identify uniquely a physical user. They could be a numerical ID or a descriptive string such as *"ArnaudRamey"*.

We also suppose we have a set of n PHS $\mathbf{P} = (P_i, i \in [1, n])$ with their matching user labels $\mathbf{L} = (l_i, i \in [1, n])$. A user can have several PHS associated with her, obtained thanks to several views of her. For instance, we can have three PHS of user A, one PHS of user B, ten PHS of user C, etc. The number of known users then corresponds to the number of unique labels in $\mathbf{L}$.

The recognition of the user $\tilde{l}$ associated with a new PHS $\tilde{P}$ is obtained easily: it corresponds to the user label that minimizes the distance with the known PHS:

$$\tilde{l} = \left(l_j | j = \underset{i \in [1,n]}{\arg\min}\, d_{PHS}(\tilde{P}, P_i) \right)$$

Now that we have defined how to perform user recognition based on PHS, we will present some benchmarking results of this technique.

5.2.5.iv Benchmarking of PHS-based recognition

For benchmarking user recognition based on PHS, we make use of the DGait dataset [Igual et al., 2013] already presented in subsubsection 5.2.4.ii, page 120.

It contains 55 videos of unique users, 35% of which are female. For each user, we generated 20 training user images and 10 test user images. Consequently, the dataset $\mathbf{P}$ contains $55 \times 20 = 1.100$ PHS, each of them associated with its user label, which belongs to $[1, 55]$. Some samples of the user views are visible in Figure 5.20. Note how some users have similar clothing colors, such as grey or black clothes.

The algorithm is then run on training user images to estimate the most likely user. A match is considered as correct if the output label corresponds to the expected label. Results are visible in Table 5.5. We obtain an accuracy of 100%, which means the algorithm successfully recognizes the correct user among 55 in all cases. Taken in consideration the similarity of some users, the good performance of this user recognition module is of great interest for performing user recognition in challenging conditions, such as crowded environments.

Figure 5.20: *A few samples of the users in [Igual et al., 2013] dataset. Some users have very similar clothes colors and patterns. This makes the user recognition challenging.*

Unique users	55
Training images	1.100
Test images	550
Correct matches	550
Accuracy	100%

Table 5.5: *Benchmark results for PHS user recognition.*

Summary of the chapter

In this chapter, we have seen different methods for user recognition based on vision. Several of these can also be used to obtain hints about the user gender, which helps for her identification.

Face recognition (subsection 5.1.1, page 91) is a key algorithm: it is a powerful tool for long-term recognition of the users. However, it requires training images, whose acquisition by the robot can be cumbersome for the user. Gender-from-face recognition can also give hints of the gender of unknown users, and the training data is much easier to obtain, as it is not user-specific (subsection 5.2.1, page 103).

Morphological features of the user can be used, such as her height (subsection 5.2.3, page 108) or the shape of her body (subsection 5.2.4, page 118). Moreover, the visual appearance of the user is a powerful short-term recognition algorithm, that can be characterized with color histograms (subsection 5.2.5, page 122)

However, each of them has a given domain of use where it works better, and none of these methods have an accuracy rate enough to be used alone as a user recognition system. However, they can run in parallel and be coupled for getting more accurate results, which will be seen later on, in chapter 7.

CHAPTER 6

Other techniques for user recognition

Introduction

The first part of this PhD is focused on detecting users thanks to various methods using a variety of sensors and algorithms. This part instead aims at giving temporal coherency to these detections, in other words, recognizing the users across time. The previous chapter focused on vision-based techniques to reach that goal, for instance, face recognition or descriptions of the color of the clothes.

Note that recognizing the user without using her spatial appearance is a challenging task: if we draw a parallel with how humans perform this task, they mainly rely on their vision. This explains the brevity of this chapter: many relevant techniques were already presented in the previous chapter, and they represent most of the work done in user recognition in the scope of this PhD. However, the voice of the user is also a discriminant feature: we can recognize a user according to her voice. RFID tags and other kind of tags can also be used.

6.1 State of the art

Voice recognition and user speaking detection In Human-Robot Interaction (HRI), users who want to interact with the robot will most likely intend to communicate with it using their voice. This is a precious clue about who the user is.

Using voice for user awareness was already considered in chapter 4, page 67. We then

treated how to detect if somebody is speaking in an audio stream, which is called Voice Activity Detection (VAD). We also presented how to locate spatially this talking person, using an array of microphones: the distribution of volumes on the different microphones will give us the approximate angular position of the talking user.

On top of these techniques, the voice is also a powerful characterization of each user: just like her face, the voice of a user is a way to recognize her among others.

Acoustic fingerprinting: Just as image features are compact descriptions of the characteristics of an image, audio features are numerical descriptors of a given audio sample. They can be used to describe accurately and in a compact way a voice sample. For instance, an acute voice and a low-pitched voice differ in their pitch: the pitch can be used as an audio feature. More generally, audio features are used to recognize an audio sample among a set of known ones. A collection of different features is presented in [Cano et al., 2002]. Audio features can be extracted in three domains: time, frequency and time frequency.

Overview of user voice recognition systems: The general structure for voice recognition system is presented in [Hunt and Schalk, 1996]. These systems are most often made of two phases:

1. *Enrollment:* during the enrollment phase, the user registers in the system. This phase is most often made of several questions the user needs to answer, so that the system learns how the voice of the user sounds like.
2. *Identification (searching) phase:* once the enrollment is done, the user can be identified by the system. When she starts speaking, the description of her voice will be used as a way to recognize her.

The enrollment can be either *text-dependent* or not. A text-dependent enrollment system requires the user to record a vocal passphrase, that will then be re-used for the identification. With a text-independent system, this constraint does not exist: the user can say any sentence so as to be recognized by the system.

For instance, the previous user identification system of our robot Maggie, presented in [Alonso-Martín et al., 2013], was a text-dependent system: This phase consisted of questions asked by the robot to the user. Along with her name, age, language, a key-pass sentence is chosen by the user to learn her voice tone. It can be a digit password or anything else. For this task, we used the third-party Loquendo ASR-Speaker Verification package ([Dalmasso et al., 2009]). The main drawback was that, to be correctly identified, the user needed to utter the same sentence that he used in the enrollment phase.

In this work, we instead focused on developing a speaker recognition system not based on a passphrase, but using any utterance of the user: the features computed on whatever the user says are used to match her against the registered users.

Tags The different kinds of tags that can be used for user detection and recognition were dealt with in subsection 4.1.1, page 68. Tags such as RFID tags or ARToolkit not only are easily detected by their respective readers, but in addition they convey information about the detected tag, such as a unique ID. This can be used as a way to recognize users. For more details, refer to this previous chapter (page 68).

6.2 Research contribution - text-independent user voice identification

6.2.1 Description of the system

As mentioned before, the previous system used by the robot Maggie required the user to enroll on the system. In order to avoid this drawback, we implemented a new module of *text-independent* user identification based on *pattern matching* techniques. During a similar question-based (but key-pass free) enrollment phase, the robot learned features of the voice of the new user. These features are stored into a so-called voiceprint file. Details about feature extraction and matching come below.

The work presented in this chapter includes user identification using voice features. It is implemented in Chuck and C++. It is part of a complete multimodal interaction system applied to HRI called *Robotics Dialog System* ([Alonso-Martin and Castro-González, 2013]). This RDS interaction system is based on the ROS architecture, detailed in the introduction chapter, and supplies the interaction capabilities for the RoboticsLab robots, also presented in the introduction.

Used features Two aspects of the new system can be underlined: the features are extracted without needing a specific key-pass phase, and these extracted features belong to three different domains: time, frequency, and time-frequency (more details about them in [Alonso-Martin and Castro-González, 2013]). The used features are *Root Mean Square (RMS)* (computed on time domain); *Pitch computed using Fast Fourier Transform* (frequency domain); *Pitch computed using Haar Discrete Wavelet Transform* (time-frequency domain); *Flux* (frequency domain); *RollOff* (frequency domain); *Centroid* (frequency domain); *Zero-crossing rate (ZCR)* (time domain).

Real time feature extraction Once the voiceprints of the enrolled users have been generated and stored, it is possible to identify which user is speaking at any time. *Voice Activity Detection*, which consists in detecting whether users are talking or quiet, is based on our previous work [Alonso-Martin and Castro-González, 2013]. When voice activity is detected, the previously listed audio features are extracted. Each second of voice generates several rows of features. Each row is called a *bit*, and a set of *bits* is known as *sub-fingerprint*. The number of bits that compose a sub-fingerprint will be called *sub-fingeprint size*, and is the minimum information to identify a user.

Our system works with a sample rate of 44100 Hz and a window size of 4096 samples, therefore each bit of fingerprint corresponds to 0.1 second of voice (i.e. 1 second of voice generates 10 bits).

Identification (matching) phase The identification is made by computing the distance between the current audio features and the features of the enrolled users, stored in each of the voiceprint files. If the best-match distance is below a threshold, the speaking user is considered to be the user that generated that voiceprint file.

In mathematical terms, let $n = 7$ the number of different audio features. The distance d between a current *bit* C of n extracted features and a voice print file V, a set of n_v bits of n features, is computed with the following formula:

$$d(C, V) = \min_{j \in [0, n_v]} \|[1 \dots 1] - \exp(-\alpha \circ |C - V_j|)\|_{L_1}$$

where $\|.\|_{L_1}$ is the Manhattan distance, $\circ$ is the element-wise product, and $\alpha = [\alpha_0 \dots \alpha_n]$ a scaling vector.

The best match for a given bit C is then defined as the voice print which bits obtain the minimum distance with C. It is considered a valid match if that distance is below an empirical threshold.

The distance computation presented above is called *bit-to-bit* ($n \times 1$ vectors comparison). It can also be made on successive sets of *bits* ($n \times w, w \in \mathbf{N}$ matrices comparison, w is called the sub-fingerprint size).

6.2.2 Integration as a `PeoplePoseList` Matcher

The technique we just describes is capable of recognizing the speaker, but it needs as input a sample of the user voice. In other words, we first need to detect if there is any user, and if he or she is speaking, so that we can create a voice sample.

Voice Activity Detection (VAD) was presented in subsection 4.2.2, page 78: we then presented how we could detect speaking users and estimate their spatial position thanks to machine learning techniques. The VAD algorithm keeps in its buffer a sample of the user voice, that can be re-used: we store the voice buffer in the `PeoplePoseList` (`PPL`) message: `PPL`s have an `attributes` field that can store any data, as defined in subsection 3.2.2.

Then, the speaker recognition algorithm is shaped as a `PeoplePoseList` Matcher (`PPLM`), as defined in the previous chapter. For each of the `PeoplePose`s (`PP`s) contained in the `PPL` message that has a voice samples stored along, speaker recognition is performed. When a user is tracked, this voice sample is stored permanently for future references. That way, upon reception of a `PPL` with voice samples, we can compute the cost matrix for each pair of (detected user, tracked user). This pipeline is illustrated Figure 6.1. Note that this pipeline corresponds to a robot capable of running the voice localization `PeoplePoseList` Publisher (`PPLP`): most

notably, it requires an array of microphones, which is what is equipping Maggie, presented in the introduction Chapter. However, VAD could be performed with a single microphone. In that case, some refactoring would be required to create a minimalistic VAD PPLP.

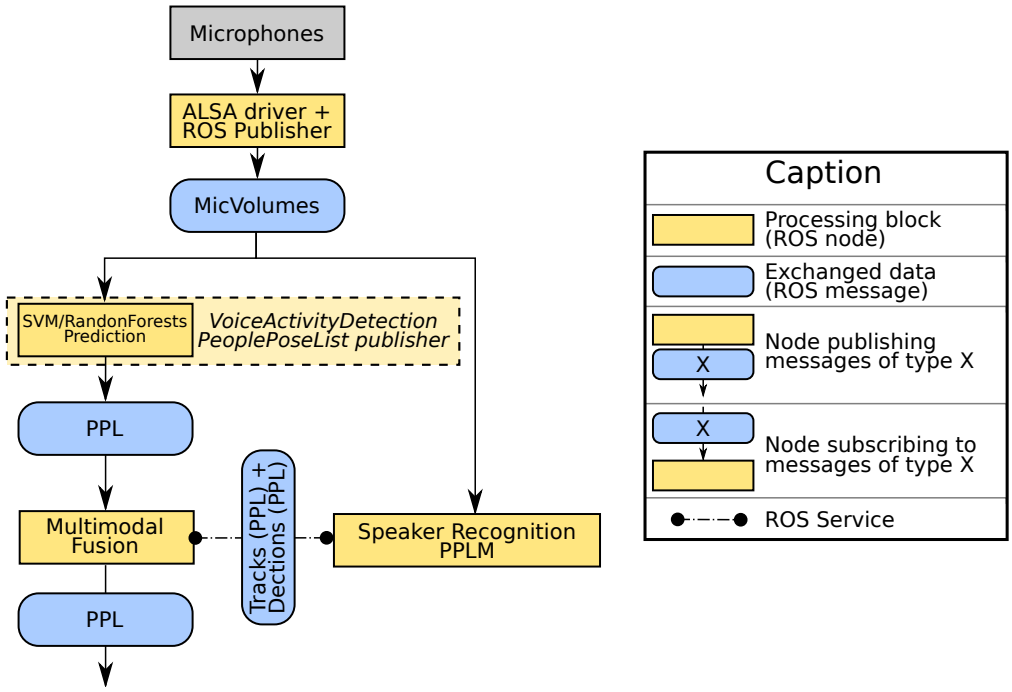


Figure 6.1: *The pipeline for the speaker-recognition `PeoplePoseList` Matcher.*

6.2.3 Experimental results

In order to verify the system accuracy to identify human peers, we have used a multi-language voice database [1]. Some samples were used for generating the voiceprints, and other for evaluating if their best match corresponded to the correct user.

Figure 6.2 relates the number of enrolled users, the sub-fingerprint size, and the accuracy rate of the user identification. The accuracy rate is close to 100% for few users, and while it decrease while the number of users increase, it remains over 70% for as many as eighteen users for the best choice of sub-fingerprint size. The best results are obtained using a sub-fingerprint size of 5. In other words, a voice sample of half a second is long enough to identify the speaker. Similar work, presented by Lite ([Li and Jr, 1982]), claims an accuracy rate of 79% for 11 users and 3 seconds for each identification. Our system, with 11 users gets an accuracy rate of 86% using 0.5 seconds of voice.

[1] The database used was VoxDB (`http://voxdb.org`).

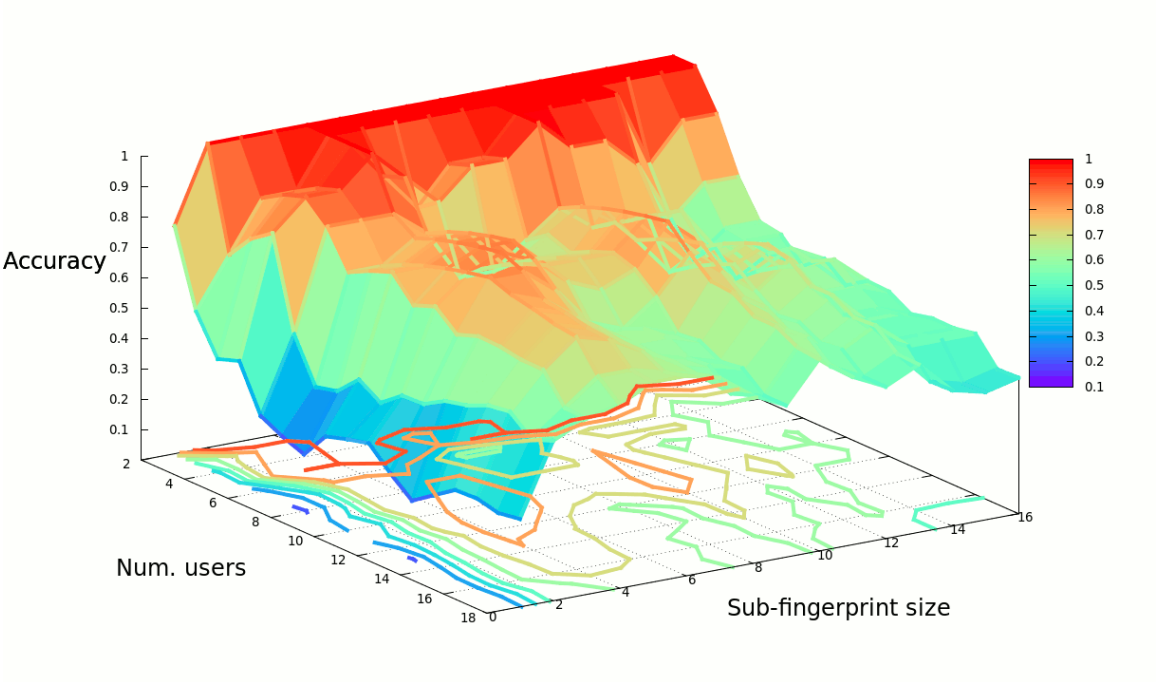


Figure 6.2: *User recognition thanks to voice analysis. The horizontal axes are the number of users and the sub-fingerprint size. In z, the accuracy of our system. We can see an optimal accuracy for a sub-fingerprint of size 5, whatever the number of users.*

Summary of the chapter

This chapter focused on user recognition using a technique that does not use image processing or computer vision, but instead processes the audio stream: speaker recognition. Speaker recognition systems are most often made of two steps: enrollment and identification. The former consists in registering the user in the database: the appearance of her voice is learned thanks to the analysis of samples of her voice. The latter matches an unidentified speaker to an enrolled one, by analyzing her voice. Most existing techniques focus on a text-dependent technique, that forces the user to say a given sentence to be recognized. This strategy lacks of flexibility: for the Human-Robot Interaction (HRI) to be natural, the robot must understand the user speaking to it the way she would do with a fellow, not a machine.

We developed our own user recognition system. So as to obtain the most intuitive interaction flow, It is a text-independent technique: the user can be recognized thanks to any sentence she utters, even if she was not enrolled using these words. The recognition is based on the computation of seven distinct powerful audio features in three domains: time, frequency and time-frequency.

Part III

Data fusion and user mapping

CHAPTER 7

Data fusion and user mapping

Introduction

The work presented in this PhD dissertation aims at giving user awareness to the robot. In other words, we want the robot to perceive the users around it, where they are, and who they are. The first part, Part I, presented a wide range of people detectors, mainly based on computer vision techniques. All of them are shaped as so-called `PeoplePoseList` Publishers (`PPLP`s): they are Robot Operating System (ROS) nodes, publishing a common message type called `PeoplePoseList` (`PPL`). In other words, a detected `PPL` does not correspond to a physical user, but only to a probable user presence at that time and place.

We now need to give spatial and temporal consistency to these detections: detections corresponding to the same physical user should be grouped under a same structure. Two different ways of reaching this goal have been explored: matching detections using the user recognition algorithms presented in Part II, or using a more ad-hoc tracking algorithm needing only a seed given by a `PPL` and using two-dimensional (2D) techniques on the depth image.

The first method relies is made of two stages. First, the different detection algorithms, presented in Part I and shaped as `PPLP`s, process in parallel the stream of data coming from the sensors, and publish their results shaped as `PPL`s. Then, a so-called *fusion node* subscribes to all these `PPL`s streams. It create and maintain the set of trajectories of these tracked users, called *tracks*, which is shaped as a `PPL`. To do so, this fusion node calls the different recognition algorithms, presented in Part II, that enable the matching of a `PPL` coming from the detection algorithms against the `PPL` of tracks. Indeed, these recognition algorithms can say to what

extent a PeoplePose (PP) detected in the current data corresponds to a previous one. Using these recognition blocks, this first method will now give temporal consistency to the PPs detected thanks to the PPLPs.
The number of users, their position, and who they are, is a set of unknown variables that we can call *state* of the system. The goal of the user awareness can then be seen as estimating this system state as accurately as possible. The different detection algorithms (PPLPs) give an estimation of this state, thanks to different input data (e.g. color video stream, laser range finder scans), or different algorithms (for instance face detection, leg pattern detector). A more precise state estimation can be obtained by merging the results of all these algorithms in a clever way: this is what is called *multimodal fusion*. This first method for people tracking is modular, but complex.

On the other hand, in some situations we might be only interested in detecting and tracking a specific user, without needing to compute additional information about the other ones. *Our second approach* will use this strategy: it aims at moving the robot to follow closely a chosen user. It is based on a user initial position, also called *seed*, and a multimodal tracking algorithm based on both 2D image processing techniques on the depth image and laser scan processing. Indeed, the depth image gives important hints about where the user is in three dimensions, but is limited by its narrow field of view, while the laser range finder has a wide Field of View (FOV) but is a one-dimensional (1D) sensor.

This chapter follows the same structure as the previous ones: in section 7.1, we will review the most relevant existing algorithms in multimodal fusion. Then, in section 7.2, we will present our own two approaches for achieving this robust state estimation, thanks to the PPL architecture and recognition algorithms, or using a seed and 2D techniques on the depth image.

7.1 State of the art

This part will present the most relevant algorithms for multimodal fusion. In other words, these techniques allow the estimation of the state of a system, made of several unknown variables, thanks to the imperfect estimation of several parallel algorithms (detectors and recognizers). These algorithms are based on the analysis of different types of inputs, or use different techniques to analyze them, hence the *multimodal* name. The action of these fusion algorithms can then be seen as a clever blending of the results of different algorithms.

These fusion algorithms can be split in two families. First, *particle filters* will be presented in subsection 7.1.1. Then *Kalman filters* will be explained in subsection 7.1.2.

7.1.1 Particle filters

Particle filters are a technique to estimate the unknown **State of the system** $x \in \mathbb{R}^n$, where n is the number of unknowns, given of set of *observations*, which are measures giving hits about

the state. For instance, in a simple single-user detection case, the state x could be made of the three-dimensional (3D) coordinates of the user: $n = 3$, and the observations can be the presence of a face or not.

We suppose that the observations and the state are related by some functional form that is known. Similarly, the statistical evolution of the state through time is modeled by a known function.

The objective of the particle filter is to estimate the value of the state, using the visible values of the observation process. A particle filter does not estimate x itself, but gives a density estimation of x: in each point of the state space $\tilde{x} \in \mathbb{R}^n$, a particle filter gives the probability $p(x = \tilde{x}) \in [0, 1]$.

Sequential importance resampling (SIR) is the original particle filtering algorithm and was presented by Gordon in 1993 [Gordon et al., 1993]. It is a particle filter commonly used in robotics.

A particle is an estimate of the state of the system: it is made of an approximate $\tilde{x} \in \mathbb{R}^n$. A particle filter is then made of an important number of these particles. Note that the estimation in each particle will not change during its life scope. On the other hand, these particles are removed, created or merged according to the detectors output. In other words, the particles having a state estimation coherent with the output of the detectors will tend to be kept and multiplied, while the particles having a state that is not confirmed by the detectors output will tend to be removed. As such, these particles represent an exploration of the state space, and the most relevant ones are kept. SIR in particular, and particle filters in general, are heavily used in robotics for estimating hidden states: see for [Thrun, 2002] for a good state of the art.

They have been applied for user detection and tracking too: particle filters are relevant for this domain, as they give the possibility of modeling different system hypotheses at the same time. Some examples can be found in [Schulz et al., 2003] [Montemerlo et al., 2002], [Zhou, 2004] or [Muñoz Salinas, 2009].

7.1.2 Kalman filtering

Similarly to particle filters, the **Kalman filter** is another way to estimate the state of a given system, using multiple uncertain measurements. The state obtained is more accurate than it would be with any of the measurements alone. Conceived by the Hungarian émigré *Rudolf E. Kálmán*, the theory was developed and gained popularity during the fifties and sixties. It is then a common sensor and data fusion algorithm. A detailed summary is available in [Bishop and Welch, 2001].

As pointed out in [Julier and Uhlmann, 2004], the Kalman Filter (KF) only utilizes the first two moments of the state (mean and covariance) in its update rule. Although this is a relatively simple state representation, it offers a number of important practical benefits. The fusion mechanism presented in this PhD is based on Kalman filtering. For this reason, more detailed explanation about Kalman filters will be given than about particle filters.

The original KF algorithm will be presented first. Note that the classical KF needs the system to be linear: in other words, noting $x \in \mathbb{R}^n$ the state of the problem, exists a matrix A of size $n \times n$ such as $\frac{d}{dt}x = Ax$. Two extensions of the original KF for non-linear problems will then be introduced: the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF).

7.1.2.i Classical KF

The Kalman filter, as formulated by Kalman and others in the sixties, is made for estimating the **State of the system** $x \in \mathbb{R}^n$, using the values of the measurements. The measurements are not supposed to be exact: in other words, they can be noisy.

Because of the algorithm's recursive nature, it can run in real time using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required.

Assumptions about the underlying system: Some assumptions are made on the system:
1. It must be a linear dynamical system, e.g., there is a matrix $A \in \mathbb{R}^n$ such as $\frac{d}{dt}x = Ax$;
2. Error measurements and terms must have a Gaussian distribution.

Model The Kalman filter assumes the true state at time k is bound with the one at time $k-1$ using the following formula:

$$x_k = A \times x_{k-1} + B \times u_k + w_{k-1}$$

Matrix A is of size $n \times n$ and is called the *state transition model*, which is applied to the previous state.
Matrix $B \in \mathbb{R}^{n \times l}$ relates the optional *Control input* $u \in \mathbb{R}^l$ to the state x.
Here $w_k \in \mathbb{R}^n$ is the *Process noise*. It is assumed to be with normal distribution: the *Process noise covariance* is called $Q \in \mathbb{R}^{n \times n}$: $\mathbf{p}(w) \sim \mathcal{N}(0, Q)$

On top of this first model, we also have another model, linking the (invisible) real state of the system and the visible measurements. This model, called *measurement equation* or *Observation model*, is:

$$z_k = H \times x_k + v_k$$

$z \in \mathbb{R}^m$ is the *Measurement*. Here $v_k \in \mathbb{R}^m$ is the *Measurement noise*. It is assumed to be with normal distribution: the *Measurement noise covariance* is called $R \in \mathbb{R}^{m \times m}$: $\mathbf{p}(v) \sim \mathcal{N}(0, R)$
The $m \times n$ matrix H in the measurement equation relates the state to the measurement z_k.

Computation Let us now suppose that we have modeled our system and that we have in real time measurements $z_k \in \mathbb{R}^m$. Our goal is to estimate the (hidden) real state of the system $x \in \mathbb{R}^n$. The computation of the KF is made of two steps:

1. Prediction step (time update equations): estimate of the current state variables and their uncertainties.

2. Update step (measurement update equations): when the next measurement output is known, update of the estimates with weighted averages. The purpose of the weights is that the values with better, i.e. smaller estimation uncertainties, are trusted more.

Computation details Here come some details about the computations involved in the KF.

Prediction step: We define $\hat{x}_k^- \in \mathbb{R}^n$ (note the "super minus") to be our **A priori state estimate** at step k given knowledge of the process prior to step k.

The projection equation for state x:

$$\hat{x}_k^- = A \times \hat{x}_{k-1} + B \times u_k$$

The projection equation for estimate covariance $P \in \mathbb{R}^{n \times n}$ (involving Q, the covariance of process noise w):

$$P_k^- = A \times P_{k-1} \times A^T + Q$$

A few properties of this estimate covariance matrix: P is positive-semidefinite and symmetric.

Update step: We define $\hat{x}_k \in \mathbb{R}^n$ to be our **A posteriori state estimate** at step k given measurement z_k. The **Estimate update equation** (from a priori to a posteriori estimate) is written (calling $I_{n \times n}$ the identity matrix of $\mathbb{R}^{n \times n}$):

$$\begin{aligned} \hat{x}_k &= \hat{x}_k^- + K_k \times (z_k - H \times \hat{x}_k^-) \\ &= K_k \times z_k + (I_{n \times n} - K_k \times H) \times \hat{x}_k^- \end{aligned}$$

$(z_k - H \times \hat{x}_k^-) \in \mathbb{R}^m$ is called the **Residual** or **Innovation**, it reflects the difference between the predicted measurement $H \times x_k^-$ (cf measurement equation) and the obtained measurement z_k.

The $n \times m$ matrix K_k in the estimate update equation is chosen to be the **Gain** or **Blending factor** that minimizes the a posteriori estimate covariance equation.

Gain computation: an explicit formulation for $K_k \in \mathbb{R}^{n \times m}$ can be determined:

$$\begin{aligned} K_k &= P_k \times H^T \times (H \times P_k^- \times H^T + R)^{-1} \\ &= \frac{P_k \times H^T}{H \times P_k^- \times H^T + R} \end{aligned}$$

We finally obtain an a posteriori estimate covariance estimate thanks to the update equation for estimate covariance:

$$P_k = (I - K_k \times H) \times P_k$$

The steps are presented in Figure 7.1.

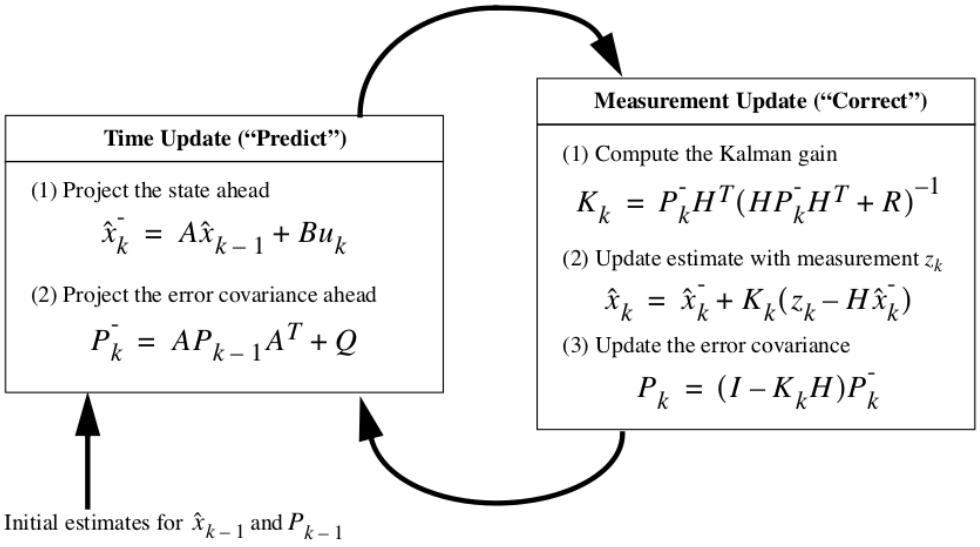


Figure 7.1: *The steps of KF computing (from [Bishop and Welch, 2001])*

Limitations As pointer out previously, Kalman filtering can only be used in case of linear problems. If the problem we want to solve is not linear, we need to use an extension of it for non linear problems. Two extensions will now be presented, *Extended Kalman Filter* and *Unscented Kalman Filter*.

7.1.2.ii EKF

Extended Kalman Filter was developed between 1959 and 1961 by Kalman, in an attempt to solve the main drawback of the original KF: it only works for a linear system, while most engineering problems are non-linear.

EKF thus can estimate the state of the system, even for a non-linear system. The non-linear function linking the state and its first order derivative is called f:

$$\hat{x}_k^- = f(x_{k-1}, u_k) + w_{k-1}$$

Function f can be integrated with numerical methods to obtain the a priori estimate of the state. However, the error covariance bound with this estimate cannot be evaluated directly, this

is why a matrix of partial derivatives, the *Jacobian matrix*, must be computed to linearize the function. EKF is made of the same two step than the original KF, prediction and update, except the Taylor linearization is made in the prediction step.

Limitations It suffers from limitations underlined in [Julier and Uhlmann, 2004]: 1. the EKF can only be applied if the Jacobian matrices of f are defined; 2. the computation of Jacobian matrices is a complicated and error-prone process.

7.1.2.iii Unscented Kalman Filter

The Unscented Kalman Filter (UKF) is another extension of the original KF for non-linear problems. The UKF was presented for the first time in [Wan and Merwe, 2000], and developed in [Julier and Uhlmann, 2004]. It works for a non-linear system.

It is based on the idea that it is easier to approximate a probability distribution than to approximate an arbitrary non-linear transformation.

Model The transformations are now non-linear. $x \in \mathbb{R}^n$ is the state of the system.

Prediction model: we suppose the state regulated by a function $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$ such as:

$$x_k = \mathbf{f}(x_{k-1}) + w_{k-1}$$

This function $\mathbf{f}$ can include a process command (which was written separately in the linear KF). $w_k \in \mathbb{R}^n$ is called the process noise, as in the classical KF. We call $Q \in \mathbb{R}^{n \times n}$ the covariance of the additive process noise ($\mathbf{p}(w) \sim \mathcal{N}(0, Q)$).

Observation model: The observation model is no longer linear (the H matrix of the linear KF) but defined by an explicit function $\mathbf{h} : \mathbb{R}^n \mapsto \mathbb{R}^m$:

$$z_k = \mathbf{h}(x_k) + v_k$$

$v_k \in \mathbb{R}^m$ is still called the measurement noise. We call $R \in \mathbb{R}^{m \times m}$ the covariance of the observation noise ($\mathbf{p}(v) \sim \mathcal{N}(0, R)$).

Computation As explained in [Bellotto and Hu, 2009], the steps are the following:

1. **Sigma points :** A set of relevant points, called **Sigma points**, are computed. They are chosen so that their mean and covariance are $\bar{\mathbf{x}_k}$ and Σ_x, the ones of the system state. The non-linear function is then applied to each of these sigma points to convert them into a cloud of points. The statistics of these transformed clouds are then computed, and give an estimate of the non-linearly transformed mean and covariance.

2. **Prediction step:** for each sigma point, we instantiate it through the process model, and obtain its transformed point.
   We then obtain the predicted mean and covariance of the state as weighted averages of the predicted means and covariances of the projected sigma points.

3. **Update step:** once we have the real measurement of the system, we can re-inject the residual to obtain the a posteriori state and its covariance.

Computation details We have the current state mean $\bar{\mathbf{x}_k} \in \mathbb{R}^n$ and its estimate covariance matrix $\mathbf{P}_k \in \mathbb{R}^{n \times n}$.

1. **Sigma points:** we have explicit formulas to compute the $2n+1$ sigma points $\{\chi_{ik} \in \mathbb{R}^n; 0 \leq i \leq 2n\}$ and their weights $\{W_{ik} \in \mathbb{R}; 0 \leq i \leq 2n\}$.

$$
\begin{aligned}
\chi_{0k} &= \bar{\mathbf{x}_k} \\
W_{0k} &= \frac{\rho}{n+\rho} \\
\chi_{ik} &= \bar{\mathbf{x}_k} + \left[\sqrt{(n+\rho)\mathbf{P}_k}\right]_i \\
W_{ik} &= \frac{1}{2(n+\rho)} \\
\chi_{i+n,k} &= \bar{\mathbf{x}_k} - \left[\sqrt{(n+\rho)\mathbf{P}_k}\right]_i \\
W_{i+n,k} &= \frac{1}{2(n+\rho)}
\end{aligned}
$$

   $\rho \in \mathbb{R}$ can be positive or negative. It provides an extra degree of freedom to "fine tune" the higher order moments of the approximation, and can be used to reduce the overall prediction errors. [Wan and Merwe, 2000] recommends to use $\rho = \alpha^2 \times (n + \kappa) - n$, with $a \in \mathbb{R}$ a parameter determining the spread of the sigma points around $\bar{\mathbf{x}_k}$ and usually set to a small positive value (for instance $1e-3$), and $\kappa \in \mathbb{R}$ is a second scaling parameter usually set to 0. When $\mathbf{x}(k)$ is assumed Gaussian, section I of [Julier, 2000] shows that a useful heuristic is to select $n + \rho = 3$. If a different distribution is assumed for $\mathbf{x}(k)$, then a different choice of ρ might be more appropriate.

   $\left[\sqrt{(n+\rho)\mathbf{P}_k}\right]_i \in \mathbb{R}^n$ refers to the i-th row or column of the matrix square root of $(n+\rho)\mathbf{P}_k \in \mathbb{R}^{n \times n}$ [1] ($\mathbf{P}_k$ is positive-semidefinite and symmetric so it only has one positive-definite square root, which can be called its principal square root).

[1] If the matrix square root $A \in \mathbb{R}^{n \times n}$ of $\mathbf{P}_k$ is of the form $\mathbf{P}_k = A^T \times A$, then the sigma points are formed from the rows of A. However, for a root of the form $\mathbf{P}_k = A \times A^T$, the columns of A are used.

2. **Prediction step:** each sigma point is instantiated through the process model to yield a set of transformed samples:

$$\chi_{ik} = \mathbf{f}[\chi_{ik-1}; u(k); k]$$

The predicted mean is computed as:

$$\hat{x}_k^- = \sum_{i=0}^{2n} W_{ik}\chi_{ik}$$

The predicted estimate covariance is computed as: [2]

$$\mathbf{P}_k^- = Q + \sum_{i=0}^{2n} W_{ik}\left[\chi_{ik} - \hat{x}_k^-\right] \times \left[\chi_{ik} - \hat{x}_k^-\right]^T$$

We then need to compute the expected observations thanks to the observation model. $\mathbf{P}_{\upsilon\upsilon k} \in \mathbb{R}^{m\times m}$ is called the **Observation covariance**.

$$\begin{aligned}\mathcal{Z}_{ik} =& \mathbf{h}[\chi_{ik}]\\ \hat{z}_k =& \sum_{i=0}^{2n} W_{ik}\mathcal{Z}_{ik}\\ \mathbf{P}_{\upsilon\upsilon k} =& R + \left[\mathcal{Z}_{0k} - \hat{z}_k\right]\left[\mathcal{Z}_{0k} - \hat{z}_k\right]^T + \sum_{i=0}^{2n} W_{ik}\left[\mathcal{Z}_{ik} - \hat{z}_k\right]\left[\mathcal{Z}_{ik} - \hat{z}_k\right]^T\end{aligned}$$

3. **Update step:** We can compute the cross-correlation $\mathbf{P}_{kxz} \in \mathbb{R}^{n\times m}$:

$$\mathbf{P}_{kxz} = \sum_{i=0}^{2n} W_{ik}\left[\chi_{ik} - \hat{x}_k^-\right]\left[\mathcal{Z}_{ik} - \hat{z}_k\right]^T$$

We also compute the gain $\mathbf{K}_k \in \mathbb{R}^{n\times m}$:

$$\mathbf{K}_k = \mathbf{P}_{kxz} \times \mathbf{P}_{\upsilon\upsilon k}^{-1}$$

[2] Section III of [Julier, 2000] investigates the case when ρ is negative. This is likely to happen when we try to approximate system of higher order probability distributions. Then the predicted estimate covariance $\mathbf{P}_k^-$ can happen to be non-positive. In that case, its matrix square root might not be defined, and the computation of the sigma points is impossible. To solve this issue, it is then advised to evaluate the covariance about χ_{0k}, i.e. compute the covariance as:

$$\mathbf{P}_k^- = Q + \left[\chi_{0k} - \hat{x}_k^-\right] \times \left[\chi_{0k} - \hat{x}_k^-\right]^T + \sum_{i=0}^{2n} W_{ik}\left[\chi_{ik} - \hat{x}_k^-\right]\left[\chi_{ik} - \hat{x}_k^-\right]^T$$

Once we have the real measurements z_k, we can re-inject the innovation (or residual) $\hat{z}_k - z_k \in \mathbb{R}^m$ to obtain the a posteriori state estimate and its covariance:

$$\hat{x}_k = \hat{x}_k^- + \mathbf{K}_k \times (z_k - \hat{z}_k)$$
$$\mathbf{P}_k = \mathbf{P}_k^- - \mathbf{K}_k \times \mathbf{P}_{vvk} \times \mathbf{K}_k^T$$

Advantages and limitations The UT approach conjugates a very simple approach and an approximation accurate to the third order for Gaussian inputs, as underlined in [Wan and Merwe, 2000].

However, its additional computational overhead compared with EKF can generate a discussion to determine which of the two is the most appropriate for a non linear problem.

7.1.2.iv Extension to multi-target tracking

UKF, as an extension of KF, has an intrinsic limitation: the dimension of the state must be known beforehand and cannot change during the experiment. In other words, it cannot be used for the tracking of a varying number of objects: a varying number of present objects at a given time changes the dimension of the system state.

However, UKF can still be used in such a context. The way to proceed, presented by [Rong Li and Bar-Shalom, 1996], is the following: we maintain in memory a list of UKFs, corresponding to the different tracked objects, each UKF is called a **Track**. Upon each new detection, the new measures are associated to the existing tracks. However, it can be that the number of new measures and of tracks are different: in that case, the unassociated measures are stored apart. When a critical number of unassociated measures accumulates at a given spatial position in a window of time, a new track is created at that position.

One of the key steps in a tracking problem is **Data association**. It consists in determining, upon the reception of new measurements, the track or set of tracks that will be updated with each measurement. There are several popular methods for measure-to-track assignment: 1. Assignments (explained later on); 2. (Joint) Probabilistic Data Association Filters: (J)PDAF ([Fortmann, 1983]); 3. (Global) Nearest Neighbours: (G)NN ([Reid, 1979]); 4. Multi Hypothesis Tracker: MHT ([Reid, 1979]); 5. Probabilistic Multi-Hypothesis Tracker (PMHT) ([Streit and Luginbuhl, 1995]).

We here focus on the most simple of all of these: the **Assignment**. It consists in a hard one-to-one assignment: there is at most one measurement per scan per target (per sensor). A formal definition of the assignment and a presentation of the different algorithms to solve it will now be done.

Linear assignment. Let us consider the following problem: we want to associate a set of objects A to a set B. We can suppose here $|A| \leq |B|$. Let us suppose we have a "cost" function $f : (A, B) \mapsto \mathbb{R}$ that enables us to evaluate how an element from A matches one from B. The more the element a from A matches the one b from B, the smaller should be $d(a, b)$.

An **Assignment** is an association of each element of the smaller set with one element in the other set. However, each element of the bigger set can be assigned only at most once. In combinatorial mathematics terms, an assignment is thus a one-to-one mapping between the smaller set and a subset of the bigger set.

The **Linear assignment problem** aims at finding, between all possible assignments, the assignment that minimizes the aggregated cost of its pairs.

Example: For instance, we consider two sets of 2D colored points $Red, Blue \in \mathbb{R}^{\mathbb{N}}$. We suppose here $|Blue| \leq |Red|$ (if not the case, we can swap names). An assignment associates each $Blue$ point to one Red point, such as each Red point is used zero time or once. The cost function here can be for instance the Euclidean distance. The more a element from $Blue$ matches one from Rue, the smaller should be the value of the cost function between them.

$$f : \begin{cases} \mathbb{R}^2 \times \mathbb{R}^2 & \mapsto & \mathbb{R} \\ (x_1, y_1), (x_2, y_2) & \to & \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \end{cases}$$

Thus, the goal of the linear assignment problem is to find the assignment with minimal cost, that is the sum of the lengths of all segments between $Blue$ and Red. The **Cost matrix** is the matrix that stores all costs between $Blue$ and Red:

$$C \in \mathbb{R}^{|Blue| \times |Red|}, \quad C_i j = f(x_i, y_j)$$

Solving algorithms: the easiest way to solve the problem is to try and evaluate all possible assignments: it is called *brute force* solving. The possible assignments corresponds to the *k-partial permutations of n* where $k = |Blue|, n = |Red|$ (so $k \leq n$). The number of possible assignments correspond to

$${}_nP_k = n \cdot (n-1) \cdot (n-2) \cdots (n-k+1)$$

This quantity is sometimes known as the so-called **Pochhammer symbol** $(k)_n$.

However, this solution is clearly non-optimal, as it blindly tries all possible combinations. Its complexity is $O(n!)$ when $|A| = |B| = n$.

The most popular algorithm for solving linear assignments is called the *Hungarian algorithm*. It was developed and published by Harold Kuhn in 1955 [Kuhn, 2006]. It has a polynomial complexity. The original version was in $O(n^4)$, some optimizations allow a complexity in $O(n^3)$. However, its algorithmic relative simplicity and limited computational cost make it one of the most popular assignment algorithms.

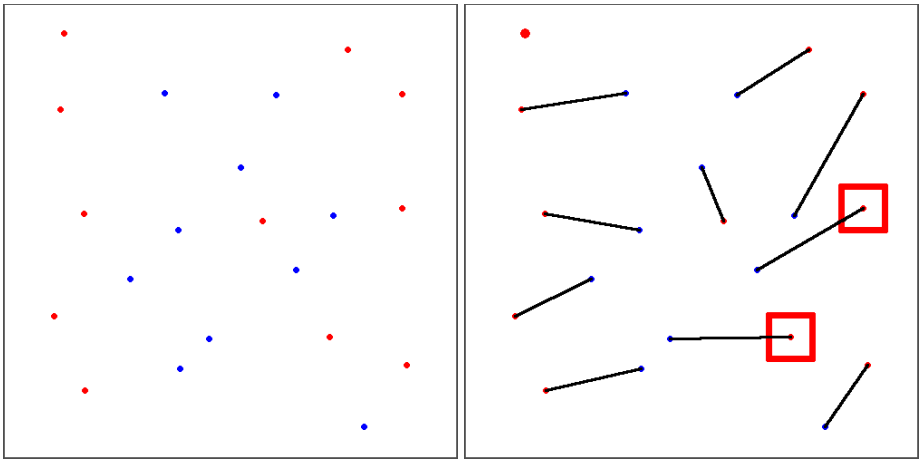


Figure 7.2: *An example of linear assignment problem for 2D points.*
(a): the two sets of points: the blue ones and the red ones
(b): the best assignment between both sets. Here the red set is bigger, and the point in the upper-left corner is unassigned. Note how the red points inside the rectangles are not associated with their closest red point: we minimize the total cost, not the cost for each point.

In the case of a sparse cost matrix, further optimizations can exploit such structure and reduce the complexity. For instance, Lemon's implementation [3] is in $O(nm \log n)$, where m is the number of edges.

The *auction algorithm*, proposed in 1979 and published in 1985 ([Bertsekas, 1985]) is a distributed iterative algorithm.

Finally, the *Jonker-Volgenant algorithm* was proposed in [Jonker and Volgenant, 1987], and is based on linear operations on the cost matrix to obtain the assignment with the lowest cost.

Fastest method A review is available in [Dell'Amico and Toth, 2000]. The previously proposed algorithms, and some others have been benchmarked on a set of problems. It is hard to determine the fast assignment algorithm. However, the authors notice the poor performance of the auction algorithm compared with the others, and the fairly good one of the Jonker-Volgenant algorithm.

[3] `https://bitbucket.org/alpar/lemon-bipartite`

7.2 Research contribution

In this part, we will present the different possibilities we explored to give spatial and temporal consistency to the people detections.

After a brief introduction that will present the benchmark of two linear assignment algorithms, in subsection 7.2.1, we will present two different ways of giving consistency to the different user detectors presented in the first part of this PhD: using the different people recognition trackers seen in Part II, or tracking the detection of one given user in the depth image.

The first method, because it is multimodal, distributed and apt to track several users at the same time, is the one who turned out to match better with our goals defined in the Introduction chapter. For this reason, it will be detailed in several subsections. We have previously seen two main categories of approaches that can be followed for multimodal fusion: particle filters and Kalman filters. In subsection 7.2.2, we will present a common interface, called `PeoplePoseList` Matcher, that we designed for all the people recognition algorithms to comply with. Then we will show how Unscented Kalman Filters, coupled with the `PeoplePoseList` Matcher (`PPLM`) mechanism, result in a robust user awareness system. In subsection 7.2.3, we will present how the different user recognition methods presented in Part II were integrated into our user awareness architecture by shaping them as `PPLM`s. The usefulness and accuracy of this user awareness system will be demonstrated thanks to careful benchmarking, in subsection 7.2.4.

The second method is less generic: it focuses on the accurate tracking of one given user using a chosen tracking method. It is based on the idea that once a user is detected, we can track him or her from one frame to another in the depth image. Indeed, the user will correspond to a blob of similar position and appearance in consecutive frames. Image processing techniques can be used for the tracking of the shape of this blob in the depth stream. This approach will be explained in subsection 7.2.5.

7.2.1 Preliminary: benchmarking of linear assignment algorithms

Multi-target Unscented Kalman Filter is one of the two widespread classes of algorithms for multimodal fusion, the other being particle filters. It has been presented in subsection 7.1.2

As seen in subsubsection 7.1.2.iv, page 148, multi-target Unscented Kalman Filter (UKF) uses assignment methods to match instantaneous detections to exiting tracks. The used algorithm is *linear assignment*. In this part, we benchmark this key part of the tracking pipeline to ensure solving the problem will not be a bottleneck.

The input of the linear assign is a cost function f, that can be also presented as a cost matrix C where the element of C at (i, j) represents the cost generated by matching the i-th element of A to the j-th element of B.

Out of all the different assignment algorithms that were presented in subsubsection 7.1.2.iv, we chose to implement two for their popularity and availability: the brute-force algorithm, trying

all possible combinations, and the linear assignment, presented by [Jonker and Volgenant, 1987]. Both were implemented in C++. The brute-force was written from scratch while the Jonker algorithm was based on the freely available original C implementation (`lap.c`).

We here set A and B of same sizes, and we iterate over $n = |A| = |B| \in \mathbb{N}$. The tests were run on a desktop PC with a full-power processor from 2008 (AMD Athlon 64 X2 Dual Core Processor 5200+). The results are visible in Figure 7.3.

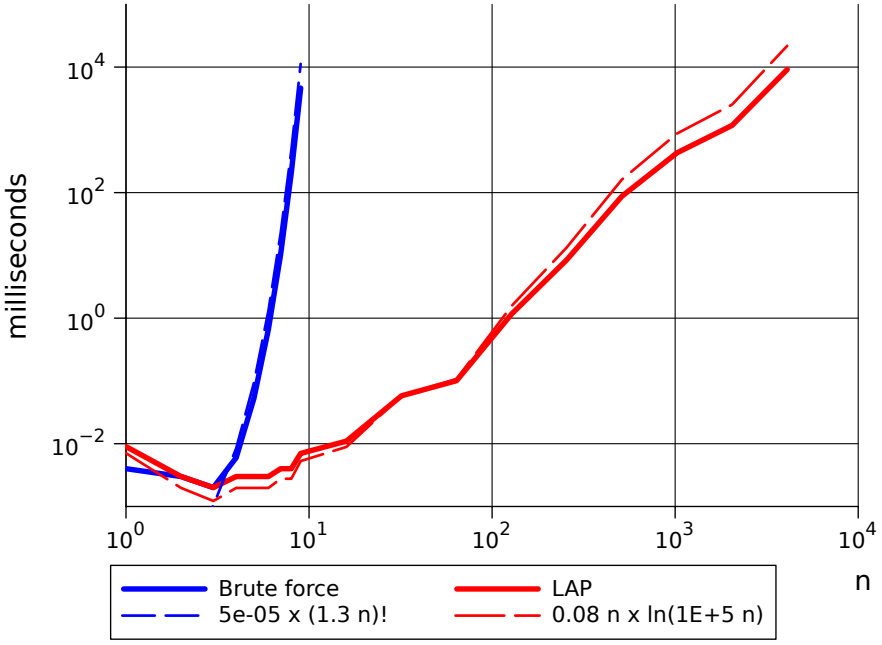


Figure 7.3: *Benchmark of both brute-force and Jonker-Volgenant assignments solvers.*

We can observe that brute-force is not an appropriate solution for real time matching: its complexity is, as expected, roughly equal to $n!$, and for 9 tracked users for instance, the matching takes approximately up to five seconds. As the matching needs to be made for each `PeoplePoseList` Publisher (`PPLP`) in each frame, the matching cannot require more than a few milliseconds, and then the use of the brute force solver must be discarded.

On the other hand, the Jonker-Volgenant solver performs much better: its complexity is roughly equal to $n \ln n$, and the matching is done faster that in ten milliseconds for any number of tracked users below 256. As it is not probable that we have as many users, the Jonker-Volgenant solver meets our requirements and will be used for solving the assignment problem in the multi-target tracking.

7.2.2 A common interface for PeoplePoseList (PPL) matchers: the PeoplePoseList Matcher (PPLM)

In this part, we will present how multimodal fusion algorithms can be used for giving spatial and temporal consistency to the users detection and recognition algorithms presented in the previous parts.

Through this section and the following ones, we will call $\mathcal{T}$ the set of tracks and $n_{\mathcal{T}}$ the number of tracks:

$$\mathcal{T} = \{p_i, 0 \leq i \leq n_{\mathcal{T}}\} \text{ where } \forall i \in [0, n_{\mathcal{T}}[,\ p_i \text{ is a PeoplePose (PP)}$$

Similarly, we also consider a given detected PPL $\mathcal{P} = \{p_i, 0 \leq i \leq n_{\mathcal{P}}\}$, containing $n_{\mathcal{P}}$ PPs. Our goal is to estimate the cost matrix C between $\mathcal{T}$ and $\mathcal{P}$, defined by the similarity between each PP of $\mathcal{P}$ and each track of $\mathcal{T}$. The more a detection PP from $\mathcal{P}$ matches a track PP from $\mathcal{T}$, the smaller should be the value of the cost function between them.

The different PeoplePoseList Publishers (PPLPs) give us estimations of the user position at each frame. On the other hand, the multi-target Unscented Kalman Filter (UKF) builds different *tracks* that correspond to highly probable user positions and states (identities, etc.). It is a good multimodal fusion algorithm, that matches a detected PPL against its set of tracks. *Linear assignment*, that solves the assignment between PPs and tracks, first requires an estimate of the likeliness of each detected PP against each track. This likeliness is presented as the cost matrix C, having $n_{\mathcal{T}}$ rows and $n_{\mathcal{P}}$ columns. A linear assignment then finds the assignment between the detected set $\mathcal{P}$ and the tracks set $\mathcal{T}$.

Definition of the PeoplePoseList Matcher (PPLM) An algorithm that computes a cost matrix between $\mathcal{T}$ and $\mathcal{P}$ is called a PeoplePoseList **Matcher** (PPLM). The idea for integrating the different user recognition algorithms is the following: this cost matrix C can actually be the combination of different user matching algorithms running in parallel, each of these providing its own cost matrix. The final cost matrix C is then the sum of all these costs matrices. Similarly to each cost matrix, The more a detection PP from $\mathcal{P}$ matches a track PP from $\mathcal{T}$, the smaller should be the corresponding element in C. From now on, both terms *"recognition"* and *"matching"* are used indifferently: a matching algorithm is the same as a recognition algorithm.

The following subsection will define the design and implementation of this idea, then the process of converting the previously presented user recognition algorithms into PPLMs will be further explained.

7.2.2.i Implementation of the multi-target Kalman filtering with PeoplePoseList Matcher (PPLM)

We have defined a common structure for matching algorithm for PeoplePoseList Matchers (PPLMs): they generate cost matrices.For a given detection PPL, the cost matrix describes how

each detected PP is similar to the different reference tracks (also structured as PPs). This design is modular, distributed, flexible, among other advantages.

Several matching algorithms have been shaped as PPLMs, for instance the height-based or the PersonHistogramSet (PHS)-based algorithm. More will be said in the next section.

However, we still need the final brick of the architecture: that brick that uses these PPLMs on the different outputs of the PPLPs to create tracks using multimodal fusion algorithms. This will be presented in this part: we will present how we implemented multi-user tracking using a combination of UKFs and PPLM.

Two classes of multimodal fusion algorithms have been presented previously: Kalman filtering and particle filters. Because of its availability, we first chose to use Kalman filtering for studying the integration of PPLPs and PPLMs. Indeed, several libraries propose an easy integration of Unscented Kalman Filters, among others LibDAI ([4], [Mooij, 2010]). An extension of our work using particle filters would be an interesting work.

From now on, the processing block in charge of retrieving all cost matrices and performing the multimodal fusion, and that will be structured as a Robot Operating System (ROS) node, will be referred as the *fusion node*. The idea of the multimodal fusion is the following. The fusion node has in memory a set of tracks corresponding to the tracked users. Each of these tracks is structured as a PPs. This node subscribes to a range of topics emitted by PPLPs. Then, upon reception of each PPL, the node will call the different matching services offered by the set of PPLMs. For each detected PP, the corresponding track will be updated. Finally, the updated set of tracks, that is also shaped as a PPL, is emitted by the fusion node. This information can be used by higher level applications that will have at their disposition all the information needed about the local users and their mapping.

Simple example: single user tracking An example is given in Figure 7.4. We here consider there is only one user to detect and track (simple case). There are two PPLPs: one based on the color images, the face detection based-PPLP, and one based on scans of the laser range finder, the leg detector PPLP. The fusion node contains only one UKF corresponding to the tracked user. The different steps upon reception of a new PPL emitted by any of the detectors are the following:

1. In this received PPL, not all PPs are necessarily used for the matching by the fusion node. Indeed, let us imagine we receive a PPL with only one PP inside, but that correspond to a false positive, several meters away from the track. We do not want to update the track with this erroneous PP. In the so-called **Gating** process, three-dimensional (3D) detections that are too far from the track position are not used to update the track, but instead are kept in a buffer. The maximum Euclidean distance is a parameter that can be adjusted. Similarly, if there are more detected PPs than tracks, they are also put in that buffer. When a critical number of unassociated measures accumulates at a given spatial position in a window of

[4] http://cs.ru.nl/~jorism/libDAI/

time, a new track is created. [5]

2. In the fusion node, once the gating is done, the set of PPLMs is called with, on the one hand, the PPL made of all the PPs that pass the gating phase, on the other hand, the track PPs. The fusion node requests each of them to compute its own cost matrix. The final cost matrix corresponds to the sum of all PPLMs cost matrices successfully computed [6]. The detection PP most likely with the track is identified thanks to the generated cost matrix. As there is only one track, the cost matrix will have only one row.

3. The unique track maintained of the fusion node, containing its own UKF, is updated with the 3D position of the detection PP.

4. Finally, the fusion node publishes a PPL corresponding to the tracks in memory, and hence only containing one PP centered on the user and containing the information gathered about her: along with her 3D pose, in this case, the image of her face, that can be used by other nodes.

Generalization to multiple user tracking This first case with one user is fictional: we cannot guarantee the number of users beforehand in most situations. A more realistic setting is presented in Figure 7.5. In this case, there are two physical users visible by the robot. The fusion node subscribes to the same two PPLPs (face detection-based and leg detection-based PPLMs). Upon reception of each new PPL, the same gating process is first applied. However, then, we need to determine which track corresponds to which detected PP. This is where the different PPLMs are useful: thanks to the global cost matrix, the more similar a given track PP and a detection PP, the smaller the corresponding element of the cost matrix. The linear assignment corresponding to this cost matrix is computed, which tells us which track corresponds to each detected PP. As such, the tracks are updated, and published as a PPL to the higher-level applications in the robot.

7.2.2.ii Design of the PeoplePoseList Matcher (PPLM) in ROS terms

We want the different user recognition algorithms to compute their cost matrices in parallel while ensuring that, for a given $\mathcal{P}$, we obtain all cost matrices. Two different programming solutions are possible for obtaining such an architecture.

- All the detectors can be explicitly included within the fusion ROS node thread. During the assignment process, the fusion node will then explicitly call all the embedded PPLMs.

[5] More advanced techniques than gating exist for this association, see for instance Probabilistic Data Association Filter (PDAF) or Joint Probabilistic Data Association Filter (JPDAF). However, gating is both simple to implement and efficient.

[6] Indeed, some PPLMs might fail in their computation for a wide range of reasons and in that case, they are not taken into account. For instance, a height detector can fail because the depth image is corrupted.

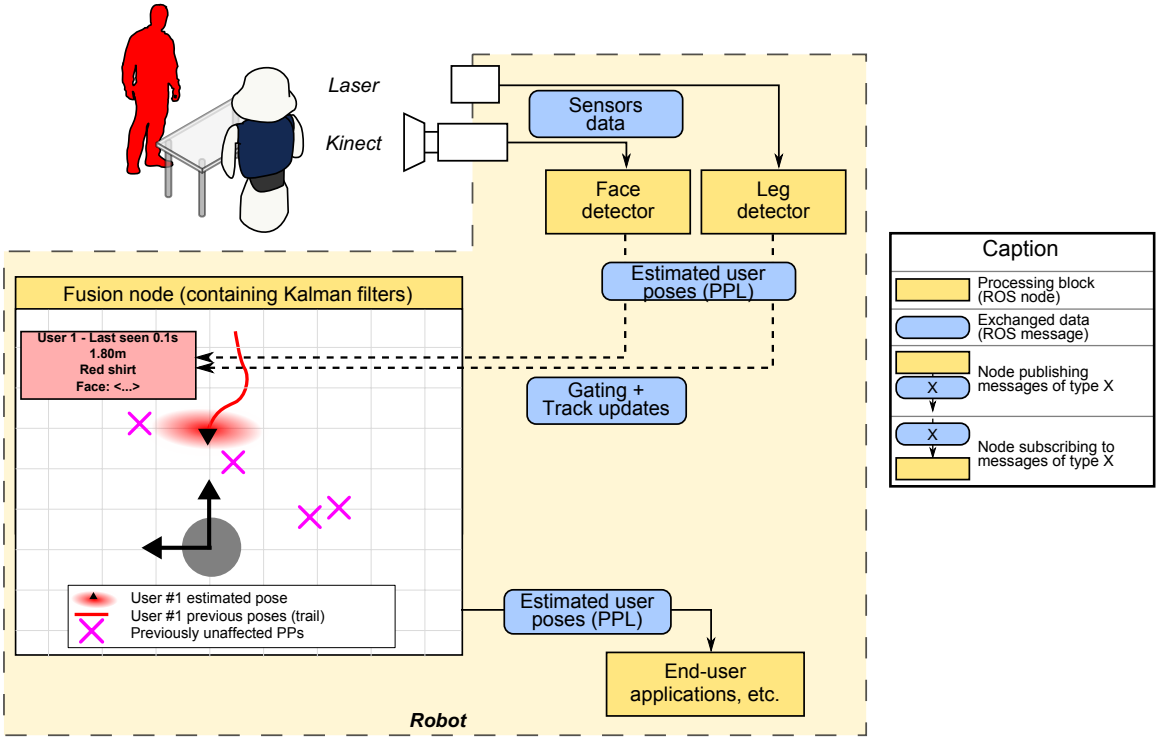


Figure 7.4: *Multimodal fusion based on Unscented Kalman Filters: fictional case of single-user detection.*

- The other solution is to structure every `PPLM` as a standalone ROS node. To allow the transfer of the data, each of this node provides a ROS *service* of identical type, called `MatchPPL`, but with different names. Note that ROS services were introduced in the Introduction chapter. They can be seen as inter-nodes remote function calling. The fusion node then calls the different services that it wants to use.

The first solution offers the advantage of being the fastest possible solution, as there is no delay generated by the transmission of data from one thread to another. However, it lacks of modularity: all the `PPLM`s will be running in the same computer, so the fusion node will be as slow as the sum of all `PPLM`s. Furthermore, all `PPLM`s need to be written in the same programming language, as they belong to the same node. On top of that, reconfiguring the set of `PPLM`s according to the needs of the robot is challenging: if we want to shut down a given one, again we need to explicitly change the fusion node. Finally, to add a new `PPLM`, the fusion node itself needs to be modified to explicitly call this new module.

On the other hand, the second solution using ROS services offers a great modularity: the different nodes can be distributed among different machines thanks to the ROS communication layer. Say for instance that a given `PPLM` is computationally costly, in that case, we can run it on a remote, more powerful computer, and send back the generated cost matrix via the network.

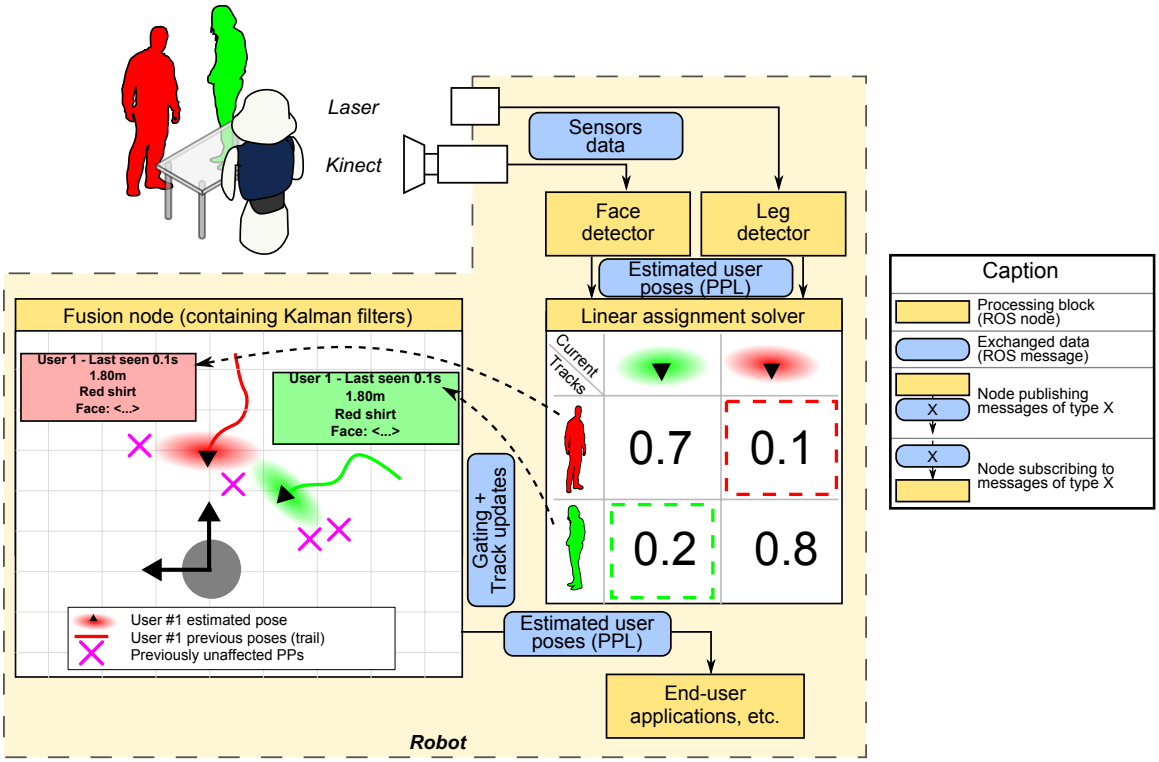


Figure 7.5: *Multimodal fusion based on Unscented Kalman Filters: multi-user detection.*

Furthermore, the CPU computations are only triggered by the service calls, ensuring that we can change dynamically the set of PPLMs used: all detectors can be instantiated in the robot architecture, but only the ones we want to use are specified to the fusion node via a parameter. For instance, let us say a given matcher supplies a MatchPPL service called /match1 and another a MatchPPLservice called called /match2. If we specify to the fusion node to use only /match2, only the MatchPPL service provided by the second node will be called, even if both PPLMs are instantiated and running. Finally, adding a new PPLM is easy and fast: the new node needs to supply its own MatchPPL service. It can be implemented in any of the programming languages offered by the ROS compilation mechanisms.

For all these reasons, the second solution was chosen: a PPLM is structured as a standalone ROS node, providing a MatchPPL service. The list of service names to use is given to the fusion node thanks to a ROS parameter. The data flow is illustrated in Figure 7.6.

Advantages The modularity of this system presented in the previous subsection was presented in the previous paragraph. They are very similar to the advantages we presented for the PPLP mechanism, in subsection 3.2.2, page 41. These advantages can be thus resumed.

- **Programming language abstraction.** ROS generates generated headers for the MatchPPL

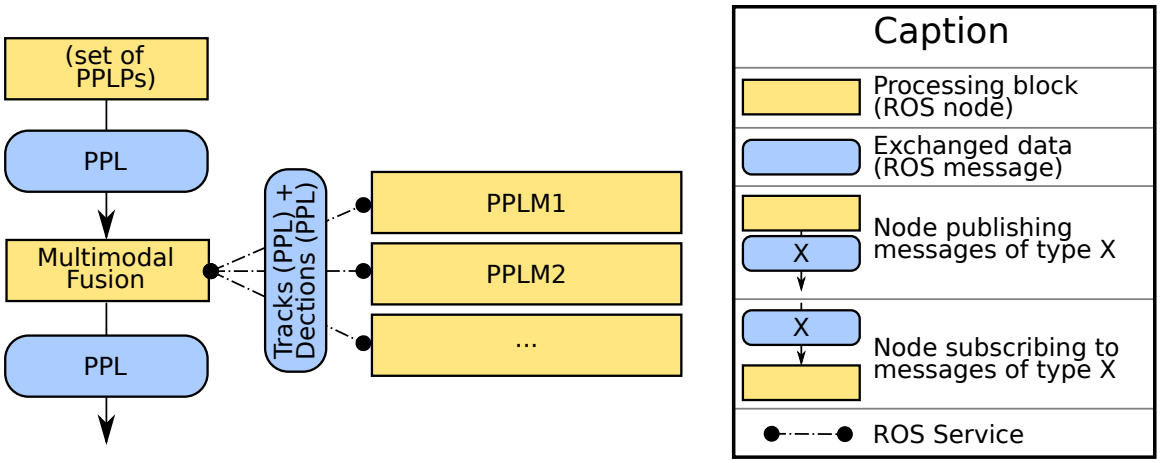


Figure 7.6: *Diagram of the dataflow between the fusion node and the different* `PeoplePoseList` *Matchers.*

service in C++, Python and Perl.

- **Workload distribution.** Thanks to ROS communication layer, the different `PPLM`s can be distributed among the different machines.

- **Integration of new algorithms.** If we want to add a new `PPLM`, the only requirement is that it supplies a `MatchPPL` service.

- **Debugging and benchmarking made easier.** We can conceive a set of tools for debugging and benchmarking all `PPLM`s, as they provide a common interface. The debugging tools can include simple cases, such as matching a given `PPL` with a copy of itself.

We will now present how the different user recognition methods presented in Part II were integrated into our user awareness architecture by shaping them as `PPLM`s.

7.2.3 Integration of recognition algorithms as `PeoplePoseList` Matchers

The most relevant user recognition algorithms presented in Part II were shaped as `PeoplePoseList` Matchers (`PPLM`s). We chose the algorithms that have proven to be robust and easily adaptable into `PPLM`s. Namely, the following `PPLM`s were created: 1. a simple `PPLM` based on Euclidean distance matching; 2. the face recognition-based `PPLM`; 3. the height-based `PPLM`; 4. the NiTE-based `PPLM`; 5. the PersonHistogramSet (PHS)-based `PPLM`. Each of these `PPLM`s will now be explained in further details. Note that the algorithms that allow the a-priori estimation of the gender, i.e. the gender-from-face and the breast detector, were not integrated in the actual version. An extension of our work with these algorithms would be an interesting work.

Distance `PeoplePoseList` **Matcher** The simplest method to estimate the likeliness of a track against a detected `PeoplePose` (`PP`) is to compare their three-dimensional (3D) position. In other words, the closer a track and a detection, the more likely they correspond to the same person.

We then have the explicit formula:

$$\forall i, j \in [1, n_{\mathcal{P}}] \times [1, n_{\mathcal{T}}],\ C(i,j) = \|\mathcal{P}_i, \mathcal{T}_j\|$$

This likeliness estimation needs to choose a distance function. The most common norms have been seen in section 5.1.3.i, page 97. We used the Euclidean L_2 norm, as it corresponds more accurately to the standard definition of distance between 3D positions:

$$\forall A, B \in \mathbb{R}^3,\ \|A, B\|_{L_2} = \sqrt{(A.x - B.x)^2 + (A.y - B.y)^2 + (A.z - B.z)^2}$$

Note on optimization: the computation of the L_2 norm can be costly as it involves two multiplications and a square root computation. For speedup reasons, the Manhattan L_1 distance could be used:

$$\forall A, B \in \mathbb{R}^3,\ \|A, B\|_{L_1} = \mid A.x - B.x \mid + \mid A.y - B.y \mid + \mid A.z - B.z \mid$$

Face recognition `PeoplePoseList` **Matcher** The height is a meaningful physical trait for recognizing one user from another. On top of that, other traits can be used. The visual appearance of the face is key information, that the humans use extensively to discriminate between people, as presented in the Introduction chapter. For this reason, the face recognition algorithm presented in subsection 5.2.2, page 104 was integrated.

It re-uses the results of the face detection `PeoplePoseList` Publisher (`PPLP`) presented in subsection 3.2.3, page 47. In each track is stored the image of the user face as soon as there is one available (for instance, when the track is initialized by a `PP` containing a user face). The `PeoplePoseList` (`PPL`) message generated by the face detection node contains for each user, along with the color and depth images of the user, the coordinates of a rectangle delimiting the face in these images. These coordinates are stored as in the `attributes` field of the `PP`. As such, we do not need to run again a face detector on the color image of the user to find where her face is. The user face images available in the `PPL` are then passed to the face recognizer presented in subsection 5.1.1, page 91.

We remind that the most similar two `PP`s, the smaller should be their cost. The cost matrix is initialized to a constant value: $\forall i, j \in [1, n_{\mathcal{P}}] \times [1, n_{\mathcal{T}}],\ C(i,j) = 1$. Then, if the i-th user face in the detection `PPL` is set, the face recognizer determines the most similar reference `PP`, say the j-th track. The corresponding cell in the cost matrix is set to zero: $C(i,j) = 0$. As such, the face recognizer suggests a match between detected `PP` i and track j in a "soft way": there is not always a suggested matching for all detections. In the case of a `PPLP` that does not find the faces, for instance the HOG detector seen in subsection 3.2.4, page 50, the cost matrix is full of ones and then does not weight in the final assignment.

Height `PeoplePoseList` **Matcher** The previously presented PHS-based `PPLM` was based on the use of soft biometrics for user identification, more especially adhered human characteristics: the color of the clothes is indeed a feature that has a short life scope, as the user can change them from one frame to another.

On the other hand, the height-based `PPLM`, presented in subsection 5.2.3, page 108, is based on physical traits: the height of the user is a permanent feature. The idea is the following: users can be somewhat recognized thanks to their height. Even though it is not a reliable feature for people of an average height (average height is 1.78 m for adult Spanish men, and 1.66 m for adult Spanish women according to [Garcia and Quintana-Domeque, 2007]), it is a powerful feature when it comes to discriminating between adults and children, or tall users and smaller ones.

The final cost is obtained by mapping the height difference in the $[0, 1[$ interval:

$$\forall i, j \in [1, n_{\mathcal{P}}] \times [1, n_{\mathcal{T}}],\ C(i, j) = 1 - \exp\left(\alpha \times |height_i - height_j|\right)$$

where $\alpha > 0$ is a scaling parameter. In our implementation, $\alpha = 3$.

NiTE `PeoplePoseList` **Matcher** the **NiTE-based** `PPLM`**:** The raw output of the NiTE algorithm was presented in subsection 3.1.3, page 36 and is shaped as a user multimap. The cost of matching a given detected `PP` with the set of tracks is defined as follows: this cost is equal to zero if both NiTE user identifiers are equal, equal to one otherwise. In other words, the cost matrix of this `PPLP` is mostly set to one, with some zero values where NiTE names correspond:

$$\forall i, j \in [1, n_{\mathcal{P}}] \times [1, n_{\mathcal{T}}],\ C(i, j) = \begin{cases} 0 & \text{if } NiTE\_id_i = NiTE\_id_j \\ 1 & \text{if } NiTE\_id_i \neq NiTE\_id_j \end{cases}$$

PersonHistogramSet `PeoplePoseList` **Matcher** The distance matcher only uses the 3D spatial information of the user. However, her visual appearance contains meaningful cues that can be used for matching. For instance, if a user has a red shirt and blue trousers in one frame, her likeliness to a `PP` with a similar appearance in another frame is higher than if this `PP` has a green shirt. In subsection 5.2.5, page 122 we presented a user recognition algorithm based on the visual appearance of the user, called *PersonHistogramSet (PHS)*.

We integrated this algorithm into a `PPLM`: on each matching request, the PHS of each detected user in $\mathcal{P}$ is computed. On the other hand, for each track of the reference set $\mathcal{T}$, the PHS needs to be computed only when the user images associated with this track is changed. The final cost between the track and the `PP` is equal to the result of the histogram comparing function presented in subsubsection 5.2.5.iii, page 124, already bounded between 0 (perfect match) and 1 (absolute mismatch):

$$\forall i, j \in [1, n_{\mathcal{P}}] \times [1, n_{\mathcal{T}}],\ C(i, j) = d_{PHS}(hist_i, hist_j)$$

Summary of the whole structure Several matching algorithms have been shaped as PPLMs: an distance PPLM based on distance matching; a face recognition-based PPLM; a height-based PPLM that uses the estimated physical height of the users to match them; a NiTE-based PPLM re-using the user identifiers extracted by the NiTE algorithm; and finally a PHS-based PPLM using the color appearance of the users.

The structure of a full user awareness sporting a multimodal fusion using all implemented PPLPs and PPLMs is presented in Figure 7.7. Note that this is not a convenient structure: using all the algorithms at the same time corresponds to a heavy computational overhead that could be lightened easily selecting the most appropriate algorithms according to the intrinsic and extrinsic configuration of the robot. For instance, if it has no laser range finder, the leg PPLP can be removed. If the users will not come close, the face recognition PPLM is not needed.

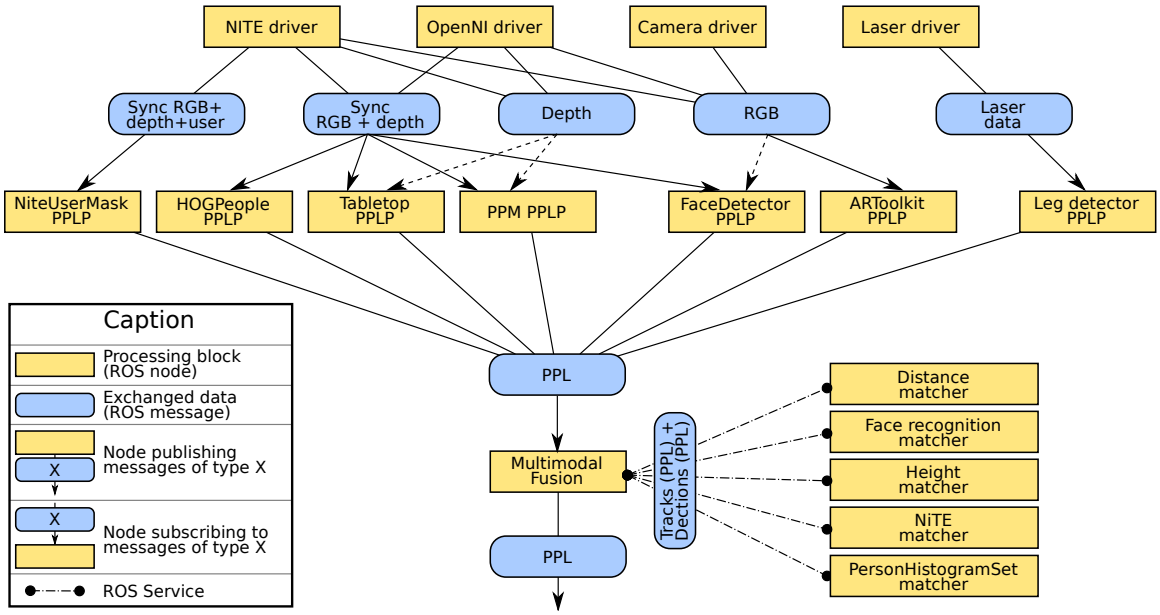


Figure 7.7: *Multimodal fusion using all implemented PPLPs and PPLMs.*

7.2.4 Benchmarking and limitations of the Unscented Kalman Filter (UKF) and different combinations of PeoplePoseList Matchers (PPLMs)

To measure the accuracy of the different PPLMs, it is better to use academic benchmarks. It does not make sense to use the DGait dataset we used for benchmarking the different PeoplePoseList Publishers (PPLPs): this database is made of 55 videos of one user at a time, walking on a stage. As such, the matching accuracy would be artificially high: the UKF having only one track, matching one user against one track always succeeds.

We then used two different datasets: the Kinect Tracking Precision (KTP) dataset ([Munaro et al., 2012, Munaro and Menegatti, 2014]), published by researchers in robotics from an Italian

university, and a homemade dataset using real data acquired from our robots, that we named RoboticsLab People Dataset (RLPD).

7.2.4.i Benchmarking using the KTP dataset

Like the DGait database, it is made of a collection of about 2200 frames of synchronized depth and Red Green Blue (RGB) data, along with annotated 2D coordinates of the users in the color frame. On top of that, the authors used a system of infra-red markers placed on top of the head of the users, so that the exact three-dimensional (3D) position of the head of each user is known with accuracy. The video sequence is made of several users walking randomly in a room. The number of users varies between one and four at the same time, with challenging trajectories and a high number of crossings and occlusions. This generates a perfect benchmark for our system. Some samples are visible in Figure 7.8.

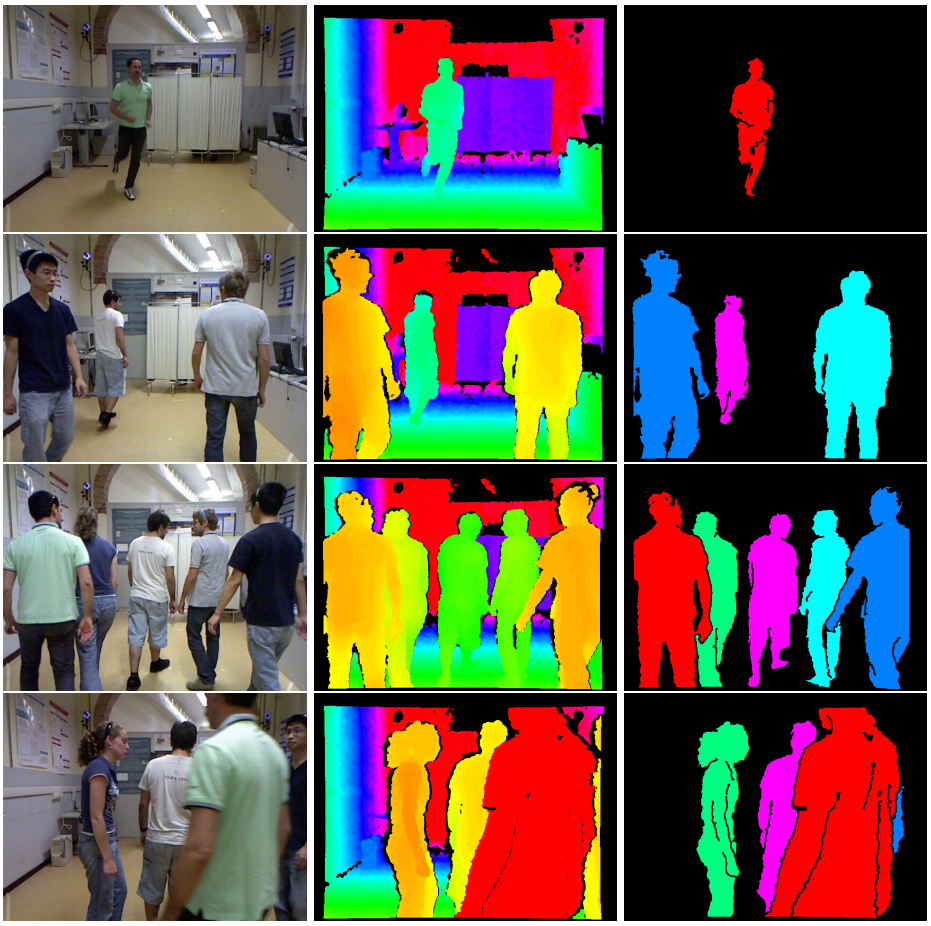


Figure 7.8: *Some samples of the KTP dataset. Note the challenging conditions: fast motion of the users, partial or complete occlusions, complex trajectories, crowded environment, etc.*

Different UKF nodes can be running in parallel, as each of these publishes their own multimodal fusion on a different Robot Operating System (ROS) topic. This gives the possibility of benchmarking several configurations of PPLMs at the same time. A total of 4 configurations were used: 1. using the distance PPLM only; 2. using the PersonHistogramSet (PHS)-based PPLM only; 3. using the height-based PPLM only; 4. using both PHS and distance-based matching.

All configurations of PPLMs cannot run at the same rate: for instance, the distance PPLM does not have heavy computation needs. On the other hand, the PHS matcher, for instance, requires, for each frame, the computation of the color histograms of each user. The comparative performance of each configuration is detailed in Table 7.1. Note that the number of processed frames helps us seeing the computational need of the different PPLMs, but the important metric is the number of swaps: all configurations run online on the same video sequence of the database, and hence give the same output as they would if the sequence was happening in the real world. This is why we chose to play the sequences at normal speed and not slowed down: the computation speed of a PPLM can be a constraint on its output in the real life.

PPLM configuration	Distance	PHS	Height	Distance+PHS
Duration (seconds)	71			
Total frames	2137			
Frames with several PPs	1277			
... of which were processed by the PPLM	100%	45%	93%	55%
ID swaps	18	35	189	16

Table 7.1: *Benchmark results for different configurations of PPLMs on the Kinect Tracking Precision (KTP) dataset.*

The distance PPLM (first column) is a naive matching algorithm: it matches a detection to its closest track, without considering any information about the users appearance. However, the computation associated being very light, it processes the frames at a very high speed. In crowded environments though, it can erroneously swap IDs. Note that it is the best single PPLM: its high frequency of matching ensure a correct labeling in most cases. Furthermore, it is more sophisticated than a simple ”nearest neighbor” approach: the linear assignment ensures the minimal matching cost, and no repeated ID.

The PHS-based PPLM (second column) obtains a significantly good performance: its number of swaps is only twice as important as the distance matcher, even though it does use any spatial information about the users: their sole color appearance gives some meaningful hints about who is who from one frame to another. This confirms the usefulness of the use of color information for user matching, and suggests that the PHS can be used to help a distance matching, as we did for the last column.

The height-based PPLM (third column) performs poorly, as expected: matching users by their height only is error prone. In the case of the KTP dataset, all the users have very similar height (about 170 centimeters), which makes the matching unreliable. Note that the real domain of use of the height-based PPLM is different: it is helpful when the different users have a significant size

difference.

Finally, the most interesting column concerns the combination of the PHS-based `PPLM` and the distance matching (fourth column). The computational complexity of this multimodal system is of course higher that each of the two `PPLM`s taken separately, which explains the framerate being lower than the distance matcher. However, the number of swaps (16) is inferior to the best single detector (18), obtained with the distance matching. This is the important conclusion of this part: *using several parallel matching algorithms improves the accuracy of the matching*. Indeed, in most cases, the distance is discriminative enough to match correctly the users. In ambiguous cases though, the distances are roughly equal, and the PHS comparison will help solving the ambiguity.

In summary, in this part we presented a common interface for the different tracking algorithms, called `PeoplePoseList` Matcher (`PPLM`). This interface is then used to link the algorithms with a data filter based on called Unscented Kalman Filter (UKF). This enables a modular architecture where the configuration of matchers can be redesigned easily according to the physical configuration of the robot and the hardware limitations. Several configurations of matchers where benchmarked on a challenging dataset called KTP. Quite surprisingly the most accurate matcher is the one based on the distance distance. It can be explained by its very high frame rate compared with more sophisticated methods, such as the one based on the visual appearance of the clothes of the user or the one based on its height. The key result is that the combination of various measures obtains better results than the one of these taken alone. This means our architecture makes sense and is useful for accurately tracking users around the robot. We will now deepen these conclusions with a more extensive benchmark using homebrew data.

7.2.4.ii Benchmarking using a homemade dataset: the RoboticsLab People Dataset (RLPD)

Academic datasets, like KTP, ensure a faithful measurement of the performance of the user awareness system. They make easier the comparison with other similar systems, and ensure that the measurements can be repeated in the same context.

However, our system is aimed at specific robots, which are the ones of the RoboticsLab, even though it was designed as generic as possible. For this reason, we decided to acquire real data from one of our robots, and with users that fit best the target audience: people from Spain, with a variety of genders and shapes. In addition, the actors do not wander randomly on the stage, as they did in the KTP dataset. We designed scenarios that mimic a realistic Human-Robot Interaction (HRI) scenario in which one or several users interact naturally with the robot: addressing the robot, using gestures, respecting the proxemics distance ([Mumm and Mutlu, 2011]) , etc. This dataset is called RoboticsLab People Dataset (RLPD) and is summarized in Figure 7.9.

Figure 7.9: *Synthesis of the RoboticsLab People Dataset (RLPD) building.*

Dataset summary The scenario presented involves three users interacting with a robot that integrate a Kinect camera. They move on the stage accordingly to a script that was previously defined and visible on the figure. Their motion is challenging: they get in and out of the room, there are occlusions and partial views.

The dataset is meaningful if and only if the real positions of the users is known. We first thought of using markers, such as ARToolkit markers, as presented in subsubsection 4.1.1.i, page 68. However, the imperfect detections could not guarantee an accurate ground truth concerning the users positions in each frame. For this reason, in each of the 600+ frames, the ground truth user positions have been manually labeled. This dataset is licensed under the terms of the GNU General Public License Version 2 as published by the Free Software Foundation, and freely available to download on the author's website, along with images and videos [7].

Input frames Each frame is labeled thanks to its timestamp. The timestamp is expressed in milliseconds since the beginning of the recording, using six digits with leading zeros. For instance, frame 065514 has been recorded one minute and five seconds after the beginning of the recording. In total, we have 647 frames for 133 seconds (2 minutes 15 seconds roughly), which is in average 5 frames per second.

Acquired data For each frame, we have 4 images and 1 data file:

- the RGB image (`"XXXXX_rgb.png"`, lossy JPG compression, quality:85).
- the depth image (`"XXXXX_depth.png"` and `"XXXXX_depth_params.yaml"`, lossy affine depth-as-PNG compression).
- the user map obtained as output of the Kinect API, called NiTE. This image stream is synchronized with the RGB and depth streams of the Kinect, and indicates, for each pixel of each frame, if this pixel belongs to a detected user using the NiTE algorithm presented in subsection 3.1.3, page 36. (`"XXXXX_user_mask_illus.png"`, lossless PNG compression)
- the hand-labeled ground truth user map. This image stream has the same purpose as the NiTE user maps, but it has been manually annotated so that it contains the exact ground truth concerning the position of each user in each depth image.
  (`"XXXXX_ground_truth_user.png"`, lossless PNG compression)

On top of that, the Kinect camera info is also stored: it is made of the camera intrinsic parameters, and enable us to convert 2D pixels into 3D points. The calibrations for both RGB and depth (IR) cameras are available.

[7] `https://sites.google.com/site/rameyarnaud/research/phd/robotics-lab-people-database`

Dataset annotation The ground truth user positions have been labeled manually in each frame, using both RGB and depth contents. This has been eased by a series of tools that we developed. Like the database, these tools are licensed under the terms of the GNU General Public License Version 2 as published by the Free Software Foundation, and freely available to download on the author's website [8].

A first tool, called `contour_image_annotator`, is made for annotating binary contour images. It is particularly handy for annotating images where the contour image has already been generated, for instance with an edge detector. It generates annotated images with a given prefix, by default `<input image>_user_illus.png`.

Another tool, called `user_image_annotator`, is an extension of `contour_image_annotator` for depth images: it first generates the contour image from the depth image using edge detection (namely, Canny filters). For instance, this tool can ease the annotation of ground truth user positions in each frame of a data recording. It can also use the RGB image: it displays it in the background, which can help annotating the user image. A sample of the GUI is visible in Figure 7.10.

Dataset analysis The dataset was recorded on July 2014. It is made of 647 frames, each of them being made of four images, i.e. about 2600 images. The dataset is roughly 65 megabytes big. These images can be easily imported into any programming language, such as C++ or Matlab. Some samples are visible in Figure 7.11.

`PPLP`**s benchmarking** Similarly to what was done in chapter 3, we benchmarked the different user detection algorithms on this new dataset. These algorithms were all wrapped with a common interface, `PeoplePoseList` Publisher.

The performance of the different `PPLP`s on the RLPD dataset, presented in the previous subsections, is gathered in a chart visible in Figure 7.12.

The analysis of this chart must be compared with Figure 3.17, page 63, which was obtained using the DGait dataset. As explained before, the RLPD dataset offers a high level of complexity: the occlusions are frequent, the users are sometimes partially shown in the images, and in general, they are somewhat far away from the camera.

The NiTE `PPLP` has a high accuracy and hit rate (both above 90%), which confirm the good performance of the algorithm that was already seen with DGait (both metrics above 99%). Both these benchmarks confirm that the NiTE-based `PPLP` is a useful method for detecting users in a social robot, even in challenging conditions.

The tabletop `PPLP` had a very good performance on the DGait dataset, as it had over 99%. However, on this benchmark, its performance is lower: it has a detection rate around 40%. Indeed, as it can be seen in the sample images, the ground is hardly visible in the images

[8] `https://sites.google.com/site/rameyarnaud/research/phd/user-image-annotator`

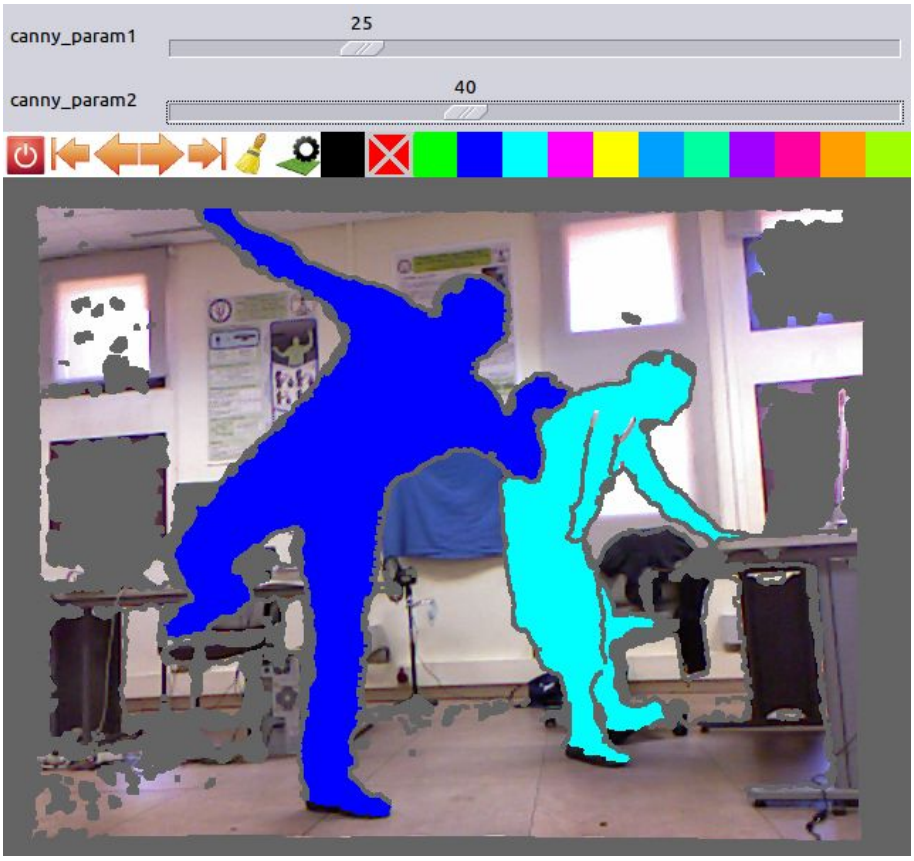


Figure 7.10: *A sample view of the Graphical User Interface we developed for annotating images, called* `user_image_annotator`.

acquired by the camera, which generates a poor estimation of the ground plane. We can then conclude that the tabletop PPLP is not appropriate for the robot spatial configuration used in the RLPD dataset.

In a way similar to the tabletop PPLP, the Polar-Perspective Map (PPM) PPLP performs much worse on our homemade dataset than on the DGait dataset: its accuracy lowers from 66% to 16%. The reasons are very similar: the limited visibility of the ground provokes a poor estimation of the transform between the camera space and the perspective map, which creates an incorrect projection into the latter.

On the other hand, the face detection PPLP has low accuracy and recall, but similar to DGait (around 25%): like the other database, the users are most of the time several meters away from the robot and often turn their back to the camera, which are challenging conditions for face detection.

Finally, the Histogram of Oriented Gradients (HOG) detector is maybe the most interesting: its detection and recall rates, which were above 70%, fall drastically to under 10%. As it can be

Figure 7.11: *Some samples of the RLPD dataset. From left to right: column 1: the RGB image, column 2: the depth image; column 3: the manually labeled user map; column 4: the Kinect API NiTE user map. The dataset gathers some challenging features: partial (second row) or complete occlusions (third row), user not fully visible (fourth row). Note how the manual color indexing of the users is consistent (third column): the same user always corresponds to the same color. On the other hand, the NiTE algorithm performs swaps and create new users, or even merges users (fifth row).*

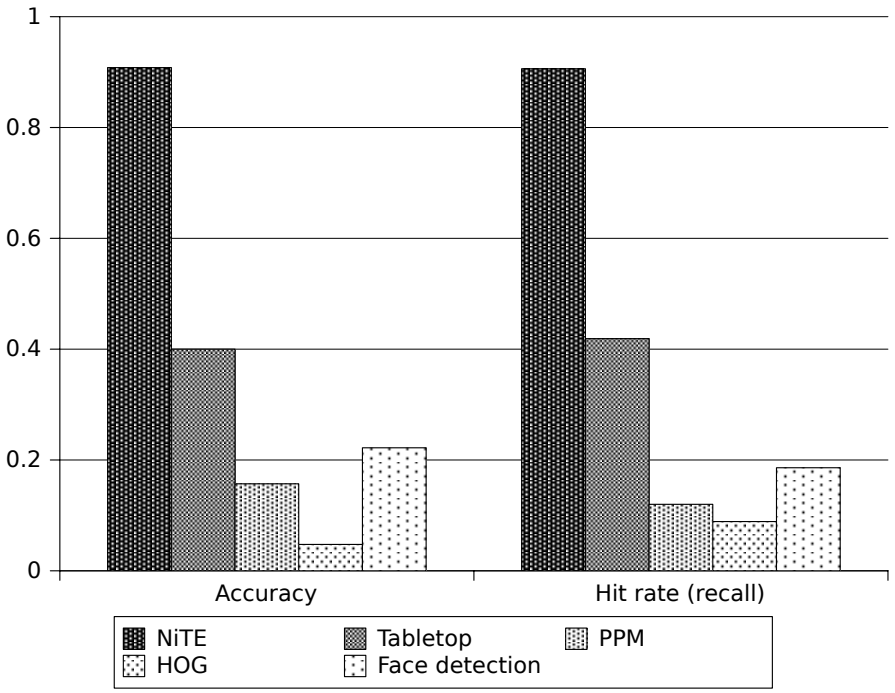


Figure 7.12: *Comparative performance of the different* `PeoplePoseList` *Publishers.*

seen in the sample images, the users are never fully seen, and especially their legs are most of the time out of the picture frame: the HOG detector being trained for detecting fully visible pedestrians, it performs very poorly on the RLPD dataset.

In conclusion concerning the PPLP benchmarking, we find similar conclusions to what was seen in chapter 3, page 31: each of the different PPLPs has both strengths and limitations, and the experimental setting will help choosing a configuration or another.

PPLMs benchmarking The different algorithms used to perform people recognition and presented in chapter 5, page 89 were benchmarked on the RLPD dataset.

Two PPLMs that were implemented and presented in subsection 7.2.3, but that could not be used with KTP, were here benchmarked: the **NiTE-based** PPLM**:** and the face-recognition-based PPLM. The raw output of the NiTE algorithm, which is a user multimap [9], was saved during the recording of the RLPD dataset, enabling the use of the NiTE-based PPLM. While we didn't have face samples of the people visible in the KTP videos, here the face recognizer was trained with a set of faces of the staff of the lab.

Over ten different configurations were used. The five configurations using a single PPLM were used: 1. using the Euclidean distance PPLM only; 2. using the PersonHistogramSet (PHS)-based

[9] This was defined in subsection 3.1.3, page 36.

PPLM only; 3. using the height-based PPLM only; 4. using the NiTE multi-map-based PPLM only; 5. using the face recognition-based PPLM only; Then, some configurations were used relying on several PPLMs, based on combinations of these five available PPLMs.

A sample PeoplePoseList (PPL) message obtained by the multimodal fusion based on both the Euclidean PPLM and the face recognition-based PPLM is shown in Code listing 7.1. It contains three users, labeled 2="david", 1="irene", 3="jc". Sample images of the data supplied by some PPLM configurations are shown in Figure 7.13.

Note on user labeling: Some PPLM configurations have no user recognition against known users. This is for instance the case of the configuration shown in the left column of Figure 7.13, where we use both color-based matching and distance-based matching. The temporary inter-frame user names are "1", "2", "3". These names are coherent, so that a given user has the same temporary name between frames.
On the other hand, a face recognition PPLM using annotated faces with annotated names such as "Alice" and "Bob" can set absolute names to the tracks. This is illustrated in Code listing 7.1, where face recognition allows the matching of a temporary name to a meaningful name, like "david", and in the right column of Figure 7.13.

Code listing 7.1: *A sample PeoplePoseList message*

```
header:
  stamp: secs: 1422132943 nsecs: 848699111
  frame_id: /openni_rgb_optical_frame
method: /hal/ukf_EF
poses:
  - header:
      stamp: secs: 1422132943 nsecs: 662736721
      frame_id: /openni_rgb_optical_frame
    head_pose:
      position: x: -0.520989051784 y: -0.0111030681214 z: 1.63646026406
      orientation: x: 0.0 y: 0.0 z: 0.0 w: 1.0
    std_dev: 1.359126091
    person_name: 2
    confidence: 1.0
    rgb:   [image raw data]
    depth: [image raw data]
    user:  [image raw data]
    images_offsetx: 116
    images_offsety: 134
    attributes:
      names: ['user_multimap_name', 'initial_confidence', 'ukf_orien',
        ↪ 'ukf_speed', 'face_name']
      values: ['2', '3.22347', '4.00146', '-0.0443592', 'david']
  - header:
```

```
    stamp: secs: 1422132943 nsecs: 662736721
    frame_id: /openni_rgb_optical_frame
  head_pose:
    position: x: -0.0590717405732 y: 0.069477229913 z: 1.06420790185
    orientation: x: 0.0 y: 0.0 z: 0.0 w: 1.0
  std_dev: 1.359126091
  person_name: 1
  confidence: 1.0
  rgb:   [image raw data]
  depth: [image raw data]
  user:  [image raw data]
  images_offsetx: 149
  images_offsety: 25
  attributes:
    names: ['user_multimap_name', 'initial_confidence', 'ukf_orien',
      ↪ 'ukf_speed', 'face_name']
    values: ['1', '3.22388', '3.32724', '-0.012546', 'irene']
- header:
    stamp: secs: 1422132943 nsecs: 662736721
    frame_id: /openni_rgb_optical_frame
  head_pose:
    position: x: 0.41589570164 y: 0.0827769975227 z: 1.5227458961
    orientation: x: 0.0 y: 0.0 z: 0.0 w: 1.0
  std_dev: 1.359126091
  person_name: 3
  confidence: 1.0
  rgb:   [image raw data]
  depth: [image raw data]
  user:  [image raw data]
  images_offsetx: 402
  images_offsety: 109
  attributes:
    names: ['user_multimap_name', 'initial_confidence', 'ukf_orien',
      ↪ 'ukf_speed', 'face_name']
    values: ['3', '3.22501', '4.2713', '0.0791527', 'jc']
---
```

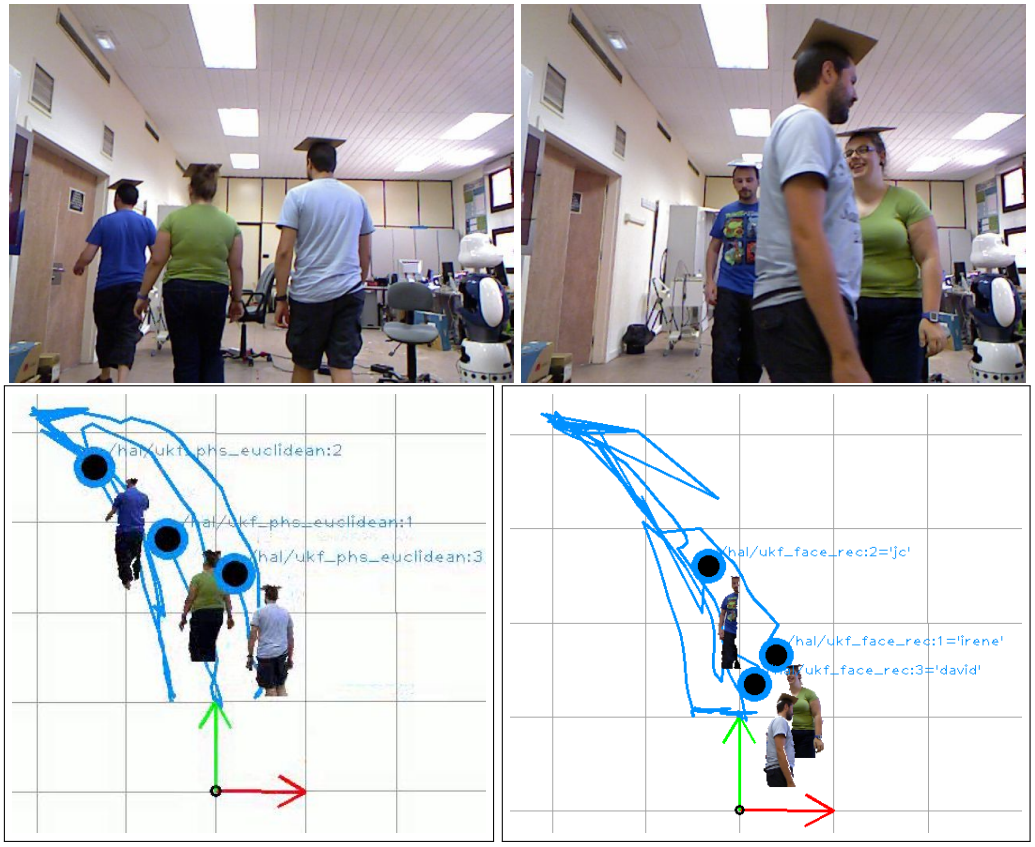


Figure 7.13: *Sample pictures of our user awareness architecture with the RoboticsLab People Dataset, with two different configurations. The lower row presents the different user tracks (blue lines), along with their user mask and names. User recognition between frames is performed by the* `PPLM` *configuration, we can know where the user has been and so display its trail. The images were obtained thanks to a visualization tool called* `PPLViewer`*, presented later on in subsection 8.1.2, page 199.*
On the left, the configuration used is "EP", using both PHS-based and distance-based matching. The three users are showing their back to the camera, and yet accurate tracking is performed thanks to the two matchers: note how the temporary names are coherent between frames.
On the right, the configuration used is "EF", using the face recognition matcher combined with the distance-based matcher. In this frame, a user is occluding the others. In this case, the user labels are displayed along with their names, obtained thanks to face recognition.

The results of the benchmark are gathered in Table 7.2.

Duration (seconds)	133
Total frames	647
Frames with several PPs	584

Conf. name	Euclidean	Face Rec.	Height	NiTE	PHS	Processed frames	ID swaps
E	X	-	-	-	-	99%	16
F	-	X	-	-	-	98%	20
H	-	-	X	-	-	98%	37
N	-	-	-	X	-	97%	16
P	-	-	-	-	X	95%	21
EF	X	X	-	-	-	98%	2
EH	X	-	X	-	-	99%	12
EN	X	-	-	X	-	98%	18
EP	X	-	-	-	X	98%	8
EFP	X	X	-	-	X	99%	8
EFHNP	X	X	X	X	X	83%	22

Table 7.2: *Database properties (top table) and benchmark results for different configurations of PPLMs on the RoboticsLab People Dataset (RLPD) (bottom). Each line corresponds to a different configuration, the PPLMss used in it are marked with the "X" letter. For instance, the last configuration, called EFHNP, uses all the PPLMs that we developed.*

The analysis of this table must be compared with Table 7.1, page 163, which was obtained using the KTP dataset. Before all, we can see that in the RLPD, each PPLM configuration is able to process almost all PPLs messages, in other words, no message is skipped, while in KTP many messages were skipped by the configurations needing more computational power. This is explained by the lower framerate of the acquisition, around 5 Hz vs 30 Hz in KTP.

Performance of each PPLM taken alone: we first can check that the Euclidean PPLM, corresponding to the E configuration, offers a very good compromise between computational needs and precision: it makes only 16 ID swaps, which is among the best single-PPLMs configurations, while requiring almost no CPU: practically all PPLs messages were processed.

The face-recognition-based PPLM corresponds to the F configuration. obtains a good performance too: its number of ID swaps, 20, is close to the performance of the Euclidean PPLM. However, the face recognition algorithm embedded in this PPLM was trained with a dataset of faces of people of the lab, which is a fairly small population. Its performance would have been lower with a greater number of people in the face dataset.

The height-based PPLM, corresponding to the H configuration, is the most error-prone: in the roughly two-and-a-half minutes of the video, 37 ID swaps are made. Like the KTP dataset, two of the three users have very similar heights, and for this reason their ID are frequently swapped. Using the height information alone, without any spatial or visual additional information, is a clue

but not enough.

The performance of the NiTE PPLM, powered by the NiTE algorithm and corresponding to the N configuration, is comparable to the Euclidean PPLM. This experimentally validates the robustness of the NiTE algorithm for user recognition. We could expect such a good tracking of the users: being the algorithm powering the user tracking for the XBox games using the Kinect device, there has been a lot of development and testing to ensure its accuracy.

And finally, the PersonHistogramSet (PHS) PPLM, corresponding to the P configuration, matches users from one frame to another uniquely using the color of their clothes. Once again, the number of ID swaps is comparable to the Euclidean matching, without using any spatial information.

Similarly to KTP, we can see that among the PPLMs, some perform better than others. Namely, the Euclidean-based and the NiTE-based PPLMs perform 16 ID swaps during the whole sequence, and the face-recognition-based PPLM get mixed up 20 times. At the other end, the height-based PPLM taken alone is more than twice more error-prone.

Performance of configurations relying on several PPLM: we also have measured the performance of configurations relying on several PPLMs. As presented in subsection 7.2.2, page 152, these configurations of data fusion call sequentially each of their PPLMs to obtain its corresponding cost matrix, then obtain the global cost matrix by making a weighted average of these matrices. Except one 2-PPLM configuration, EN, all the others (EF, EH, EP) obtains less ID swaps than the best 1-PPLM configurations, E and N, respectively 2, 12 and 8 ID swaps. In other words, like in the KTP benchmark, the tracking errors made by the multimodal fusion are fewer than each of the algorithms taken separately. The performance of the EF configuration can be underlined: during the whole sequence and in spite of the very challenging scenario, only 2 ID swaps are committed, which is the best performance obtained among all the tested configurations. We can explain this improvement: the Euclidean tracker is efficient for most situations and solves the ambiguities correctly. However, in complicated situations, for instance users disappearing through the door then reappearing, more sophisticated PPLMs such as face recognition, help solving correctly the matching.

When we increase the number of PPLMs involved in the multimodal fusion, the performance of the fusion is affected. For instance, merging the two best 2-PPLM configurations, EF and EP, into EFP (Euclidean + Face recognition + PHS), does not improve further the performance. The multimodal tracking capabilities do not compensate the computational overload generated by the successive calls of PPLMs. Worse, in some situations, while one of the trackers correctly matches the current users to the tracks, the others get mixed up and the final result is erroneous.

Conclusion: Both benchmarks on complicated datasets (KTP and RLPD) validate the structure of our user awareness architecture: *by using several parallel algorithms for detecting and matching users, then merging their outputs by multimodal fusion, we obtain a more reliable local user mapping*. The optimal configuration depends on the environment settings and the robot capabilities, and testing different configurations can be needed to determine the best one.

We will now present another solution for people tracking not using PPLMs, but instead based on a simpler technique: blob tracking in depth images.

7.2.5 Tracking of objects in a depth image

The first contribution of this PhD in data fusion was just explained. It consists in using data filters, namely Kalman filters, for giving data spatial coherence to the different user detectors across time. In this part, a different approach will be followed: when a user is first detected around the robot, his or her shape in the depth image corresponds to a blob we can characterize thanks to image processing techniques. We can then track this shape in the consecutive depth images using this characterization. The detection can still be made with a wide range of detectors as seen in Part I. This approach is less modular then the one using Kalman filters we just explained, and it is only made for single-user tracking, but it is more straightforward and easy to implement on a real robot.

The aim of this part is then to provide a mobile robot a lightweight algorithm for clusters detection and tracking. These clusters can be human users, or objects. The only required input data is the stream of depth maps. The Red Green Blue (RGB) data, i.e. color images, also supplied by the Kinect, is discarded for this algorithm. This ensures the compatibility of the algorithm with devices such as PMDVision CamCubes or Asus Xtion PRO devices, that do not supply RGB images. It should be able to be either processed on-board or on a remote computer, but the goal is to have it done on-line, in real time. As such, the visual tracking of the object-of-interest is made simultaneously with the navigation of the robot.

More accurately, the tracking is made at two levels: on the first hand, at the sensor processing level, the robot is expected to keep track of the followed cluster. On the second hand, we want the former to physically move itself and keep close to the latter. The whole process results in the robot coming to the tracked cluster and following its trail if it is moving.

7.2.5.i Approach

The whole processing pipeline is illustrated in Figure 7.14. Note that along with the description of each stage of the algorithm, sample captures will be visible in Figure 7.15.

It is detailed in the following order: the acquisition of depth images from the depth sensing device is detailed in subsubsection 7.2.5.ii. As previously said, the RGB (color) images that can be provided by the Kinect sensor are not used. Thus, the algorithm can also be used with single depth imaging devices, such as CamCubes.

Then, our tracking algorithm is articulated in three phases:

1. **Cluster detection:** the clusters are found in the current depth map thanks to 2D image processing techniques. This is presented in subsubsection 7.2.5.iii.

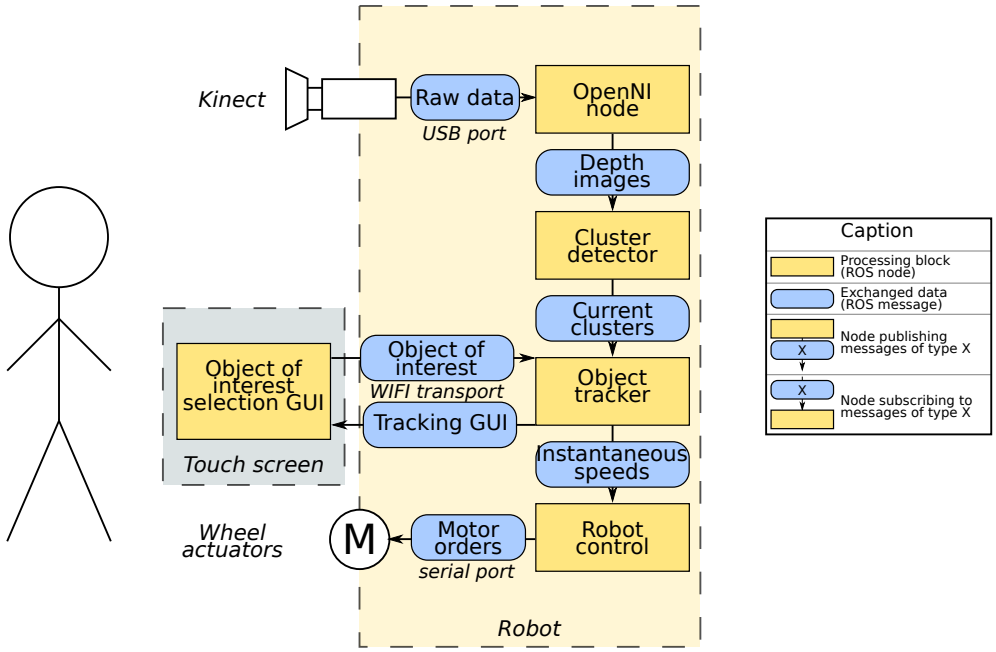


Figure 7.14: *The flow chart of the whole user tracking system. It presents all the components needed for the vision-based algorithm and does not include the fusion with the laser data presented later in subsubsection 7.2.5.v.*

2. **Cluster matching and tracking:** the clusters of the current depth map are matched to the objects found in the previous depth maps. If the user has selected an object to track, the three-dimensional (3D) position of this object is estimated. More detail is given in subsubsection 7.2.5.iv.

3. **Robot control:** depending on the tracking results, the robot motion is controlled. For instance, if the object selected by the user has been found, the robot will move and come closer. This is explained in subsubsection 7.2.5.v. Will also be presented how the results from our algorithm and from another one, based on the data of the laser, are mixed.

7.2.5.ii Depth image acquisition

We saw in subsection 3.1.3 how a depth imaging device, such as the Microsoft Kinect, can provide at a high frame rate depth maps indicating the distance to the closest object at each pixel. We however need to take into account that the constructed light patterns projected by the device, used to measure the distances, can be reflected, for instance by shiny surfaces. The depth map then contains some undefined values at those pixels, represented by `NaN` . A sample is visible in Figure 7.15 (a).

For an easier handling of the depth map, `NaN` values can be filtered. For instance, the Canny algorithm explained afterwards would fail with an image containing `NaN` values, as they are seen

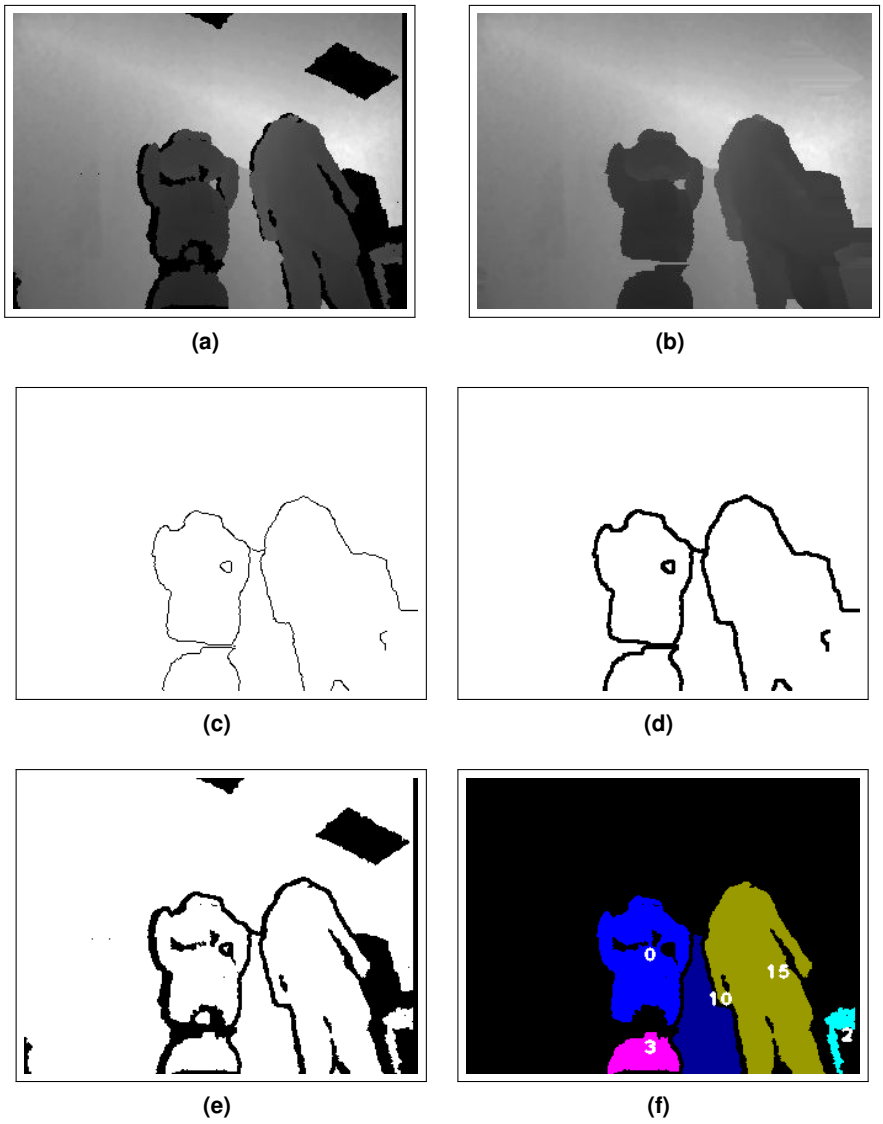


Figure 7.15: *The flow chart for the detection of connected components in the depth image.*
(a): The depth map, as supplied by the range-imaging device (Kinect), and remapped to visible colors (Hue scale).
(b): The "clean" depth map, after handling the `NaN` *values.*
(c): Canny edge detection on the clean depth map
(d): Morphological erosion of the Canny edges
(e): Recombination of the opened Canny edges and the `NaN` *values*
(f): The connected components found and their matches to objects. Each one is drawn with a color corresponding to the object it was matched to, and the index of this object is overlaid.

as a regular 0 value. To get rid of the undefined values, the values are replaced with the closest value to the left (or to the right if we are at the left border). This technique is called *directional propagation* (to the right). A sample is visible in Figure 7.15 (b).

7.2.5.iii Clusters detection

In this section, the method used to find the 3D clusters in the current depth map is explained.

To be able to make use of conventional vision techniques, the float depth map is first converted into an usual image with values in the $[0, 255]$ span, by using an affine mapping from the range of values of the map to $[0, 255]$.

Edge detection Now the depth images have been transformed into standard byte images and the undefined values have been handled, we can apply without problem "classical" image processing algorithms. As the goal is to find clusters in the depth map, we use a Canny filter on the remapped image as seen in section 3.2.1, page 39.

A sample is visible in Figure 7.15 (c).

The two parameters of the Canny filter are defined for values of the RGB space, and their meaning depend on the values of α, β. Hence, they are not consistent between frames. This is why instead of setting the thresholds, we set as constants the values of $\alpha \times low\_threshold, \alpha \times high\_threshold$ between frames. Parameter β is not taken into account as it is a constant offset factor, it is neutralized by the gradient simulated by the Sobel operator.

Morphological erosion We have explained previously how we get rid of undefined `NaN` values by directional propagation. However, it still can weaken local contrasts. To compensate this effect, we apply a morphological transformation that thickens the edges. This can close some edges that had been left opened by the Canny filter.

This result is obtained by passing a morphological erosion filter on the image. The erosion filter replaces each value in the image with the maximum of the values of the surrounding pixels. We use here a 3×3 pixels kernel. It could close gaps in the border that are up to 4 pixels wide. A sample is visible in Figure 7.15 (d).

Then, as we do not want to include the undefined `NaN` values in the objects, we need to restore the undefined values erased in the step of handling of undefined values. They are restored by: first, computing the minimum of the original remapped and the eroded edge map, second, thresholding this minimum with a binary threshold at value 0. A sample is visible in Figure 7.15 (e).

Fast connected component detection First, we define *connected components* for depth maps: two pixels of a depth map belong to the same connected component if there is a chain

from one to the other, such as there is no depth gap between two consecutive elements of the chain.

At this step of the processing pipeline, an edge map has been computed that also includes the undefined `NaN` values returned by the range-imaging device. A 3D object visible in the current depth map corresponds to a cluster without discontinuity in the inside, and with a discontinuity at the border with the neighbor pixels. It should then correspond to a connected component in our edge map.

In section 3.2.6, we presented how to quickly retrieve the connected components in a binary image. Our edge map being a binary map, we can use these methods. A sample is visible in Figure 7.15 (f).

The method also returns the bounding boxes of the components. We recall here that the bounding box of a set of 2D points is the smallest rectangle that fits all the points within its surface.

Cluster filtering Some filters can be applied for removing some of the non-relevant clusters found in the current depth map. The actual version of the algorithm discards the too small clusters. This is obtained by checking if the size of the corresponding connected component is under a given threshold.

7.2.5.iv Cluster matching and tracking

In the previous section, the detection of the different clusters in the current depth map was explained. However, our aim is to give temporal consistency to this cluster information: we want to match these clusters to objects detected in previous images. For instance, if the robot is following a person, we want to spot what is the cluster corresponding to this person in the current image. Having this temporal consistency for objects gives the possibility of a meaningful chasing motion for the robot.

Let us define the concept of *object* in the scope of our detector: an object, is a set of connected components of the successive depth maps, each of which corresponds to the same physical entity. There is, at most, one connected component in each depth map corresponding to a given object. However, it might be that some frames do not contain any connected component corresponding to a given object, i.e., the object might be occluded or not successfully recognized.

For the clarity of the concept, an example issued from real data is presented in Figure 7.16. It shows the recognition state of an object labeled 3. This object has been recognized five times in the previous depth maps. For each frame, we have stored the connected component and the bounding box representing it. On the other hand, the object has not been recognized in the depth maps number $9, 10, 11, 14, 15$.

In order to maintain the computation time as low as possible, the recognition is made in two steps.

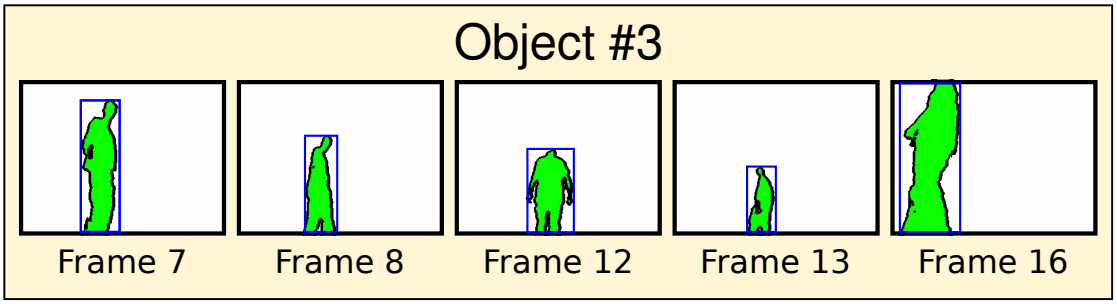


Figure 7.16: *A example to clarify the concept of an object representation. The data is real and comes from the tracking sequence of a given user. The object connected component is the filled area and the bounding box is marked as a rectangle.*

1. A first rough matching using only the bounding boxes of the components gives us a first estimation of which clusters correspond to which objects. This is explained immediately after.

2. Ambiguities are solved using an analytic distance with strong discrimination properties, the Hausdorff distance. This is explained in section 7.2.5.iv.

The definitive matching is the one obtained at the end of this second phase.

Rough estimator: bounding boxes correspondences Let us represent a given connected component C of the current depth map. We want to have a fast estimation of what object is the most likely to be matched with C.

Let us now consider a given appearance at frame i in the past of an object O in the recognition history, that we call O^i. We compare the bounding box bb_O^i of this appearance with the bounding box bb_C of C.

Definition of the bounding box distance Let us define a bounding box distance d_{bbox} such as $\forall X, Y$ bounding box, $d_{bbox}(X, X) = 0$, and the more similar X and Y, the smaller $d_{bbox}(X, Y)$.

Let us write $bb_X = \{TL_X, BR_X\}, bb_Y = \{TL_Y, BR_Y\}$ where TL refers to the top-left corner of the bounding box, and BR refers to the bottom-right corner. Then, we define d_{bbox} as defined in Equation 7.1. This corresponds to the sum of the distance between corresponding corners of both bounding boxes.

$$d_{bbox}(X, Y) = d(TL_X, TL_Y) + d(BR_X, BR_Y) \tag{7.1}$$

A distance function between 2D points, however, needs to be chosen. The most usual norms (L_1, L_2, L_∞) have been seen in section 5.1.3.i, page 97. Keeping in mind the speed of execution, the L_1 norm is here chosen. The (Euclidean) L_2 norm indeed needs expensive square root computations.

Application to the tracking The result of $d_{bbox}(bb_O^i, bb_C)$ gives us an idea of the similarity of O^i, i.e. the appearance of the object O at frame i in the past, and C, i.e. the current connected component of the depth map. The smaller the value, the more likely it is that C is an appearance of the object O in the current frame. In addition, to take into account the age of this appearance of O, we weight the obtained $d_{bbox}(bb_O^i, bb_C)$ by the age (in seconds) of the depth map where O^i belongs.

However, this estimation only takes in consideration the position of the objects in the frame, and not their shape. For our given connected component C, we compare it to all the appearances of all objects and store all the obtained marks in a list. We then sort this list by similarity, that is with the lowest d_{bbox} distances firsts. Then, we use the precise estimator presented in the next section iteratively on each element of this sorted list. This gives us the final distance d_{final} of C with each object that appeared the previous frames. [10]

Precise estimator: modified Hausdorff distance Using the rough matching with bounding boxes, matching ambiguities can occur. For instance, two connected components, similar in size and close to the last apparition of a given object, could both correspond to the same object then generate an ambiguous matching. For solving such ambiguities, we need a mathematical tool for properly comparing the shapes of components.

Modified Hausdorff Distance In [Dubuisson and Jain, 1994], a comparison is made between the different ways of computing a distance between two sets of points. According to these findings, we use the distance d_{22}, defined as in Equation 7.2.

$$\forall A, B \in \mathbb{R}^{2\mathbb{N}}, d_{22}(A,B) = max\left(d_6(A,B), d_6(B,A)\right)$$
$$\text{with } \begin{cases} d_6(A,B) = \dfrac{1}{\mid A \mid} \displaystyle\sum_{a \in A} d(a,B) \\ d(a,B) = \displaystyle\min_{b \in B} \|a-b\| \end{cases} \tag{7.2}$$

For a given connected component C, $d_{22}(C,C) = 0$, and given two connected components A and B, the lower the result returned by $d_{22}(A,B)$, the more similar they are.

d_{22} requires the choice of a norm for estimating the distance between two points.

[10] We can thus avoid the computing of d_{final} for the majority of the object appearances. If, for a given object appearance, its d_{bbox} is superior to the smallest d_{final} computed at that time, we can stop comparing all the following appearances in the list. They will indeed inevitably obtain a higher d_{final} distance than this match.

Accurate component comparator The Hausdorff distance helps to solve ambiguities. For each frame, all the components are first scaled to a given size, say 32×32 pixels.

Then, the different candidates for the same object are compared using d_{22} distance. In a similar way to how we weighted d_{bbox}, d_{22} is weighted by the age of the object appearance. The final distance is then given by:

$$d_{final}(O^i, C) = d_{bbox}(bb_O^i, bb_C) + d_{22}(O^i, C)$$

The best object candidate for the recognition of C is the one getting the lowest d_{final}. It is considered as a positive match if it gets a mark inferior to a given threshold. This mark is empirically determined and can be set through the Graphical User Interface (GUI). Its default value is 0.8.

Object tracking: selection of the object-of-interest At this step, the connected components of the current depth map are matched to the objects already found. However, the tracking needs to know what object we aim at tracking. In other words, the user needs a way to select the object-of-interest. A Graphical User Interface we have developed and working with an auxiliary touch-screen computer will be presented in section 7.2.5.vi.

The 3D pose of the object-of-interest is obtained thanks to the 3D reprojection of the barycenter of all points of the object-of-interest. The cluster matcher periodically republishes this 3D pose of the tracked object-of-interest.

7.2.5.v Robot motion and user tracking

Multi-sensor fusion As all range-imaging devices, the Kinect is limited by a field of view. Horizontally, it can see objects that belong to an angular domain of 57 degrees, and of 43 vertically. Neither can it detect objects at a distance inferior to 1.2 meters according to the Kinect datasheet. Furthermore, for higher distances, the further the object, the lower the precision. On the other hand, the robot is also equipped with a Hokuyo laser scanning range finder, which can only detect objects on a horizontal plane, but with a field of view of 240 degrees. However, its range is limited to approximately four meters. Both fields of view are compared in Figure 7.17.

The poor visibility of the Kinect device results in objects moving laterally getting easily out of its view spectrum, and hence getting lost by the tracking algorithm, while they remain visible for the Hokuyo laser. On the other hand, distant object are out of range for the latter.

Thus, to enable a robust tracking even with challenging trajectories, including curves with hairpin turns, we chose to combine two different tracking algorithms: the vision-based method presented before and another one based on the laser data only. The latter consists of a simple 2D cluster-tracking algorithm. A cluster is defined by a continuous set of points where each point is separated from its neighbors by a distance inferior to a given threshold, say 35 cm. The tracking is initialized thanks to a 3D seed point supplied to the tracker.

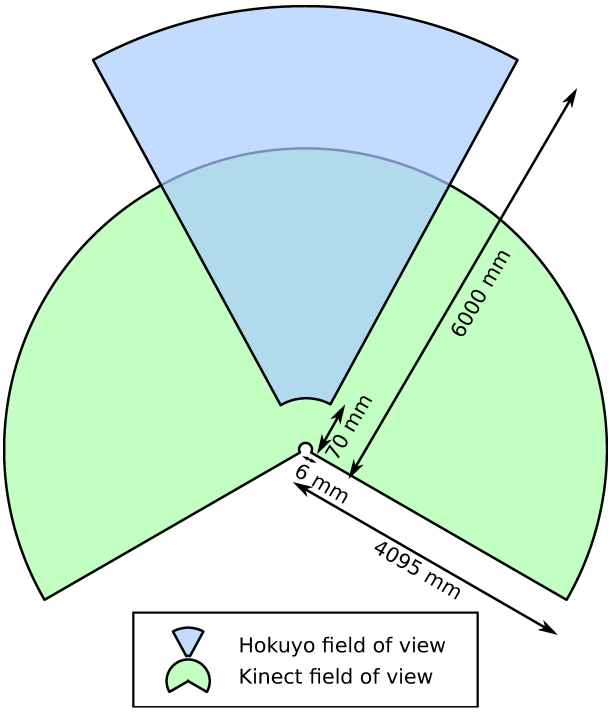


Figure 7.17: *Field of view of both sensors mounted on the robot: the Kinect depth camera and the Hokuyo range finder.*

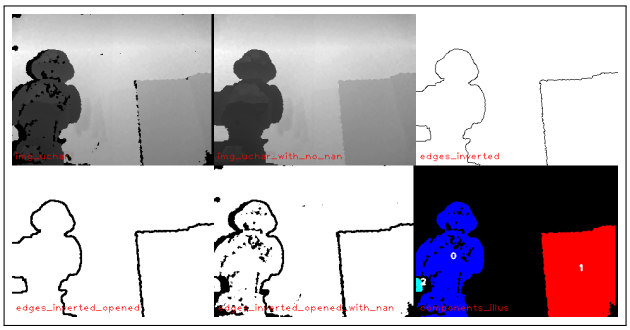
These two tracking algorithms run simultaneously on the robot. They publish periodically their status, and the 3D pose of the object-of-interest obtained by the tracking (as defined previously). On top of them, a *dialog* node decides what algorithm has priority, and reinitialize the other thanks to the 3D pose it returns. When it performs successfully, the vision-based algorithm presented before always has priority on the laser-based one. As such, if the tracked object is located in the view frustum of the Kinect device, the former is used before the latter. The latter enables the tracking to go on successfully outside of the view frustum, and recovering from eventual failures of the vision-based tracking algorithm.

Goal navigation The dialog node republishes the resulting 3D pose of the tracking algorithm that has priority. This is used as a *goal* for the motion planning system, that is the point to reach by the robot. This goal is moved after each iteration of the tracking algorithms. This corresponds to each acquisition of a depth map by the range-imaging device.

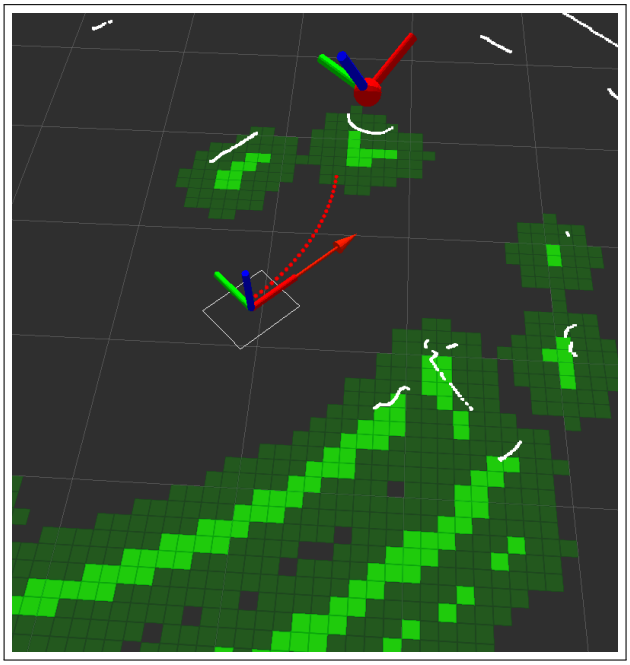
The motion control is made through a Dynamic Window Approach, as presented in [Fox et al., 1997].

The admissible velocities, that is, the ones where the robot is then able to stop without collision, are determined with the local costmap. This local costmap is obtained via the fusion of the laser scanning range finder and the reprojected Kinect point cloud.

When the robot is close enough to the goal, the robot keeps steady till the goal moves again. An example is visible in Figure 7.18.



(a)



(b)

Figure 7.18: *Motion planning towards the goal at a given time.*
(a): The detection status at a given time. The tracked object is the shape with a 0 *index on the left of the bottom-right image.*
(b): Visualization of the plan of the robot. The white spots correspond to the laser scan. The light gray squares represent the local costmap, and the dark ones to its inflated version (inflation by a radius corresponding to the one of the robot). The robot is indicated by its white footprint and its frame axes. The goal is the sphere with the tilted axes on top of the image. The direction indicated by the arrow corresponds to the orientation of the Kinect within the robot. The dotted curved line corresponds to the planned trajectory until the goal.

7.2.5.vi Experimental Results

The method explained before has been implemented into the robot MOPI, already presented in the Introduction chapter. The experiments have been made in several phases. First, the GUI developed to select the object-of-interest is presented. Then the times needed for the algorithm to run in several hardware platforms is presented and discussed. After that, how the workload can be distributed between several computers is explained . And finally, the accuracy of the whole tracking algorithm is measured. The success rate of the process is measured, while the user goes across a marked path with obstacles and occlusions.

Selection of the object-of-interest thanks to a GUI A GUI interface was developed for the touch screen. It enables the user to select the object-of-interest by clicking on it. The touch-screen computer used is a TravelMate C110 computer equipped with a touch-sensitive screen. It is equipped with a CPU of modest performance dating back from 2003. This makes it appropriate for selection and drawing by users via Graphical User Interfaces (GUIs), but not for heavy computation.

A sample is visible in Figure 7.19.

Figure 7.19: *A sample picture of a user selecting the object-of-interest. It is indicated by the shape with a white border. The other objects are drawn with a different color.*

Time costs for clusters detection and tracking The tracking algorithm has been tried in several hardware architectures. It was first used on the on-board computer of the MOPI robot, which is an embedded computer with a CPU with limited capabilities (Intel Atom CPU Z530 @ 1.60Ghz). In a second configuration, everything was processed on the touch-screen, which dates from 2003 (Intel Pentium M @ 1000Mhz). And finally, a desktop PC with a full-power

processor from 2008 was used (AMD Athlon 64 X2 Dual Core Processor 5200+).

The time needed for running the algorithm is shown in Figure 7.20. The on-board computer needs around 50 milliseconds for a performing a cycle of the algorithm. As such, it can run up to 20 Hz. However, it represents a workload for the CPU with limited capabilities. The use of a remote computer for processing solves this issue.

It would also be possible to run the algorithm in the touch screen, reducing the number of devices to two. However, it is preferable having a smooth GUI running at high frequency to a choppy display that might cause trouble to the user when selecting the object-of-interest of her choice. This would also reduce the battery autonomy of the touch screen.

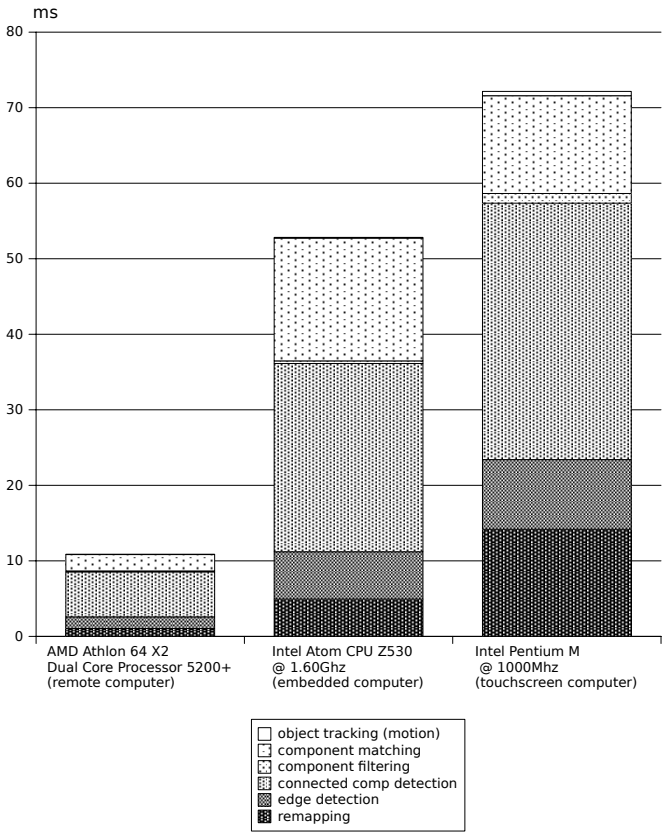


Figure 7.20: *Time needs for running the algorithm on different hardware platforms. The time allocated for each step is also indicated.*

Workload distribution among several computers It is possible to make use of the communication strengths of Robot Operating System (ROS): several computers can share the same data and the different subtasks can be distributed between them, without affecting the result.

As such, it is possible to send via a wireless connection the raw depth maps to a remote

computer. It will run the algorithm presented previously and return the 3D pose of the object if found.

The whole architecture for processing is then structured as presented in Figure 7.21. The task demanding much CPU is the detection of the clusters. It has been moved to a remote computer with a faster processor. The results of the processing are sent back to the robot embedded computer.

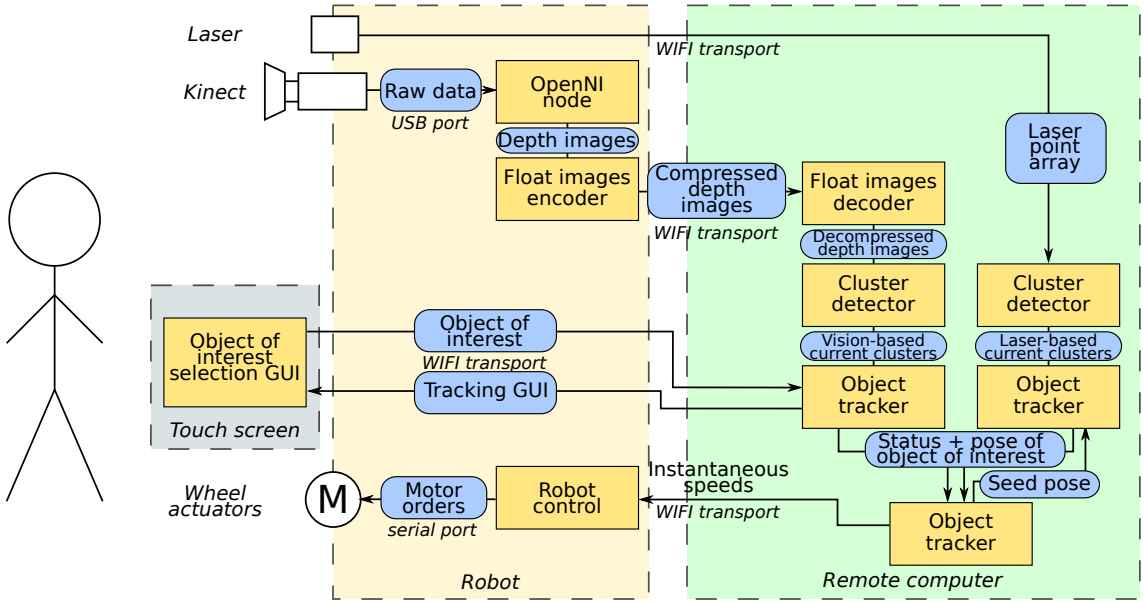


Figure 7.21: *The flow chart of the whole system when distributed between several computers. It also includes the laser data fusion presented in subsubsection 7.2.5.v.*

Tracking accuracy on a complicated path The multi-sensor fusion has been presented in subsubsection 7.2.5.v. It gives a high priority to the presented vision-based algorithm, and uses a laser-based fallback tracking to enable a successful tracking along a complicated path.

We measured the performance of the tracking system on an unstructured lab environment, which represents a complicated path. The path followed by the user is visible in Figure 7.22. The length of the whole path that the robot has to follow, from its start position to the finish circle, is around 33 meters. It includes straight lines, hairpin turns, and narrow passages.

A user initializes the tracking on him thanks to the GUI presented in section 7.2.5.vi, then follows the path without giving more orders to the robot, until reaching the finish line. A run is declared as *successful* if the robot follows the user along the whole path, does not collide with any obstacle, and reaches the finish line. The experience is repeated for twenty runs.

The results are presented in Table 7.3. Some screenshots of the GUI, obtained during a tracking sequence, are visible in Figure 7.23 and Figure 7.24. [11] 90 % of the runs have finished successfully, in an average time under two minutes. The cases of failure are most often due to a wrong match of the user's shape from one frame to the another. For instance, during one of the failed runs, the robot incorrectly recognized a passive observer as the tracked user, and started tracking him. A standard deviation of more than ten seconds can be noticed. The time needed to go along the whole path depends indeed on the speed of the user, which may vary from one run to another.

This high success rate experimentally validates the robustness and usability of the developed tracking system.

Number of runs	20
Success rate	90 %
Average time for successful runs	114 s
Standard deviation of average time	12.8 s

Table 7.3: *The result of the tracking runs along the complicated path.*

[11] The full output of the algorithm, is visible in an on-line video here. It lasts about two hours.

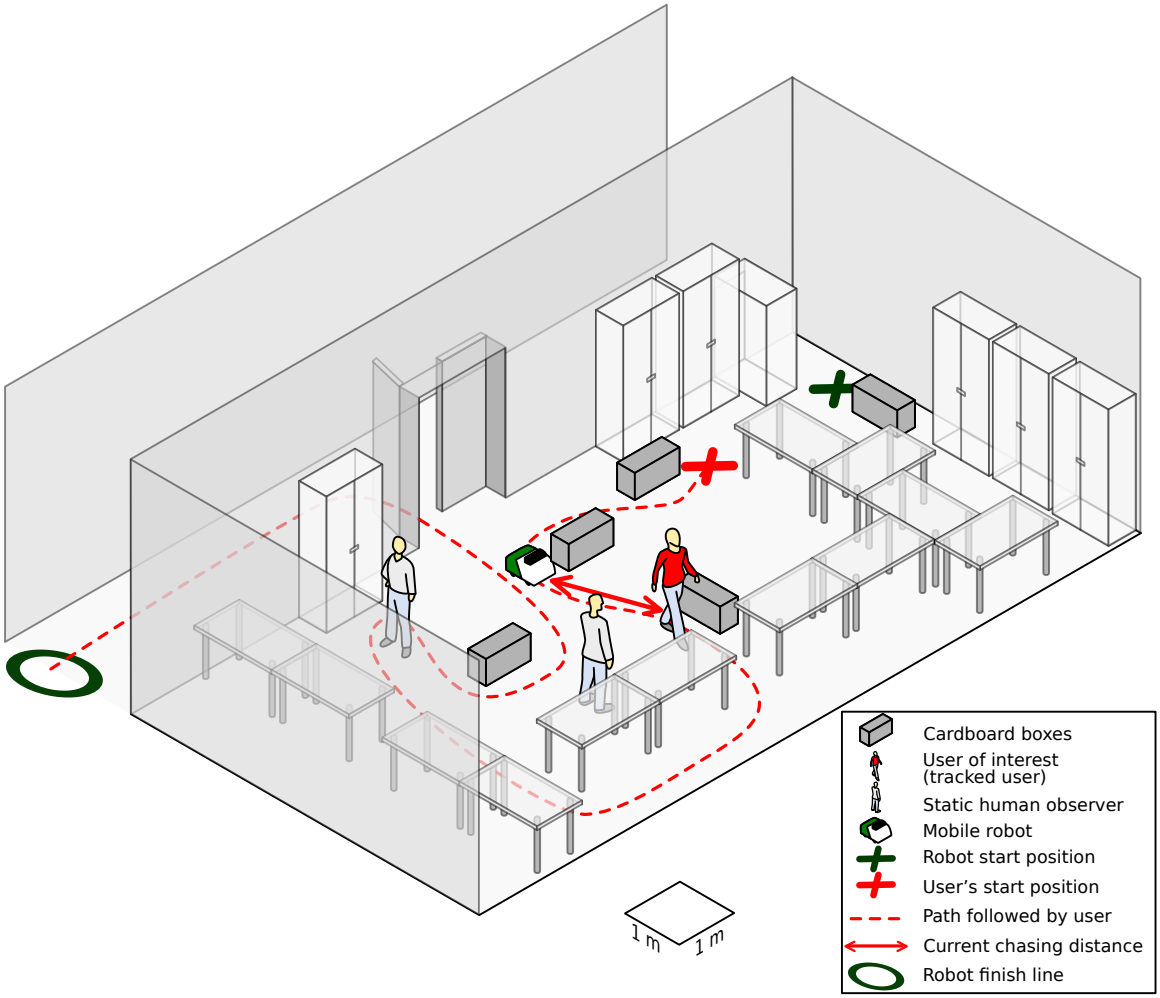


Figure 7.22: *The path followed by the user to test the tracking accuracy. The gray boxes correspond to cardboard boxes aimed at making the path planning harder and preventing the robot from cutting the curves.*
The robot intends to maintain the chasing distance as close as possible to the goal distance. When the robot enters the circle without losing the track of the user, the test is marked as successful.

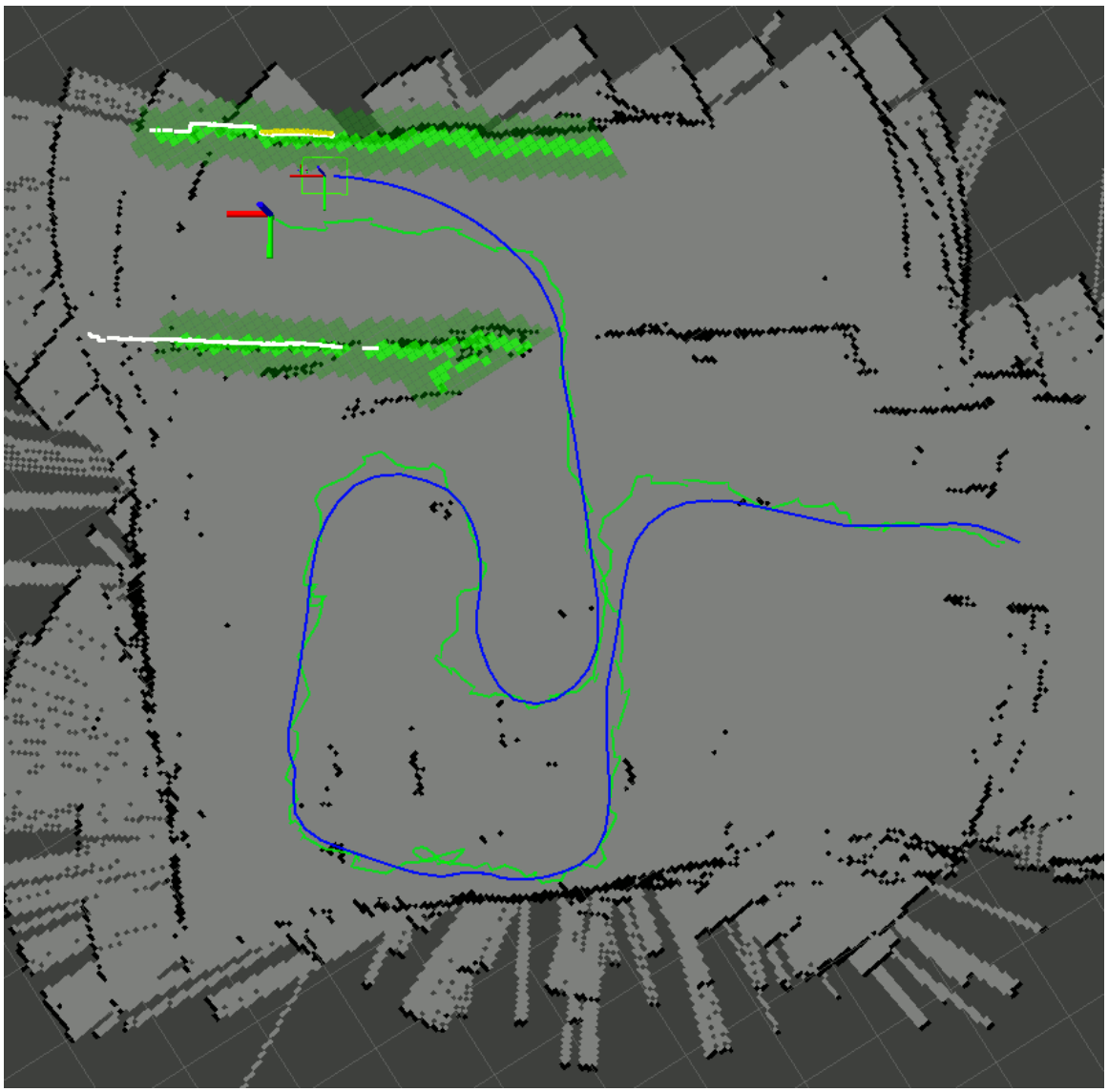


Figure 7.23: *Map generated by a SLAM algorithm (GMapping [Grisettiyz, 2005]) overlaid with the paths generated during a run.*
The light gray area corresponds to the free space of the map and the black edges its occupied space.
The irregular line is the path of the user, as detected by the algorithm.
The smooth line ending into the rectangular shape corresponds to the path of the robot, as determined by its odometry.
The remaining symbols are identical to the ones used in Figure 7.18 (b).

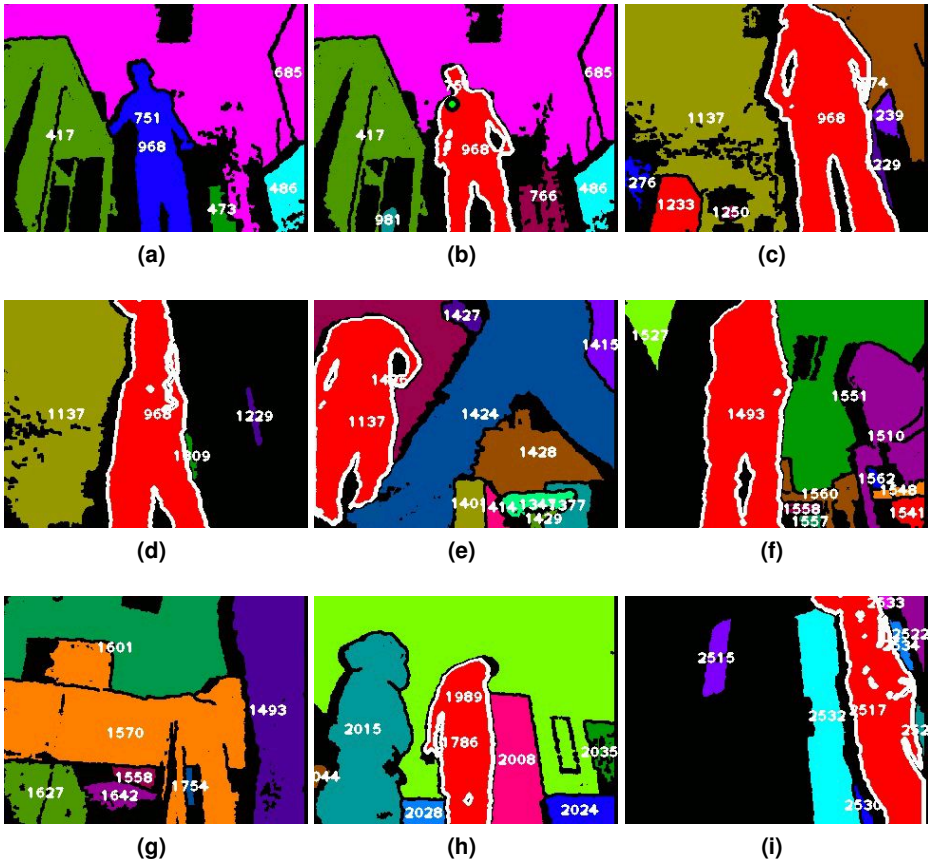


Figure 7.24: *Different frames of the GUI during a tracking sequence on the test path. The numbers stand for the objects names. The object-of-interest has a white stroke.*
(a): GUI just before the user initialization, and (b), just after. The selected user then is marked with a white stroke (cluster 968*).*
(c): Shortly after initialization, the robot has come next to the user (cluster 968*), who already started walking along the path.*
(d): The user starts walking in a transverse direction to the robot motion, which triggers a fast on-place rotation.
(e): Another user (cluster 1424*) crosses the path of the robot. The tracking is not affected.*
(f): In the narrow passage between the tables (cluster 1510*) and the wall.*
(g): The vision-based tracking loses track of the user (cluster 1493*) after a sharp turn. The tracking goes on thanks to the laser-based tracking.*
(h): After the sharp S-turns at the end, the final door is visible (cluster 2008*).*
(i): Approaching the final door (cluster 2532*).*

Summary of the chapter

The previous parts of this PhD presented methods for detecting and recognizing users around a social robot, by using various algorithms mainly based on vision and image processing, but also on other devices and methods. However these different detection and recognition algorithms did not provide any temporal coherence to the results given. This is why, in this chapter, we presented two methods to recognize and track given users by the robot, and hence to build a local map of the users around the robot.

The first method uses an Unscented Kalman filtering algorithm for merging the output of these different detection and recognition methods. A standard interface for the recognition algorithms, called `PeoplePoseList` Matcher (`PPLM`), was also defined to help integrating smoothly the algorithms into a common fusion process.
Note that the proposed system offers a range of advantages. It is highly modular, which means that one can choose the different detection and recognition components, according to the physical configuration of the robot and the hardware limitations. It makes possible the workload distribution, thanks to ROS communication layers. A new algorithm is easily integrated: it just has to comply with this interface, and it can be implemented in the programming languages supported by ROS: C++, Python and Perl.
The experimental usefulness of the proposed solution was experimentally benchmarked. We used a challenging academic dataset on the one hand, and realized from scratch a new dataset supplying realistic data for Human-Robot Interaction (HRI) on the other hand. Some very interesting results were obtained. First, simple tracking algorithms that perform at a very high rate give better results than sophisticated ones that take a long time to compute. The distance tracker worked much better than the height-based one for instance. Second, the combination of different `PPLM`s improved the tracking accuracy compared with the output of each of these trackers taken separately. In other words, using a set of trackers adapted to a robot helps having a precise detection and tracking of the users around this robot.

The second method takes a different approach. Instead of aiming at tackling the recognition of all visible users simultaneously, which is an error-prone process, it focuses on tracking accurately one given user. This given user is initialized either thanks to one of the detection algorithms, called `PeoplePoseList` Publisher (`PPLP`), defined in earlier chapters, or thanks to a Graphical User Interface. The tracking of this given user of interest does not use the different `PPLM`s, but instead focuses on a straightforward method with a minimized computational overhead, based on the shape of the objects in the depth image. Similarly to the the first method (`PPLM`s), this depth-tracking method was extensively benchmarked with real life tests, thanks to dynamic tracking processes of a user across a crowded stage. This second method is simpler than the `PPLM`s-based one, but it offers great stability and little computational needs, and turns out to be perfectly adapted for human following situations, such as *follow-the-leader*.

Thanks to the work developed in the three first parts of this PhD, we now have a robust, modular and fast person detection and tracking system building local user maps for social robots. It is aimed at being used by end-user applications: the next part will present some applications relying on our system.

Part IV

Applications for the system

CHAPTER 8

Applications of our user awareness architecture for social robotics

The three first parts of this PhD presented how we implemented a user awareness system into social robots, our approach being made of three parts: user detectors, user recognizers, and data fusion thanks to Unscented Kalman Filters. In this chapter, we will focus on possible applications for such a system.

Indeed, user awareness in HRI can be put to use for a wide range of applications. We have seen in the chapter linked with the literature review, chapter 2, page 15, how user awareness is a key component for social robots, and how the applications for such skills are numerous. Please refer to this chapter for a detailed list of robots using user awareness.

In addition with what was said in that chapter, note that not only end-user applications can benefit this architecture. It can also help tackling some more advanced problems that are also required by end-user applications. An example is **user action recognition**. This handles with understanding what the users are doing: thanks to their motion and behavior, their intentions and actions can be understood. For instance, dangerous actions such as children playing with electric plates, or anomalous behaviors such as elderly people falling in staircases can be detected. ([Schüldt et al., 2004, Laptev et al., 2008]) A survey of such algorithms can be found in [Poppe, 2010].

In short, applications for user awareness are numerous. They go beyond the scope of the work presented in this PhD, but we chose to adapt a few, as samples of the modularity of the whole system. We will first present how we adapted our system in the most agile possible way to fit to this range of possible applications, in section 8.1. Then, the following two sections will

explain how we integrated two sample end-user applications: surveillance, section 8.2, page 203 and games, section 8.3, page 215.

8.1 Use of our user awareness architecture and associated tools

The work presented in this PhD dissertation presented the implementation of a full user awareness system. Its final output consists of a so called `PeoplePoseList` (`PPL`) message that was defined in subsection 3.2.2, page 41 and that contains all the relevant information about the users that could be gathered: the number of users, their position, their most probable identity, etc.

We will in this section explain how to use the developed system for building end-user applications that also sport the user awareness architecture.

8.1.1 Using `PeoplePoseList`s (`PPL`s) for end-user applications

Our main contribution, the user awareness system, is implemented in several Robot Operating System (ROS) packages. Here comes an explanatory list.

- `people_msgs`: this is where the proper `PeoplePose` (`PP`) and `PPL` messages are defined. Along with these come some useful applications for writing both `PeoplePoseList` Publisher (`PPLP`) and `PeoplePoseList` Matcher (`PPLM`) applications, and for testing them. Also come some visualization tools, we will come back on that later on.
- `people_detection_vision`: in this package are present all the vision-based `PPLP`s (Part I, page 31).
- `people_recognition_vision`: All the files implementing the vision-based `PPLM`s are present in this package (Part II, page 89), along with the Unscented Kalman Filter (UKF) used for the multimodal fusion (Part III, page 139).

To be able to use `PPL` messages in a ROS node, you need to depend on the `people_msgs` package. This package contains almost exclusively the definitions of `PPL`, which means it is a very light dependency.

However, if you want to bring up the user awareness pipeline, you need to launch some ROS nodes belonging to the two other packages. Some *launch files* [1] are present in these packages, that present different possible detection and recognition pipelines. Most notably, `people_detection_vision/all_pplp.launch` will launch all implemented `PPLP`s, along with a Graphical User Interface (GUI) to start and stop each of them, while `people_recognition_vision/all_pplm.launc` contains all implemented `PPLM`s.

[1] ROS launch files are XML files that contain a list of ROS nodes, along with their parameters, that will be all launched at once.

The applications that we will present in the following sections, for instance, use PPLs for user detection. Consequently, the package they belong to, called games_vision, depends on person_msgs. The full list of dependencies can be seen on Figure 8.1.

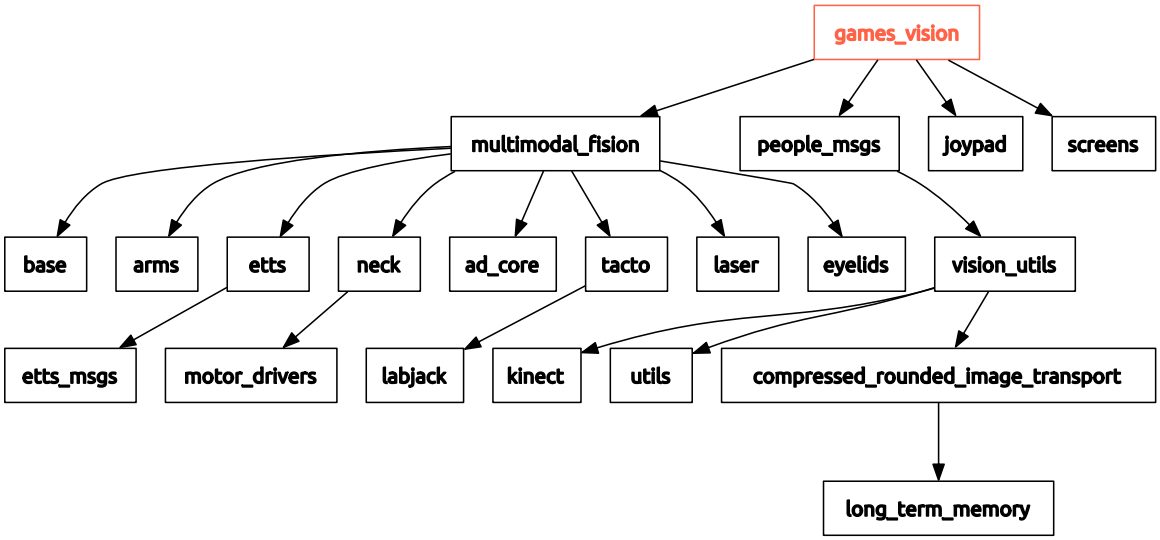


Figure 8.1: *The package dependencies of games_vision, the package containing the different applications. Note how the end-user package, games_vision, depends on people_msgs, the package defining the PeoplePoseList message, but not on the packages defining detectors and matchers, people_detection_vision and people_recognition_vision*

8.1.2 User visualization tools

User awareness, the capacity to know how many users are around the robot.

However, the knowledge delivered by the PeoplePoseLists (PPLs) can be somewhat cumbersome to visualize. It is a complex message that encapsulates abundant data, and the different three-dimensional (3D) positions of the users are given in the coordinates frame in which they were detected, which is often the camera optical frame, while it is easier to represent things in the robot frame.

For this reason, two different visualization systems were developed and will now be presented: on the one hand, ROS rviz markers taking advantage of the ROS visualization native tool, on the other hand, PPLViewer, a lightweight, self-contained and simple tool for displaying the PPL emitted by the robot.

8.1.3 ROS Rviz Markers

The Robot Operating System (ROS) framework includes a wide range of useful tools for visualizing data.

For instance, *rqt_plot* can plot in real time any numeric field of a topic. In our case, we could plot the confidence of the detection, thanks to the field `confidence` belonging to $[0, 1]$ and contained in the `PeoplePoseList` (`PPL`) message defined in subsection 3.2.2, page 41.

Another useful tool, *rqt Package Graph*, shows the dependencies between ROS packages in a graphical way. This tool was used to generate the previous Figure 8.1.

The tool we will focus on in this subsection is the ROS visualization tool, called *rviz*. It is a three-dimensional (3D) viewer, based on OpenGL and Ogre. It gives a representation of the world, in a given, selectable, coordinates frame. The user can scroll, tilt and zoom the view thanks to the mouse. Many kinds of topics are convertible into some display inside `rviz`. For instance, the robot footprint can be displayed as a polygon, the different registered coordinates frame can be displayed thanks to 3D arrows. New message types can be converted into visualization in two ways: using `rviz` plugins or through markers. The first one consists of writing a C++ chunk of code that converts the message into OpenGL primitives. The second consists of using an intermediary ROS message type, called *Marker*. They are actually a way of displaying different OpenGL primitives straight from the node code. Supported primitives are sphere, points, cubes, 3D text, etc.

We wrote a generic `rviz` wrapper, called `ppl2marker`, that subscribes to a `PPL` topic and converts it into a ROS marker. For each `PeoplePose` (`PP`) of the received message, a sphere is displayed in the user position, and another circle displays the confidence of the detection: the wider the circle, the less reliable the detection. Finally, a 3D text displays the detector used and the user name. A sample is visible in Figure 8.2.

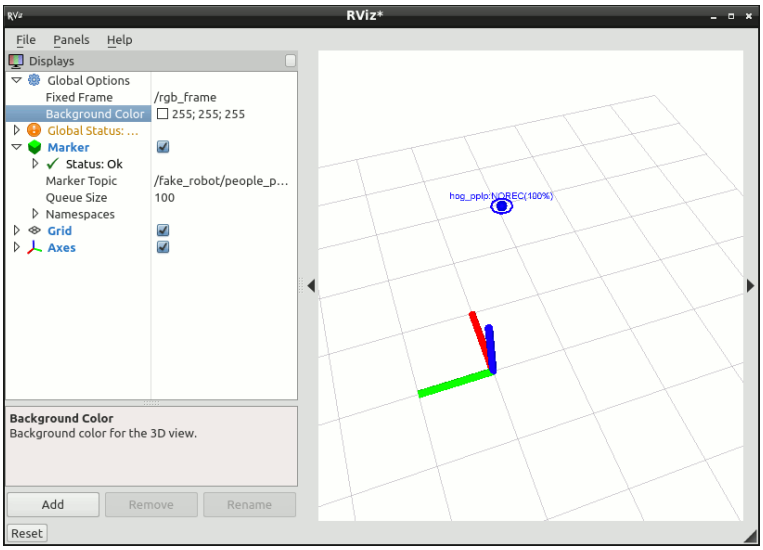


Figure 8.2: *A sample rendering of the PPL visualization tool rviz with PPL markers.*

This tool is adaptable, as the `PPL` topic of the wrapper is reconfigurable via a ROS parameter, and so are the visual properties of the marker (Red Green Blue color, persistence in second,

etc.). It also blends in perfectly in a typical ROS architecture, as it converts our custom data type into `rviz`, one of the key tools of ROS.

8.1.4 PPLViewer

However, it might not always be desirable to use `rviz`. Among others, it is a fully accelerated three-dimensional (3D) visualizer, that requires heavy CPU and GPU resources, thus being impractical on robots with limited hardware, and that cannot be run via remote connection (`ssh`).

For this reason, we developed another, much lighter, two-dimensional (2D) tool, called `PPLViewer`. It is written in C++, using OpenCV ([Bradski and Kaehler, 2008]) drawing capabilities. It offers a birdview representation of the world, and can subscribe to various `PeoplePoseList` (`PPL`) topics at the same time. All these topics are then drawn on the interface in real time. In addition, the previous position for each pair (user name, detector name) is also stored in memory, which enables the drawing of the trail of each user. This is a polygonal line going through the previous positions of that user. On top of that, the color image of the user is drawn next to her trail in the interface. A sample is visible in Figure 8.3.

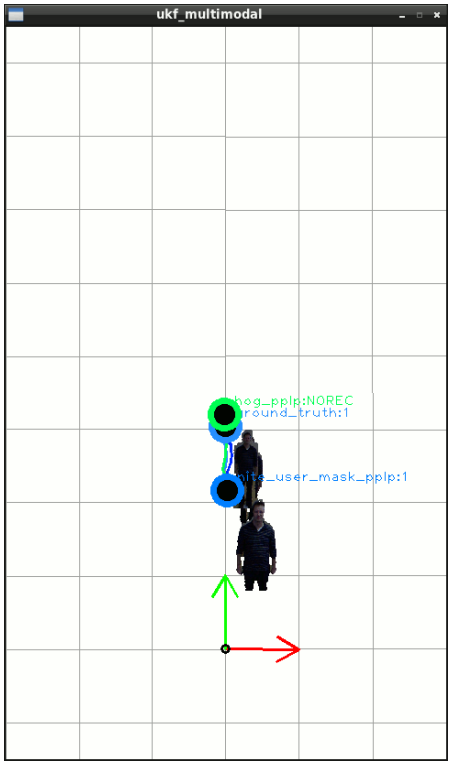


Figure 8.3: *A sample rendering of the `PPL` visualization tool `ppl_viewer`.*

This allows us to see in real time the result of different `PeoplePoseList` Publishers (`PPLPs`) and the output of the Unscented Kalman Filter (UKF)-based multimodal fusion. While running

a benchmark, we can visually compare if the output is correct by comparing the tracks of the different detectors, with track of the ground truth.

Conclusion In this section, the different applications in social robots using user awareness were present. We then presented how to use our own user awareness system for end-user applications. The development of a Robot Operating System (ROS) package using our system is straightforward: this package needs to depend on the package defining the `PeoplePose` (`PP`) and `PeoplePoseList` (`PPL`) messages. Then the two visualization tools for `PPL` were detailed. One uses the ROS visualization environment, called `rviz`. The other offers a minimalistic 2D viewer that shows in real-time the trails of the different users for several `PPLP`.

8.2 Example of use #1: Surveillance

The previous parts of this PhD dissertation presented the multi-modal, modular, distributed user awareness system I developed. Part I was focusing on user detection thanks to different sensor inputs and algorithms, Part II was about user recognition from one detection to another, and Part III merged the results of these different detectors thanks to the results of the recognition. This section will present an application of the whole system: *personalized surveillance of an Alzheimer's patient by a social robot*. Note that the application we will describe can in fact be used for any kind of surveillance, but it will be applied to this special category of people, as part of a broader work with Alzheimer's caretakers.

Surveillance consists of observing a given zone over time and determining if intruders enter that zone. In such a case, the surveillance application is in charge of triggering an alarm system. This can be notifying a human operator, who can be either at the same place as the robot, or activating other emergency mechanisms.

However, it is hard to control an area with *selective rules*, that depend on the identity of the detected person, i.e. allow certain people to be in that area and other not. Our system gives this possibility.

8.2.1 Problem definition

Surveillance has been an active field of research for the last decades. Most related work focus on the analysis of the Red Green Blue (RGB) or greyscale image, cf for instance [Haritaoglu et al., 1999, Haritaoglu et al., 2000, Bouchrika and Nixon, 2006] Other include depth information thanks to stereo vision, or other physical devices. [Kepski and Kwolek, 2012] for instance presents a monitoring algorithm that detects when the people around the robot fall on the ground, based on the depth image analysis.

However, most of these surveillance systems rely only on one detection algorithm. Furthermore, they do not always include any mechanism of user recognition, as all detected persons are treated equally. Such limitations are acceptable when the goal is to ensure nobody enters a forbidden area, say, next to the edge of a cliff. However, in most scenarios, not all persons have the same rights and permissions: for instance, a banker will be allowed to enter the vault of a bank, while a client will not. Such scenarios require an accurate person detection and recognition.

The user detection and mapping system presented in the previous chapters gives us the ability to make *selective surveillance*, in other words, the behavior of the robot can depend on the identity of the persons detected.

Alzheimer's patients caretaking The RoboticsLab has a collaboration with the *Spanish Alzheimer's Foundation* or Fundacion Alzheimer España (FAE) [2]. This project aims at helping both patients of Alzheimer's and their caretakers by the means of robotics. We aim at building small robots that are present in the house of the patients for long term instances.

The goal is not that the robot substitutes the caretaker, but on the other hand that it helps her in the most tedious daily tasks. Indeed, most caretakers are close family: the husband or wife of the patient, her son or daughter, etc. Taking care of a patient is an exhausting task, first because of the very important amount of time needed, second because taking care of a beloved one losing one's capacities is a terrible burden.

One of the tasks that can be heavily automatized for the wellbeing of the caretaker is the surveillance of the patient: the patients have trouble staying quiet at night, and tend to wake up and stroll around the house.

This creates a perfect testbed for the person detector we presented in the previous chapters: the caretaker is allowed to go wherever she wants, while there are some so-called dangerous zones for the patient. When dangerous activities occur, such as entering the kitchen at night, going through the entrance doors, the caretaker should be warned.

8.2.2 Implementation

The surveillance system uses the user awareness system, based on `PeoplePoseList` Publishers (`PPLP`s) for detection and `PeoplePoseList` Matchers (`PPLM`s) for recognition and tracking.

The surveillance system can be articulated into different modules: 1. definition of the forbidden zones by the caretaker; also called **Costmap**; 2. user detection and recognition, shaped as a `PeoplePoseList` (`PPL`) and performed by a set of `PPLP`s and `PPLM`s; 3. **Costmap watchdog**: detection of anomalous activity thanks to the costmap and the `PPL`; 4. caretaker notification.

The full pipeline is visible in Figure 8.4. The user awareness architecture presented in the previous parts corresponds to the first orange block, the surveillance pipeline is in the second block. Note that the definition of the forbidden zones and the detection of the users are two independent, concurrent modules; the fusion is made in the anomalous activity detector.

Definition of the forbidden zones We first need to define the forbidden zones in the room. We can think of forbidden zones semantically (for instance, the door, the oven, etc.) or geometrically, thanks to three-dimensional (3D) coordinates.
The former solution is higher-level and corresponds more closely to how we conceive the potential dangers for the patient. However, it requires the identification of these semantic concepts during the phase of surveillance (for instance, is the patient using the oven, or is he just next to it?). Such an identification is challenging and error-prone.

[2] `http://alzfae.org/`

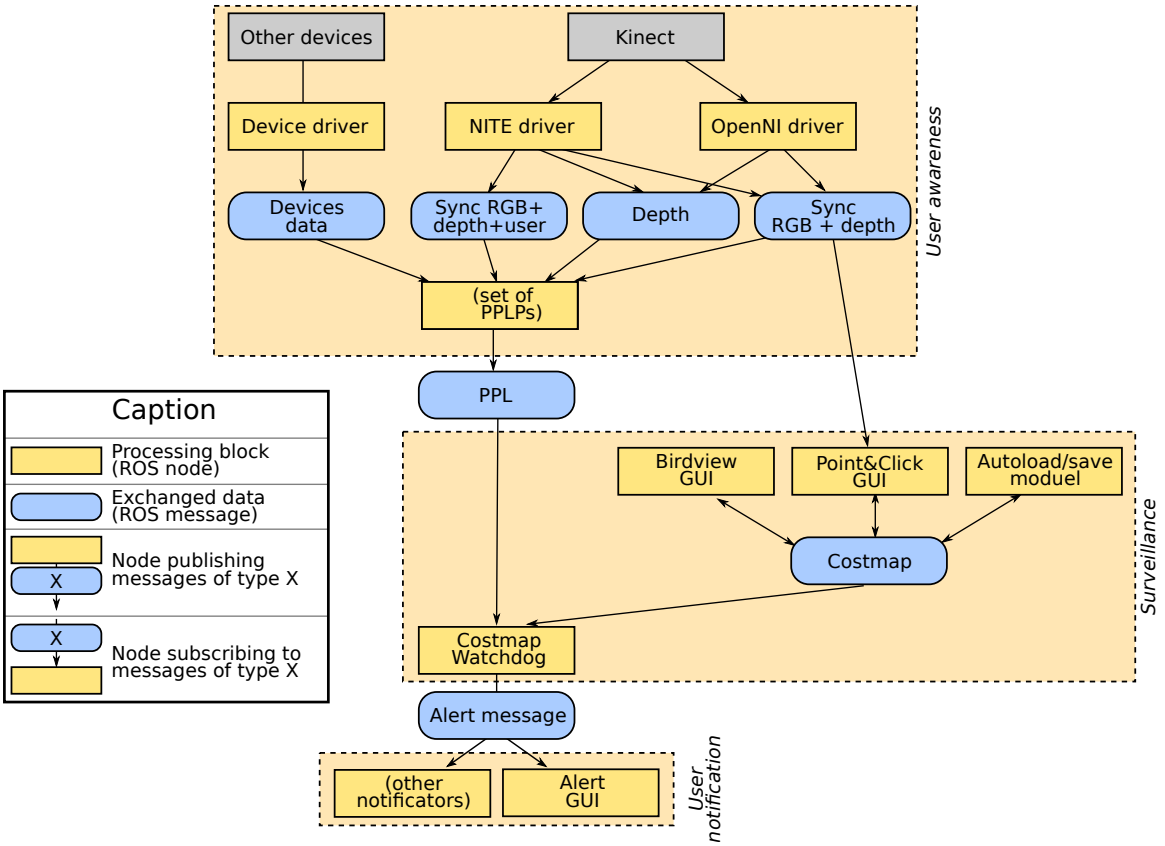


Figure 8.4: *The surveillance pipeline.*

This is why we chose the geometrical definition of dangerous zones, using a so-called **Costmap**. The concept of costmap comes from geometrical autonomous navigation (see for instance [Wooden et al., 2010]). It consists of dividing the surroundings of the robot into an infinite grid of squared cells of fixed width, centered on the robot. This width, in meters, is bound with the resolution of the map: the smaller the width, the more accurate the representation of the environment. Each cell has an associated cost (hence the name): this cost is a floating point number that represents the penalty for the robot to enter in it. For instance, a cell distant from all obstacles will have a cost close to zero, a wall cannot be crossed by the robot and will result in an infinite cost, while zones close from walls have a high cost as they represent a collision danger for the robot.

In our case, the costmap will represent the forbidden zones: if a cell has a cost of zero, then it can be freely entered by the patient, if its cost is positive, it is forbidden. The patient should not enter it, if she does, then the caretaker must be warned.

Initially, the costmap is empty: all cells are allowed. The caretaker, who has the knowledge of the house and who knows what are the dangerous zones, is in charge of indicating the dangerous zones in the costmap. Three tools were developed for that goal:

- **a birdview Graphical User Interface (GUI)** that displays the state of the grid cells. It gives a representation of the world as seen from above. In other words, the 3D world is projected to two-dimensional (2D) by dismissing the vertical component. By clicking on a given cell on the grid, the caretaker toggles the state of that cell between forbidden and allowed. Because it presents the world seen from above and does not overlay any information of the real world (the furniture and other visible objects are not drawn), it can be challenging for the caretaker to get her bearings. A sample is visible in Figure 8.5 (a).
- **a "Point&Click"GUI** that shows the camera frame. By clicking on the a pixel on the image, the user can toggle the state of the corresponding floor cell (vertical projection of the pixel to the floor). It is more natural to use for the caretakers, as they can see what the camera sees. However, it is computationally more expensive as it requires subscribing to the Red Green Blue (RGB) and depth stream of the camera (when the caretaker clicks on a 2D pixel, the pinhole model of the camera is used to reproject this 2D point in a 3D ray, then the depth value at that pixel converts the clicked point into a 3D point). A sample is visible in Figure 8.5 (b).
- Finally, the **autosave/autoload tool** saves the costmap in a file on the hard drive when the surveillance system is shutdown, and loads it when the system is turned on again. It ensures no data is lost on a reboot of the surveillance system. It has no visible interface and does not require any action of the user.

Note that these three modules exchange the built costmap in real time, as they subscribe and publish on the same costmap Robot Operating System (ROS) *topic* [3]. This means the caretaker can for instance create the costmap with a the birdview GUI, then continue editing it with the "Point&Click"GUI.

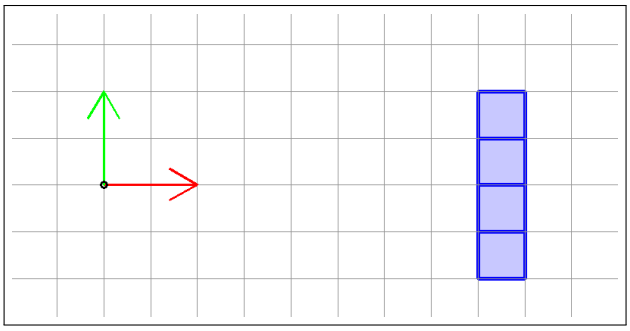
These three tools enable the editing of the costmap in a given frame of coordinates, most often the camera frame. However, in the case of a moving robot, we can obtain a consistent costmap, even if the robot moves, by generating the costmap in a static frame. In that case, a conversion between the static and the robot frame is performed, for instance via odometry or SLAM.

User detection and tracking At the same time as the costmap building, we need to detect if the patient around the robot is entering a forbidden zone. This is where the main contribution of this PhD dissertation comes handful. The `PeoplePoseList` (`PPL`)-based user awareness system we presented before was built for finding and recognizing the users.

Moreover, its flexibility relies in its modularity: the different user detectors can be added or removed according to the hardware specifications of the robot. For instance, the robot prototype we use for the Alzheimer's project has a Kinect, but no laser range finder.

An example architecture that can be used in that case would be: a face detection-based `PPLP` seen in subsection 3.2.3, page 47, good for close user detection; a HOG-based `PPLP` as

[3] ROS architecture was presented in subsubsection 1.4.1.i, page 9.



(a)

(b)

Figure 8.5: *The different tools for the costmap needed in our surveillance application.*
(a): The birdview GUI.
(b): The "Point&Click"GUI.
Note that both GUIs share the costmap data and so represent the same costmap. Both can be used for further editing this shared costmap.

seen in subsection 3.2.4, page 50, that detects users from far away; and a Polar-Perspective Map (PPM)-based `PPLP` seen in subsection 3.2.6, page 56, reasoning on the shape of the clusters. User matching can be made thanks to an distance matcher based on the geometrical distance, and a PersonHistogramSet (PHS) one using the color appearance of the user. This pipeline is drawn in Figure 8.6.

Detection of anomalous activity Once the costmap and the patient detection system are in place, it becomes easy to detect if the patient enters a forbidden zone. The *costmap watchdog* subscribes to both the costmap and to the resulting `PPL` ROS topics, the latter encapsulating both user detection and recognition.

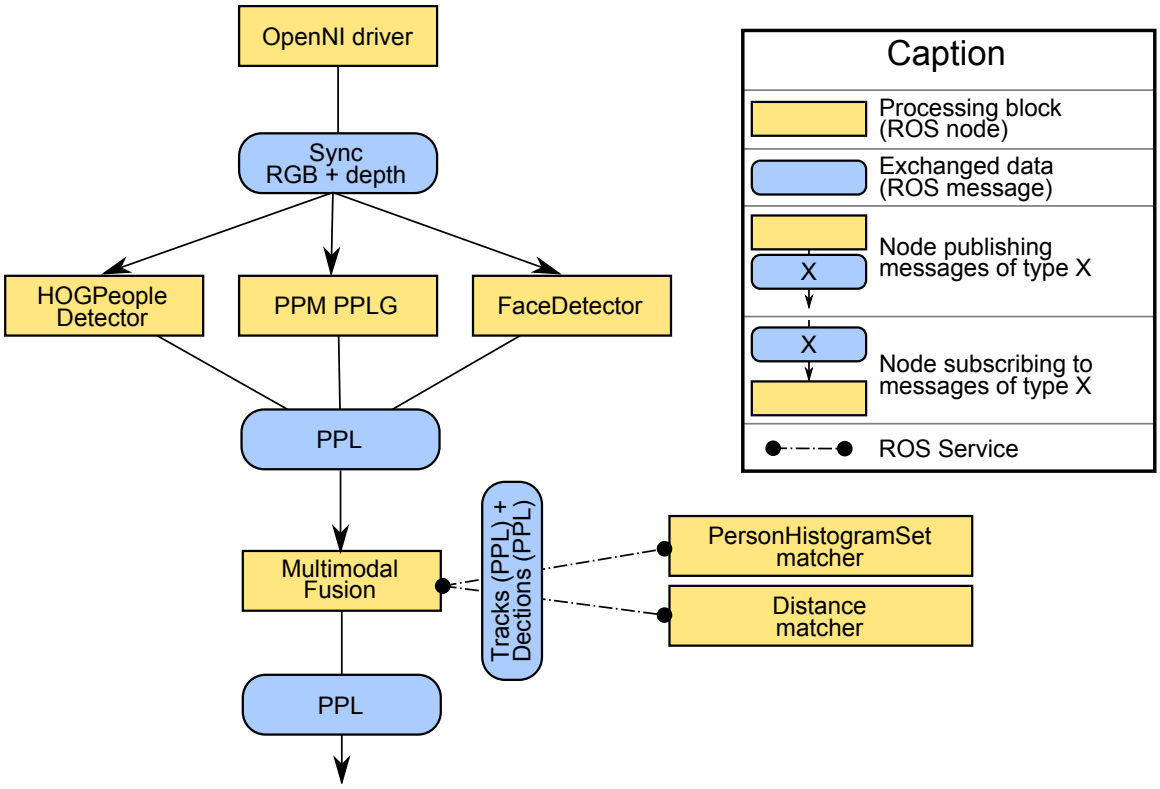


Figure 8.6: *A possible user awareness architecture for the surveillance pipeline.*

If there is a detected user and it is the caretaker, nothing is to be done, as the caretaker is allowed to move in all areas. However, if the user seems to be the patient, the robot checks if she enters a forbidden cell of the costmap. In that case, an **Alert message** is built. It is a data structure containing all the relevant information for the caretaker: the time and date; the type of alert (here, an anomalous activity in the surveillance module); the position of the detected users; the gravity of the warning; and a snapshot of the camera color stream.

Two cases of use are visible in Figure 8.7. This alert message is published by the watchdog for further notification to the caretaker.

Caretaker notification There are many desirable ways to notify the caretaker: a phone call, an audio alarm, an instant email, etc.

These notification modules, as they deal more with software engineering, are out of the scope of this PhD. They can easily be connected to the architecture, as the alert message published by the watchdog contains all the relevant information. As a proof of concept, we wrote a simple alert GUI that generates an alarm sound whose tone depends on the gravity of the alert message, displays the snapshot of the message, and says aloud the content of the message thanks to the robot Emotional-Text-To-Speech (ETTS) system. Note that this simple GUI was

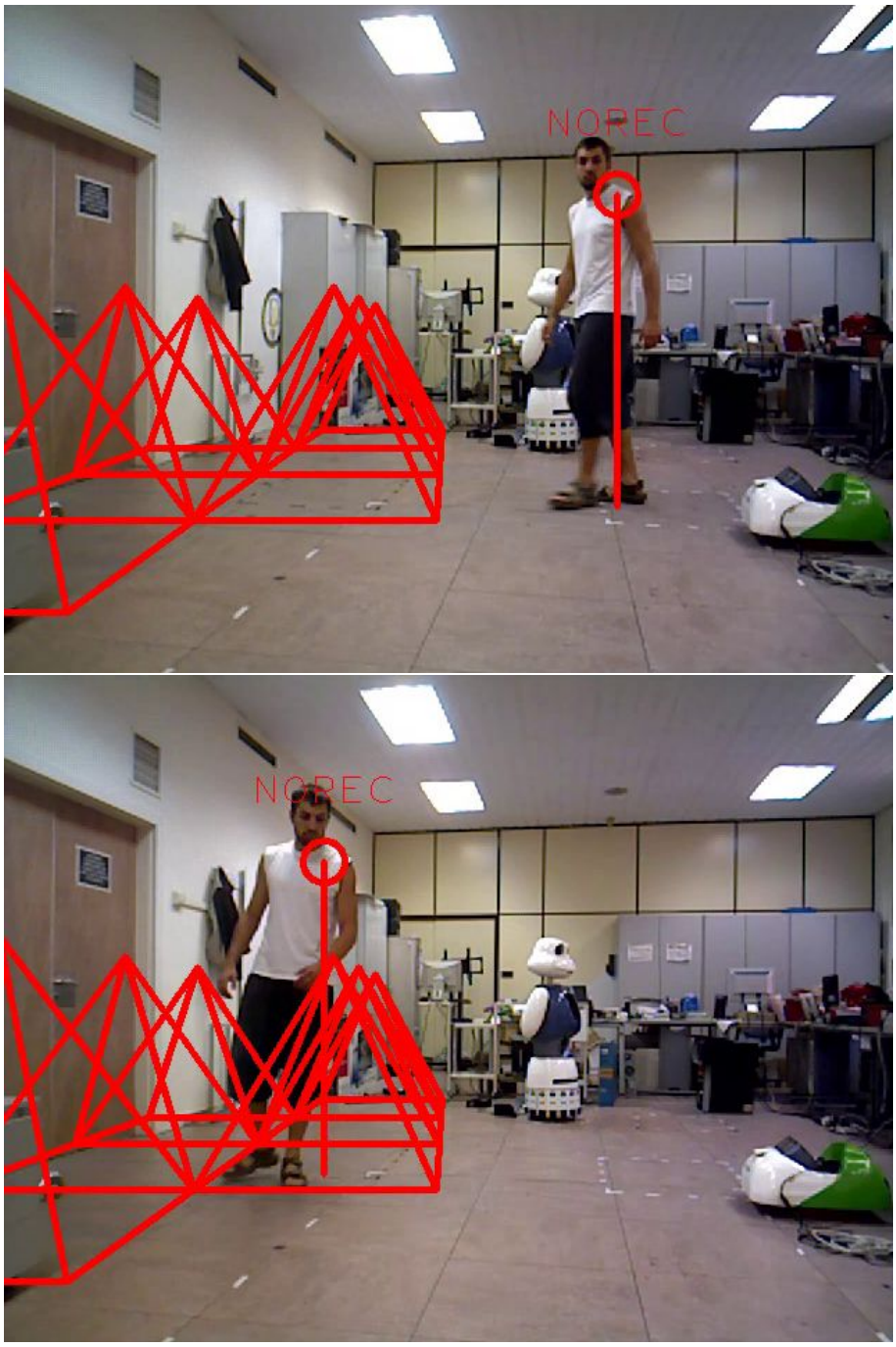


Figure 8.7: *Two cases of use of the costmap watchdog. Above, the user is detected, but he is out of the forbidden cells: no notification is needed. Below, the user comes close to the door, which is marked as forbidden: an alert message is generated.*

written to demonstrate the capabilities of the surveillance block, and that the final end-user application integrates these alert messages smoothly and includes a more adapted automatic Skype call to the caretaker, giving her access to the video stream of the robot. A screenshot of the robot GUI is visible in Figure 8.8.

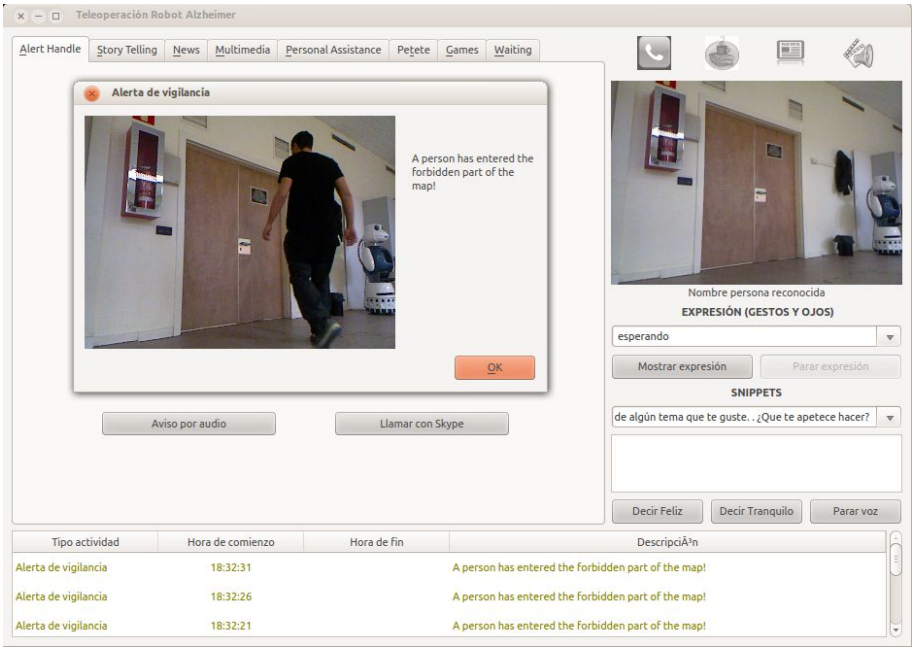


Figure 8.8: *Screenshot of the robot Graphical User Interface integrating the alert messages.*

8.2.3 Testing the surveillance application with users, results and conclusions

The surveillance algorithm detailed above was implemented for MOPI, presented in subsubsection 1.3.1.ii, page 6. MOPI is a car-like robot, which can move around in the apartment of the Alzheimer's patient. To ensure the coherency of the costmap, defined in the camera frame (robot coordinates), but meant to be used in a static frame (world coordinates) the conversion between the camera frame and the static frame is made thanks to the robot odometry.

Note that the internal Kinect of MOPI is about 30 cm high and tilted to 45 degrees, which means the floor is not visible. This can generate confusion among the users when it comes to using the "Point&Click"Graphical User Interface (GUI). A sample is visible in Figure 8.9.

A small testbed was built to test the usefulness of the application. Note that this user test only focuses measuring the usability of the surveillance block for unexperienced users. Other aspects, such as detection accuracy, are not the goal of the test, even though they are measured: they are indeed directly correlated with the accuracy of the user awareness system, presented in the previous parts of this PhD dissertation.

Figure 8.9: *Consequences of the tilted Kinect in MOPI on the "Point&Click"GUI. The floor is not visible, which makes the pyramids indicating the floor look confusing.*

The robot MOPI is placed in front of the door of the test room. Even though the application also works for a robot thanks to the geometrical transformations between frames of coordinates, MOPI is here configured to stay still at that position during the whole experiment with the user. First, the user is asked to play the role of the caretaker. She is asked to mark the main door of the test room as dangerous using first the birdview GUI, then the "Point&Click"GUI.
Then, the user is asked to play the patient role. She must test the surveillance capabilities by trying to escape through the monitored door several times.

In order to measure the so-called **User satisfaction**, i.e. to what extent their interaction with the application is satisfactory, a method called the *Microsoft Desirability Toolkit* is used [Benedek and Miner, 2002]: after the experience, the user has to pick, among a list of about one hundred adjectives, five that according to her resume best the system. These adjectives can be either positive, for example *efficient* or *fast*, or negative, as *hard to use* or *unattractive*. This choice leads to a guided interview where the number of all positive and negative comments said by the user is recorded. The interest of using this adjective-picking method is that it avoids the frequent user bias generated by post-experience rating questionnaires: in these questionnaires, they rate the system high (satisfactory) even if they had a disappointing experience [4]. Some metrics were also used to measure both **User effectiveness** (i.e. can people complete their tasks?): and **User performance**, also called **User efficiency**, (i.e. how long do people take to complete the task?). Note that user satisfaction and its performance are two different concepts, but they are

[4] These reasons are further detailed by [Wiklund et al., 1992]: *In studies such as this one, we have found subjects reluctant to be critical of designs when they are asked to assign a rating to the design. In our usability tests, we see the same phenomenon even when we encourage subjects to be critical. We speculate that the test subjects feel that giving a low rating to a product gives the impression that they are "negative" people, that the ratings reflect negatively on their ability to use computer-based technology, that some of the blame for a product's poor performance falls on them, or that they do not want to hurt the feelings of the person conducting the test.*

only moderately correlated: users tend to prefer the interface they are more at ease with, but not always ([Gelderman, 1998]).

According to different usability experts, a number of *five* non-expert users is enough to detect most of the usability problems of a system. Theses non-expert users were recruited thanks to the **Hallway testing** paradigm [Nielsen, 1993], which recommends to recruit random users who are in the surroundings of the testing area and do not have special knowledge about the tested system, here the surveillance system of the robot MOPI.

Eight users, ages ranging from 26 to 56 (average 33 years old), 38% of which were female, all of them graduate students or professors but with no direct link with the application, participated in the benchmark. Some previous explanations were given to the users about how to use both interfaces. A paper printed screenshot of both interfaces was used to explain them how the interfaces work. Note that the birdview interface print was annotated to help the users to understand the frame of coordinates. A scanned version of the annotated print is in Figure 8.10. These annotations help the user when they later use the interface, and must be taken into account in the results analysis.

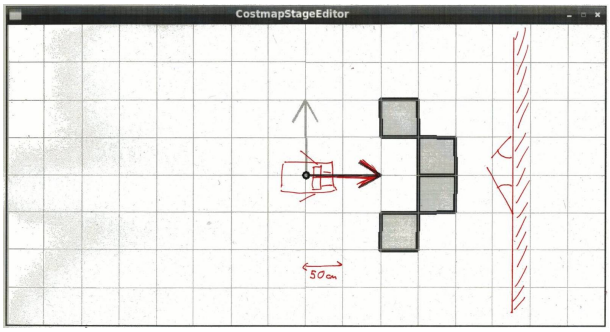


Figure 8.10: *The annotated birdview GUIs used during the user briefing.*

The numerical results of the experience are gathered in Table 8.1. The words picked up by the users for each interface are visible in the words cloud in Figure 8.11. This gives us a first hint of the user satisfaction.

Furthermore, we can estimate the global user efficiency and satisfaction and study any correlation between them. For any variable x, if X is the set of values of x for all users, and μ is the mean of X and σ its standard deviation, $\bar{x}$, the normalized version of x is defined as $\bar{x} = \frac{x-\mu}{\sigma}$. This normalized version has a mean of 0 and a variance of 1. For each interface, we then define the *user effectiveness and performance score* as the average of the normalized ratio of detected escapes, the normalized user clicks numbers, and the normalized time needed to use the interface; and the *user satisfaction score* as the average of the normalized user subjective satisfaction score and the normalized ratio of positive-to-negative comments. We can then check if there is a correlation between performance and satisfaction. The curves are visible in Figure 8.12.

Several conclusions can be drawn.

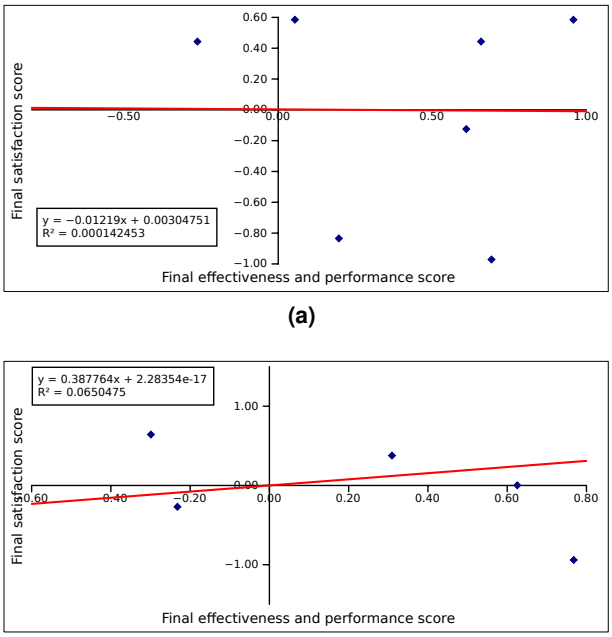
GUI	”Point&Click”	Birdview
Number of users	8	8
Average time for door labeling (seconds)	21.5	21.1
Average click number for door labeling	11.0	7.8
Number of escapes per user for this interface	5	5
Successful escape detection rate	80%	90%
Average subjective satisfaction score (1-7)	3.9	5.8
Average positive comments	2.0	4.1
Average negative comments	3.9	1.5

Table 8.1: *Benchmark results for the surveillance application.*

(a) **(b)**

Figure 8.11: *Clouds of words chosen by the users to describe the surveillance GUIs. Note the fond size is proportional to the amount of times the word was picked by users.*
(a), with the birdview GUI.
(b), with the ”Point&Click”GUI.

First, the birdview GUI, although not faster to use, is felt as better by the users. It includes more stable, efficient and reliable. This can partly be explained by the annotations given in the briefing, but also because there is no reprojection ambiguity, contrarily to the ”Point&Click”GUI. The ”Point&Click”GUI, although it is visually more familiar (it includes the camera stream), turns out to be more confusing for users because the ground is not visible. This problem could be avoided with a tiltable Kinect, which is challenging with the robot actual shape.
There is a correlation between the users satisfaction and the performance of the interfaces: they trust more the birdview one, which misses twice as few users as the ”Point&Click”: 90% against 80%.
However the correlation between the users performance score and their satisfaction is loose:



(a)

(b)

Figure 8.12: *User satisfaction against user effectiveness and performance for surveillance GUIs.*
(a): With the birdview GUI.
(b): With the "Point&Click"GUI.

this conclusion is clearer with the birdview plot, where these variables are independent (the regression curve is almost a constant value).

8.2.4 Conclusions for the surveillance application

User surveillance is a natural extension for the user awareness architecture proposed in this PhD dissertation. Thanks to the modularity of this architecture, it can easily be used in a wide range of robots. The detection rate is high enough, thus validating the main goal of the surveillance application: monitor forbidden zones. In addition, the different interfaces for labeling the forbidden zones make it easier to use for users with different profiles, even for the ones averse to technology.

However, the notification modules lack of user-friendliness: a simple alert sound and verbal warnings are emitted. These simple reactions limit the interaction with the robot. Furthermore, the challenging orientation of the imaging device suggests that a hybrid interface, to be designed, would solve these limitations. It would need to combine the easiness of use of the "Point&Click"interface with the robustness and intuitiveness of the birdview one. We can for instance imagine a birdview-like interface, that would project the three-dimensional (3D) point cloud onto the plane ground. In any case, the usefulness of the proposed application was demonstrated.

8.3 Example of use #2: games

The previous section focused on surveillance as an application of the user awareness architecture presented in this PhD. In this section, we will focus on the possibilities of using it for games adapted to young users interacting with the robot.

Indeed, social robots are often capable of playing games with users, as entertainment is often one of their goals. Several examples were given in chapter 2, page 15. For instance, in [Geiger et al., 2013], the authors stress the interest of a robot as a gaming platform for elderly persons. The goal is to stimulate and keep active the user through board games. In this article, the only game presented is Tic-Tac-Toe, played through the means of a touchscreen located between the robot and the user. The Aibo dog pet-robot was capable of playing tic-tac-toe in [Tira-Thompson et al., 2004] An iCat robot plays chess against schoolchildren in [Pereira et al., 2008] and its emotional state affects the way it plays. Other examples are numerous.

We explored the possibilities of the user awareness architecture proposed in this PhD when applied to games in robotics. This choice is motivated by two reasons: first, in a way similar to surveillance, games demonstrate well the modularity of the proposed system, second, the robots already embedded some simple games in their software: this would make easier the assessment of the advantages and drawbacks of our architecture when applied to these games. Two possible game applications are proposed in this section: tic-tac-toe, in subsection 8.3.1 just below, and Red Light Green Light, in subsection 8.3.2, page 217.

8.3.1 Tic-tac-toe

tic-tac-toe is a board game made for two players. It is played on a square grid of 3×3 cells. One player has cross-shaped pawns, the other circles. Turn by turn, each player locates a pawn in a free cell of the grid. The goal of the game is to align three of his own pawns (horizontally, vertically or in diagonal) before the other does.

Because of the simplicity of the rules and the simple hardware needed to play, it is a popular game among children. It has been used several times in robotics too, as it can be a simple application for measuring Human-Robot Interaction (HRI) metrics ([Johnson et al., 2008, Johnson et al., 2006, Dewdney, 1989]).

Previous work In [Ramey, 2011], we presented how we turned the robot Maggie into an autonomous tic-tac-toe player. [5]. The user plays with Maggie in a very natural way: either using cardboard pawns, or drawing on a piece of paper the crosses and circles. This scenario is illustrated in Figure 8.13.

The interaction with the robot relies on the touch sensors present on the robot. The user touches these sensors when she want to indicate the robot that she has played.

[5] A full description of the robot is available in the subsubsection 1.3.1.i, page 5

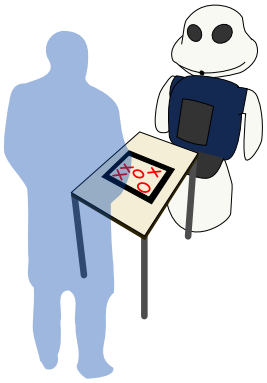


Figure 8.13: *The spatial configuration for playing tic-tac-toe with Maggie.*

To help the robot finding where the proper tic-tac-toe game is in the picture, we invented the concept of **Playzone**. It consists of a black thick square that is located so as to make a tight frame for the game. It can be a rigid frame, made for instance of plastic, or drawn by the user around the tic-tac-toe game to help the robot.
The detection of the playzone marker in the image is made of a few steps.

1. The input image is thresholded thanks to an adaptative thresholding of the color image [Jain, 1977].
2. Dark connected components are quickly retrieved by a fast labelling algorithm based on *Disjoint Sets*, already seen in section 3.2.6, page 56.
3. The closest connected component is obtained thanks to a pattern matching distance called *Modified Hausdorff Distance*, seen in section 7.2.5.iv, page 182.
4. The closest component is approximated by a 4-edges polygon.
5. Finally, the content of this closest connected component is rectified into a birdview perspective, see [Semple and Kneebone, 1998].

The image processing is made much easier by the mean of the playzone marker: for the tic-tac-toe for instance, we can divide the rectified playzone into a 3×3 grid, and determine if the content of each cell is a cross or a round.

Limitations of the previous work While the interaction flow we obtained at the time of writing of [Ramey, 2011] was smooth and most users were satisfied with their experience with the robot, the understanding of the environment by the robot was very poor.

Indeed, the mechanism of markers (playzones) was of a great help to find and understand the game, but the robot had no concept of the users in its surroundings. Even more, it would play as long as the capacitive touch sensors were activated, even though the user was gone.

Contemporary HRI focuses on the contrary on the robot engaging the user during their interaction: see for instance [Pereira et al., 2008] for some early experiments between children and a iCat robot chess player, or [Sanghvi and Castellano, 2011] for a series of metrics for determining the engagement of the user according to the way she stands and sits.

These studies underline how important it is for the engagement of the human user that the robot companion gives feedback according to the state of the game, but also to the affect of the user. This feature was absent in [Ramey, 2011].

Towards a dynamic tic-tac-toe: use of `PeoplePoseList` (`PPL`) We presented in the previous chapters how a robust user recognition and tracking system was built. The information flow is based on the building and transmitting of a special kind of message, called `PeoplePoseList` (`PPL`) (defined in subsection 3.2.2, page 41).

These messages encapsulate all the useful information for a personalized interaction: how many users are around the robot, where they are located, and their most probable identity.

It is then easy to build a user-aware tic-tac-toe game: by subscribing to the `PPL` topic, it can detect who the users are and how engaged they are in the game.

The personalization of the game is multiple:

- at the beginning of the game, the robot uniquely identifies the player it will compete against. It then can greet him or her.
- during the game, the robot checks periodically where the user is. If the user seems to walk away from the game, it can ask her about her intentions to resume the game.

Integration and future works The bridge between the previously existing version of the tic-tac-toe games presented in [Ramey, 2011] and the work presented in this PhD was implemented and integrated into the robots of the lab.

It is actually a re-writing of the original *skill* present in the robots, that substitutes the turn-by-turn playing with a more reactive behavior according to the motion of the user.

However, due to a lack of time in the final phase of this PhD, no experimental tests could be integrated to measure the improvement of the HRI with this new version. This work will hopefully be realized as an application of the PhD. Note that thanks to the open-source nature of most of the code associated with this PhD, it can also be reused and adapted by other researchers.

8.3.2 Future works: Red Light Green Light

Another application of the user awareness architecture is under development at the time of writing this dissertation: the Red Light Green Light game.

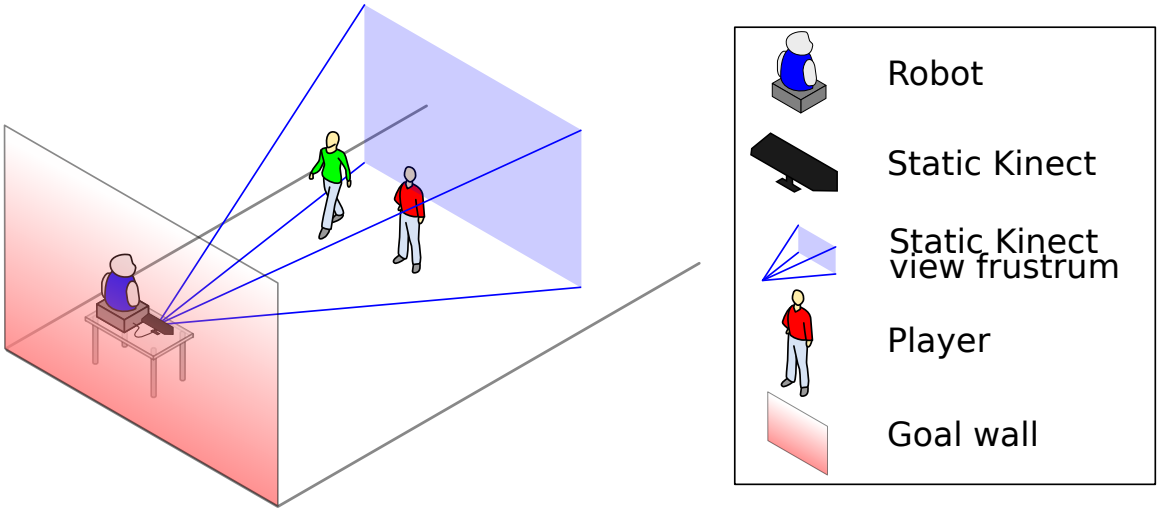
Goal: This game has been popular for generations among children around the world (Spanish name: *escondite inglés*, French name: *1, 2, 3, Soleil*). The rules are simple: a group of children stand in line several meters away from a wall. Their goal is to reach and touch the wall before the others. Another child stands against the wall, she is the ”keeper”. When she stands facing the wall, the other players can move freely. However, when she turns around and faces the other players, they have to freeze themselves in the position they had, and any player that is caught moving while she observes is penalized: she has to go backwards, hence further away from the target wall, back to the start line. The first player to reach the wall without getting caught wins the round, and becomes keeper. Before the keeper turns around to see the other players, she warns them by saying a catchphrase, most often the name of the game.

Implementing Red Light Green Light on a robotic platform presents a lot of advantages. Thanks to its popularity among children for generations, most children that interact with our robots already know the rules, and hence are ready to play. The game requires the child to have both physical strength (speed) and control of her own body, which is good for her development. The game does not need a lot of verbal interaction, once the game has started, which eases the Human-Robot Interaction for shy users, such as small children. Because they are based on the children motion, consecutive rounds of the game tend to be less repetitive than static games, such as the tic-tac-toe presented before, and hence make the interaction last longer before the user is bored. And finally, and maybe most important, Red Light Green Light requires by definition powerful user detection and tracking capabilities, which is what provides our architecture. Without such a mechanism, teaching the robot to play Red Light Green Light is impossible. On the other hand, as seen before, user awareness only *improved* the tic-tac-toe game.

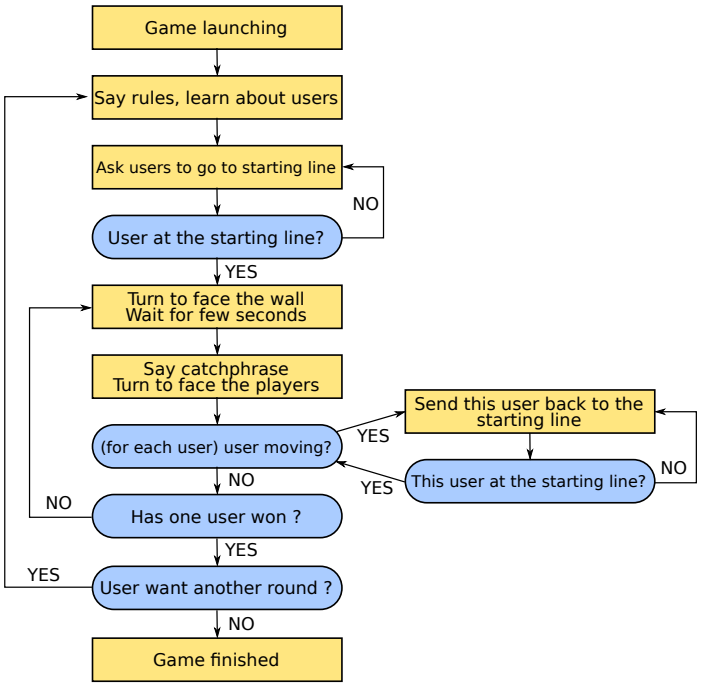
Method: User awareness is required mainly if the robot is the keeper. Furthermore, giving the robot the role of keeper solves all issues of robot motion and control: the only motion of the keeper is turning in place. This is why we focused on implementing the game with the robot being the keeper.

Interaction flow: here is how we plan the interaction flow. If the robot is static (Flori), it has to be put against the wall at the beginning of the game. If it is mobile (MOPI, Maggie), it can start the game from anywhere, then moves against the wall. First, the robot announces the beginning of the game, and sums up the rules, facing the users. It asks the user to go to the virtual starting line (a few meters away from the robot) and checks the user complies with it. Then the game starts: the robot turns around against the wall, wait for some time, then say the catchphrase and turn around. If one of the users is caught moving, the robot asks her to move backwards. The setup of the game is visible in Figure 8.14.

Implementation: The user architecture proposed in this PhD supplies all the data needed for an easy implementation of the Red Light Green Light game: thanks to it, the users are detected and recognized. The `PeoplePoseLists` (`PPLs`) messages that are emitted not only contain the



(a)



(b)

Figure 8.14: *The game design for the Red Light Green Light game.*
(a): The perception of the environment is made by a static device, here the Kinect, that will not be disturbed by the motion of the robot.
(b): The interaction flow of the game. It more or less follows the way two children would play together.

identity and position of the different users, but also their user mask, i.e. a binary image that indicates all pixels belonging to the user. With this data, we can easily determine if a user is moving: by matching the shape of this user from one frame to another, its relative motion can be estimated. Then, if this motion is over an experimental threshold (for instance, if one of the points of the shape has moved of more than 20 centimeters), then the user is set as moving and sent back to the start line.

Because the robot is moving (facing the wall, engagement gestures, etc.), the camera embedded in it is often in motion and suffers motion blur: especially during the rotation phases when the robot turns around, the images are blurry and the user detection becomes challenging. For this reason, we use an external sensing device, an external Microsoft Kinect plugged into the robot but that lies next to it and remains static whatever the gestures of the robot.

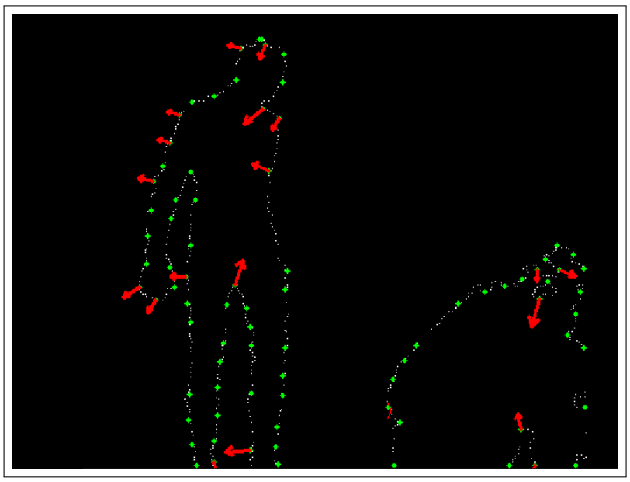
State and results: The Red Light Green Light game is a good application for the user architecture proposed in this PhD. It is in development at the time of writing this dissertation, and will be used as a support for testing the usability of the user architecture by other users than the author. For this goal, an extra effort concerning the documentation and integration of the pipeline has been made. The motion detection components have already been developed for a previous project and were successfully integrated into the Red Light Green Light pipeline. A sample is visible in Figure 8.15. We hope to develop this work into a useful testbed for measuring other Human-Robot Interaction (HRI) metrics.

8.3.3 Conclusions for games

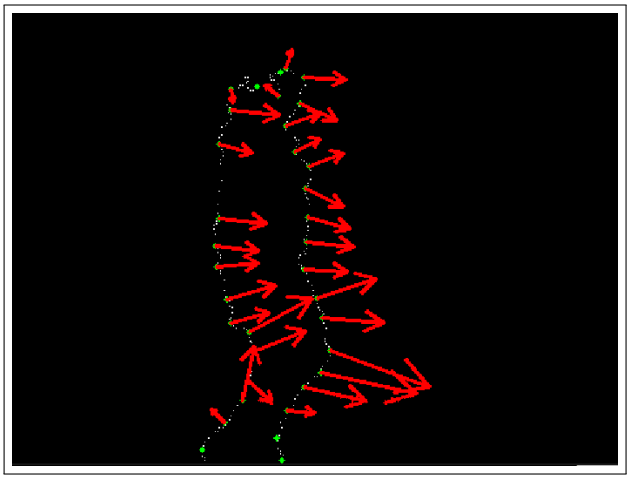
Games represent a important and popular field for social robots, and they make a good application of the user awareness architecture presented in this PhD. Two end-user games are under development at the time of writing: tic-tac-toe and Red Light Green Light.

The former is actually an upgrade of an existing version of the game that was developed during the Master Thesis of the author on the robots. The user awareness enables a customization of the game play and the Human-Robot Interaction (HRI) to engage the player more efficiently. The latter is a game that was developed from scratch. It requires a precise user detection and recognition system to measure the motion of the players, which would have been a difficult challenge without our work.

As said, both games are still under development and already have some prototype versions ready. We hope in a near future that they will be ready for deeper experiments with users, and that they will make a good testbed for HRI and widen the range of possibilities the robots propose.



(a)



(b)

Figure 8.15: *Sample pictures of motion detection in a user mask for Red Light Green Light. The white points correspond to the edge of the user mask. Green points correspond to a polygonal approximation of this white edge, and red arrows represent the motion of each of the green points, both in direction and amount: the bigger the acceleration, the greater the arrow.*
Picture (a) corresponds to two still users, picture (b) corresponds to a user walking to the right direction.

CHAPTER 9

Conclusions, main contributions and future developments

Social robots need to be aware of the users around them. Without this key skill, they cannot offer a meaningful interaction with their users.

This PhD dissertation tackled the problem of giving user awareness to social robots. Many techniques for detecting, recognizing and tracking users are already published by other authors, and I chose to reuse and improve the most relevant ones.

User awareness has been divided in two sub-tasks: user detection on the one hand, user recognition on the other hand. The former consists of finding the users around the robot thanks to the sensors data. User recognition aims at recognizing these detections of users against previous ones.

User detection User detection is mainly based on image processing and computer vision techniques. In this field, this PhD contributions have consisted of designing a common data message, called `PeoplePoseList` (`PPL`), that enables the standardization of all the user detectors that were integrated; and selecting the best algorithms to improve them and adapt them to this data message.
A `PPL` message is structured as a Robot Operating System (ROS) `msg` data file. ROS is a communication middleware, originally described in [Quigley et al., 2009] .
The various user detectors are shaped as ROS processes, capable of creating `PPL`s messages. They are then called `PeoplePoseList` Publishers (`PPLP`s). A `PPLP` is structured as a ROS *node*, subscribing to the data topics it needs, for instance the color image stream, and publishing `PPL` messages for every new data received. Thanks to the ROS architecture, various `PPLP`s can run

in parallel, even if they are written in different programming languages (among C++, Python, and Perl) and running on distributed computers.

Most of the PPLPs I created are based on existing image processing techniques but were improved in different ways and their performance was extensively measured through the use of different datasets. Face detection is based on the Viola-Jones two-dimensional (2D) face detector [Viola and Jones, 2004] but uses the depth image to remove outliers. Similarly, the Histogram of Oriented Gradients (HOG) PPLP discards false positives obtained by the original 2D algorithm [Dalal and Triggs, 2005] thanks to the depth shape of the detections. A driver using the NiTE middleware, originally written for the Kinect Software Development Kit (SDK) [Latta and Tsunoda, 2009], was written so as to share its data with the rest of the ROS architecture. A technique originally based on stereo-vision for autonomous cars, called Polar-Perspective Map (PPM) ([Howard and Matthies, 2007]), was also integrated and benchmarked. State-of-the-art algorithms in object detection for grasping by robotic arms have been used too: we can indeed consider users standing on the floor as big objects standing on a planar surface. And finally, we experimented techniques that are not based on image processing but instead rely on other sensors: a PPLP based on the microphone activity uses Voice Activity Detection and sound source localization, while the leg-pattern detector analyzes the horizontal 2D scans of the laser range finder to locate the users around the robot.

All PPLPs do not have the same accuracy or computational needs. The NiTE, tabletop and PPM PPLPs have both excellent accuracy and recall on uncluttered scenes, but get easily fooled by human-like objects as a coat rack. On the other hand, the HOG detector is based on the analysis of the color image, and its accuracy is reasonably high to discriminate between real users and these user-like objects, but it needs to see the entire person from head to feet to detect it. Note that the face detection-based PPLP is especially appropriate for close user detection, while it performs poorly when the user is further.

User recognition The second sub-problem of user awareness is user recognition. We focused especially on determining if two successive user detections correspond to the same physical user.

Our contributions to user recognition follow a structure similar to user detection: first, a common design was designed for all the algorithms that could perform user recognition. These algorithms are called PeoplePoseList Matchers (PPLMs) and are in fact structured as ROS nodes. They offer a ROS *service* that matches reference users against detected ones (services can be seen as a kind of callable inter-process function or web-service). This service takes as a argument a reference PPL, corresponding to the tracked users by the system, and a detected PPL. They return the cost matrix, i.e., for each pair of (reference PeoplePose (PP), detected PP), the cost of associating one to the other. The higher the cost, the more different they are.

This sub-problem is also a point that has been studied by others before this PhD. Some of the techniques I propose reuse and improve these previous works. Most of the PPLMs that I developed handle with physical traits of the user. Face recognition, based on different algorithms

such as Eigenfaces ([Turk and Pentland, 1991]) , Fisherfaces ([Belhumeur, 1997]) , and Local Binary Patterns Histograms ([Ahonen et al., 2004]) give reliable hints of the user identity. Two algorithms I developed handle with the user height and the shape of her breast. Note that the user height can be used as a real recognizer and estimates the identity of the user, while the breast-based PPLM only estimates the gender of the user.

Another class of PPLMs focuses on users' adhered human characteristics, and more especially the color of her clothes. The descriptor is called a PersonHistogramSet (PHS), a set of color histograms structured according to the part of the body they describe, and the matching is made thanks to sums of histograms distances.

Finally, techniques from other fields can be used. Fiducial markers, such as ARToolkit tags ([Kato and Billinghurst, 1999]) , give a very reliable estimation of the user pose, although the process of wearing a tag is somewhat cumbersome.

Multimodal fusion and tracking The first block of the architecture, the PPLP, analyzes the data of the sensors and published possible sets of detections of users found in it. The second block, the PPLM, evaluates to what extent one of these detections corresponds to a user seen before. The functionality that we now need is combining these detections across time using the matching results to form *tracks*, or trajectories, of each real user. In other words, we need to merge together the outputs of different algorithms: this process is called *data fusion*.

Two mains classes of algorithms are available for this task: particle filters ([Gordon et al., 1993]) and Kalman Filters (KFs) ([Bishop and Welch, 2001]). They both give an estimate of the so-called *state* of the system, given a series of *measurements* over time. I chose to focus on the second one, for its important presence in related work and availability. Actually, KFs are made for estimating the state of linear systems (i.e., there is a linear transformation between the state and the measurements). For this reason, we used an extension of KFs for non-linear problems called Unscented Kalman Filter (UKF) ([Wan and Merwe, 2000, Julier and Uhlmann, 2004]) .

In our case, the state is the position and orientation of the user, and the measurements are the PPs given by the different publishers. However, KF (and UKF) needs to know a-priori the dimension of the state. The number of users being unknown a-priori, we cannot use one unique filter to estimate all the user states at the same time. Consequently, there is actually one filter per detected user, i.e. per track. During a phase called *gating*, the detections that are too far away from existing tracks are kept in memory. If numerous PPs accumulate at a given point, it will trigger the creation of a new track at that point. The matching between the set of KFs (also called *tracks*) and the detections is made through a process called *assignment* ([Jonker and Volgenant, 1987, Rong Li and Bar-Shalom, 1996]), which find the combination of matches that minimizes the cost generated by each of these matches. This assignment is based on the so-called cost matrix, describing the likeliness of each track with each detection, estimated by each of the PPLMs running.

This structure is expandable: a new PPLM, whatever its algorithm and input sensors are, will generate a cost matrix that will be handled by the generic algorithm of the data fusion node. It is

modular: a PPLM ca be started or shut down dynamically, which starts or stops the associated ROS service, and so is seamlessly used or not by the data fusion node.

We have systematically run extensive benchmarks on each of the sub-components of our user architecture, and on the architecture itself. Standard academic datasets were used, such as the *DGait* dataset ([Igual et al., 2013]) or the *Kinect Tracking Precision* dataset ([Munaro et al., 2012, Munaro and Menegatti, 2014]). However, these dataset do not offer data that corresponds to typical HRI scenarios, so we recorded our own dataset, named the *RoboticsLab People Dataset*. This data is freely available for other researchers to use and compare their results against ours. These extensive benchmarks have underlined that, thanks to multimodal fusion, the use awareness architecture supplies an accurate local mapping of the users around the robot.

Applications The described system provides user awareness to the robot: it indicates how many users are around it, where they are and possibly who they are. This information of key importance for a meaningful HRI, but is not a end-user application: it is thought to be used by higher-level applications. Although it is not the scope of the work presented in this PhD, I developed two end-user applications that make use of our user architecture.

- **Surveillance:** the robots of the RoboticsLab are used in a project dealing with helping caretakers of Alzheimer's patients. One of the tasks that must be carried out is surveillance of the patients: especially at night, they are expected to remain in a safe area (bedroom) and not wander in potentially dangerous zones (kitchen, outside...). The surveillance application I developed, thanks to the user awareness architecture, detects where the users are, and when they enter a forbidden area, mechanisms are triggered to notify the caretaker. The interest of this application was demonstrated thanks to extensive experiments.
- **Games:** one of the main targeted audiences for personal robots is children. User awareness comes handy when we want to personalize the games. A previous version of the tic-tac-toe ([Ramey, 2011]) was extended using our architecture. Whereas the robot was blind in this previous version and was assuming there was a user in front of it, now it is aware of who is playing against it and what is her behavior.

9.1 Summary of the contributions

Here are the main contributions of the work presented in this PhD to the field of user detection in HRI:

- **A full user awareness architecture that supplies a local map of the users surrounding the robot**. It is based on 3 modules: PeoplePoseList Publisher (PPLP), a common

interface for user detection algorithms, at the same time lightweight and adapted to a wide range of detectors; `PeoplePoseList` Matcher (`PPLM`), a common interface for user recognition algorithms; and finally a data fusion algorithm, based on Unscented Kalman Filters and using the output of the two first modules. This architecture is implemented using the ROS middleware. The map it supplies can be used for end-user applications. The architecture is modular, and was successfully implemented on four robots with very different morphologies.

- The **integration and improvement of several existing user detection algorithms** into the `PeoplePoseList` Publisher (`PPLP`) format: Viola-Jones face detection ([Viola and Jones, 2004]) and HOG ([Dalal and Triggs, 2005]), both improved with false positive removals thanks to depth; NiTE ([Latta and Tsunoda, 2009]); Polar-Perspective Map ([Howard and Matthies, 2007]); and more. Some of these algorithms are fast but with a poor accuracy, while other perform better but require more computation time.

- The **integration and improvement of existing user recognition algorithms** into the `PeoplePoseList` Matcher (`PPLM`) format: namely, face recognition based on three different techniques, and using the improved face detection.

- The **conception, development and extensive benchmark of novel user detection and user recognition algorithms**: height-based user recognition; breast-based user recognition; user recognition based on structured histograms of clothes color (PersonHistogramSet); and voice recognition using Machine Learning techniques. The performance of each of these novel techniques have been benchmarked.

- The realization of useful **end-user applications** using the user awareness architecture: surveillance, a game sample such as tic-tac-toe. The surveillance application has been used for a user study that demonstrates its usefulness and relevance.

- A precise and extended **annotated image dataset for user detection in HRI context**, called RoboticsLab People Dataset. It contains data that we acquired in realistic HRI scenarios and that we manually labeled. It is freely available for other researchers to benchmark their user detection, recognition and tracking algorithms.

9.2 Future works

In this section, we identify possible extensions of the work presented in this PhD dissertation. I hope these goals will be compatible with my future work and will be accomplished promptly.

- The number of possible configurations of `PPLP` and `PPLM` is very high, and some configurations are more adapted to given robots than others. We have demonstrated that some configurations outperform others, most notably the simultaneous use of several `PPLM`s can increase the precision of the user mapping compared with each of these `PPLM`s taken

alone. We wish to establish guidelines concerning the design of configurations offering a good trade-off between speed and accuracy, according to the state of the robot.

- We are currently testing a end-user game were user awareness is a key component: Red Light Green Light. In contrary to the previously presented tic-tac-toe, the rules themselves of this game need to understand where the users are and their motion.

Published contributions

All my published contributions are listed in chronological order in my list of publications, page 242.

Bibliography

[Ahonen et al., 2004] Ahonen, T., Hadid, A., and Pietikäinen, M. (2004). Face recognition with local binary patterns. *Computer Vision-ECCV 2004*. [93, 224]

[Alonso-Martin and Castro-González, 2013] Alonso-Martin, F. and Castro-González, A. (2013). Multidomain Voice Activity Detection during Human-Robot Interaction. *Social Robotics*. [72, 78, 81, 131]

[Alonso-Martín et al., 2013] Alonso-Martín, F., Gorostiza, J. F., Malfaz, M., and Salichs, M. (2013). Multimodal Fusion as Communicative Acts during Human-Robot Interaction. *Cybernetics and Systems*, 44:681–703. [130]

[Alonso-Martin et al., 2011] Alonso-Martin, F., Ramey, A., and Salichs, M. A. (2011). Maggie : el robot traductor. In UPM, editor, *9 Workshop RoboCity2030-II*, number Breazeal 2003, pages 57–73. Robocity 2030, Madrid. [12]

[Anezaki, 2011] Anezaki, T. (2011). Development of a human-tracking robot using QR code recognition. *Frontiers of Computer Vision (FCV), 2011 17th Korea-Japan Joint Workshop on*. [70]

[Austin and Kouzoubov, 2002] Austin, D. and Kouzoubov, K. (2002). Robust, Long Term Navigation of a Mobile Robot. In *Proc. IARP/IEE-RAS Joint Workshop on Technical Challenges for Dependable Robots in Human Environments*. [18]

[Barber and Salichs, 2002] Barber, R. and Salichs, M. (2002). A new human based architecture for intelligent autonomous robots. In Asama, H. and Inoue, H., editors, *Proceedings of The 4th IFAC Symposium on Intelligent Autonomous Vehicles*, pages 85–90. Elsevier. [5, 12]

[Belhumeur, 1997] Belhumeur, P. (1997). Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. [92, 93, 224]

[Bellotto and Hu, 2009] Bellotto, N. and Hu, H. (2009). Multisensor-based human detection and tracking for mobile service robots. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*. [xviii, 32, 73, 74, 81, 85, 145]

[Benedek and Miner, 2002] Benedek, J. and Miner, T. (2002). Measuring Desirability: New Methods for Evaluating Desirability in a Usability Lab Setting. In *Proceedings of Usability Professionals Association*, volume 2003, pages 8–12. [211]

[Benesty et al., 2008] Benesty, J., Chen, J., and Huang, Y. (2008). *Microphone Array Signal Processing*, volume 1. [73]

[Berliner and Hendel, 2007] Berliner, T. and Hendel, Z. (2007). Modeling Of Humanoid Forms From Depth Maps. *United States Patent Application 20100034457*. [32, 36, 65]

[Bertsekas, 1985] Bertsekas, D. (1985). A distributed asynchronous relaxation algorithm for the assignment problem. In *1985 24th IEEE Conference on Decision and Control*, volume 24, pages 1703–1704. IEEE. [150]

[Bishop and Welch, 2001] Bishop, G. and Welch, G. (2001). An introduction to the kalman filter. *Proc of SIGGRAPH, Course*. [xix, 141, 144, 225]

[Blauth et al., 2012] Blauth, D. A., Minotto, V. P., Jung, C. R., Lee, B., and Kalker, T. (2012). Voice activity detection and speaker localization using audiovisual cues. *Pattern Recognition Letters*, 33(4):373–380. [72]

[Blodow and Rusu, 2009] Blodow, N. and Rusu, R. (2009). Partial view modeling and validation in 3D laser scans for grasping. *Humanoid Robots, 2009*. [59]

[Bloom et al., 2012] Bloom, V., Makris, D., and Argyriou, V. (2012). G3D: A gaming action dataset and real time action recognition evaluation framework. *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*. [45]

[Bouchrika and Nixon, 2006] Bouchrika, I. and Nixon, M. (2006). People detection and recognition using gait for automated visual surveillance. In *IET Conference on Crime and Security*, volume 2006, pages 576–581. IEE. [94, 203]

[Bradski and Kaehler, 2008] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. [47, 50, 61, 99, 101, 111, 201]

[Brandstein and Ward, 2001] Brandstein, M. and Ward, D. (2001). Microphone arrays: signal processing techniques and applications. [73]

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32. [79]

[Burgard et al., 1998] Burgard, W., Cremers, A., and Fox, D. (1998). The interactive museum tour-guide robot. *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial AAAI '98/IAAI '98 intelligence*, pages 11–18. [17]

[Cai et al., 2010] Cai, Y., Laws, J., and Bauernfeind, N. (2010). Design Privacy with Analogia Graph. *IAAI*. [96, 121]

[Canny, 1986] Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698. [39]

[Cano et al., 2002] Cano, P., Batle, E., Kalker, T., and Haitsma, J. (2002). A review of algorithms for audio fingerprinting. *2002 IEEE Workshop on Multimedia Signal Processing.* [130]

[Castellano et al., 2009] Castellano, G., Pereira, A., Leite, I., Paiva, A., and Mcowan, P. W. (2009). Detecting User Engagement with a Robot Companion Using Task and Social Interaction-based Features Interaction scenario. *Proceedings of the 2009 international conference on Multimodal interfaces*, pages 119–125. [16]

[Chang et al., 2010] Chang, S., Ham, S., and Suh, D. (2010). ROHINI: A robotic flower system for intuitive smart home interface. In *Control Automation and Systems (ICCAS), 2010 International Conference on*, pages 1773–1776, Gyeonggi-do, Korea. [57]

[Comport et al., 2003] Comport, A. I., Marchand, E., and Chaumette, F. (2003). A real-time tracker for markerless augmented reality. page 36. [71]

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*. [35, 79, 120]

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on (Volume:1)*. [32, 34, 65, 224, 227]

[Dalmasso et al., 2009] Dalmasso, E., Castaldo, F., Laface, P., Colibro, D., and Vair, C. (2009). Loquendo - Speaker recognition evaluation system. *Acoustics Speech and Signal Processing ICASSP 2009 IEEE International Conference on*, pages 4213–4216. [130]

[Damasio et al., 1982] Damasio, A. R., Damasio, H., and Van Hoesen, G. W. (1982). Prosopagnosia: Anatomic basis and behavioral mechanisms. *Neurology*, 32(4):331–331. [2]

[Darrell et al., 2000] Darrell, T., Gordon, G., Harville, M., and Woodfill, J. (2000). Integrated person tracking using stereo, color, and pattern detection. *International Journal of Computer Vision*. [94, 101, 111, 116]

[Dell'Amico and Toth, 2000] Dell'Amico, M. and Toth, P. (2000). Algorithms and codes for dense assignment problems: the state of the art. *Discrete Applied Mathematics*, 100(1-2):17–48. [150]

[Dewdney, 1989] Dewdney, A. (1989). A Tinkertoy computer that plays tic-tac-toe. [215]

[Deyle et al., 2014] Deyle, T., Reynolds, M. S., and Kemp, C. C. (2014). Finding and navigating to household objects with UHF RFID tags by optimizing RF signal strength. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2579–2586. IEEE. [71]

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. [113]

[Dubuisson and Jain, 1994] Dubuisson, M.-P. and Jain, A. K. (1994). A modified Hausdorff distance for object matching. In *Pattern Recognition, 1994. Vol. 1 - Conference A: Computer Vision Image Processing., Proceedings of the 12th IAPR International Conference on*, volume 1, pages 566 –568 vol.1. [182]

[Duda et al., 1995] Duda, R., Hart, P., and Stork, D. (1995). Pattern Classification and Scene Analysis 2nd ed. [91]

[Dudek et al., 2007] Dudek, G., Sattar, J., and Xu, A. X. A. (2007). A Visual Language for Robot Control and Programming: A Human-Interface Study. *Proceedings 2007 IEEE International Conference on Robotics and Automation*. [32, 70]

[Fiala, 2005a] Fiala, M. (2005a). ARTag, a Fiducial Marker System Using Digital Techniques. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:590–596. [70]

[Fiala, 2005b] Fiala, M. (2005b). Comparing ARTag and ARToolkit Plus fiducial marker systems. In *Haptic Audio Visual Environments and their Applications, 2005. IEEE International Workshop on*, page 6 pp. [70]

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A para- digm for model Fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24:381–395. [40, 62]

[Fisher, 1936] Fisher, R. (1936). The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*. [92]

[Fitzgibbon and Fisher, 1995] Fitzgibbon, A. W. and Fisher, R. B. (1995). A Buyer ' s Guide to Conic Fitting. In *British Machine Vision Conference*, pages 513–522. [112, 119]

[Fortmann, 1983] Fortmann, T. (1983). Sonar tracking of multiple targets using joint probabilistic data association. *Oceanic Engineering, IEEE Journal of*. [148]

[Fox et al., 1997] Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *Robotics & Automation Magazine, IEEE*, 4(1):23–33. [184]

[Galler and Fisher, 1964] Galler, B. A. and Fisher, M. J. (1964). An improved equivalence algorithm. *Commun. ACM*, 7(5):301–303. [57]

[Garage, 2010] Garage, W. (2010). Beer Me, Robot. [25]

[Garcia and Quintana-Domeque, 2007] Garcia, J. and Quintana-Domeque, C. (2007). The evolution of adult height in Europe: a brief note. *Economics and human biology*, 5:340–349. [117, 159]

[Gavrila and Munder, 2006] Gavrila, D. M. and Munder, S. (2006). Multi-cue Pedestrian Detection and Tracking from a Moving Vehicle. *International Journal of Computer Vision*, 73:41–59. [38]

[Geiger et al., 2013] Geiger, J., Leykauf, T., and Rehrl, T. (2013). The Robot ALIAS as a Gaming Platform for Elderly Persons. *Lebensqualität im Wandel von Demografie und Technik - 6. Deutscher AAL-Kongress mit Ausstellung*. [21, 215]

[Gelderman, 1998] Gelderman, M. (1998). The relation between user satisfaction, usage of information systems and performance. *Information & Management*. [211]

[Gockley et al., 2005] Gockley, R., Bruce, A., Forlizzi, J., Michalowski, M., Mundell, A., Rosenthal, S., Sellner, B., Simmons, R., Snipes, K., Schultz, A. C., and Wang, J. (2005). Designing robots for long-term social interaction. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 2199–2204. [19, 24]

[Gonzalez-Pacheco et al., 2011] Gonzalez-Pacheco, V., Ramey, A., Alonso-Martin, F., Castro-Gonzalez, A., and Salichs, M. A. (2011). Maggie: A Social Robot as a Gaming Platform. *International Journal of Social Robotics*, pages 1–11. [21]

[Gordon et al., 1993] Gordon, N. J., Salmond, D. J., and Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *Radar and Signal Processing IEE Proceedings F*, 140(2):107–113. [141, 225]

[Grisettiyz, 2005] Grisettiyz, G. (2005). Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. *Robotics, IEEE Transactions on*, 23(1):34–46. [xix, 191]

[Gross et al., 2009] Gross, H. M., Boehme, H., Schroeter, C., Mueller, S., Koenig, A., Einhorn, E., Martin, C., Merten, M., and Bley, A. (2009). TOOMAS: Interactive shopping guide robots in everyday use - Final implementation and experiences from long-term field trials. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, pages 2005–2012. [22]

[Grossman et al., 2000] Grossman, E., Donnelly, M., Price, R., Pickens, D., Morgan, V., Neighbor, G., and Blake, R. (2000). Brain Areas Involved in Perception of Biological Motion. *Journal of Cognitive Neuroscience*, 12(5):711–720. [2]

[Guo and Hall, 1989] Guo, Z. and Hall, R. (1989). Parallel thinning with two-subiteration algorithms. *Communications of the ACM*. [109, 110, 111, 115]

[Haritaoglu et al., 1999] Haritaoglu, I., Harwood, D., and Davis, L. (1999). Hydra: multiple people detection and tracking using silhouettes. In *Proceedings 10th International Conference on Image Analysis and Processing*, pages 280–285. IEEE Comput. Soc. [203]

[Haritaoglu et al., 2000] Haritaoglu, I., Harwood, D., and Davis, L. (2000). W/sup 4/: real-time surveillance of people and their activities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):809–830. [203]

[Howard and Matthies, 2007] Howard, A. and Matthies, L. (2007). Detecting pedestrians with stereo vision: safe operation of autonomous ground vehicles in dynamic environments. In Kaneko, Makoto; Nakamura, Y., editor, *Proceedings of the 13th Int. Symp. of Robotics Research*, Hiroshima, Japan. [18, 32, 38, 56, 65, 224, 227]

[Hunt and Schalk, 1996] Hunt, A. and Schalk, T. (1996). Simultaneous voice recognition and verification to allow access to telephone network services. *Acoustical Society of America Journal*. [130]

[Igual et al., 2013] Igual, L., Lapedriza, A., and Borràs, R. (2013). Robust Gait-Based Gender Classification using Depth Cameras. In *Eurasip Journal On Image And Video Processing*. [xix, 45, 54, 62, 82, 94, 117, 121, 126, 127, 226]

[Jain, 1977] Jain, A. K. (1977). A Fast Karhunen-Loeve Transform for Digital Restoration of Images Degraded by White and Colored Noise. *Computers, IEEE Transactions on*, C-26(6):560–571. [216]

[Jang and Chin, 1990] Jang, B. and Chin, R. (1990). Analysis of thinning algorithms using mathematical morphology. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. [109]

[Johnson et al., 2008] Johnson, M. J., Loureiro, R. C. V., and Harwin, W. S. (2008). Collaborative tele-rehabilitation and robot-mediated therapy for stroke rehabilitation at home or clinic. *Intelligent Service Robotics*, 1:109–121. [215]

[Johnson et al., 2006] Johnson, M. J., Wisneski, K. J., Anderson, J., Nathan, D., and Smith, R. O. (2006). Development of ADLER: The activities of daily living exercise robot. In *Proceedings of the First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, 2006, BioRob 2006*, volume 2006, pages 881–886. [215]

[Jonker and Volgenant, 1987] Jonker, R. and Volgenant, A. (1987). A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*. [150, 151, 225]

[Julier, 2000] Julier, S. (2000). A new method for the nonlinear transformation of means and covariances in filters and estimators. *Automatic Control, IEEE Transactions on*. [146, 147]

[Julier and Uhlmann, 2004] Julier, S. and Uhlmann, J. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*. [141, 145, 225]

[Jung et al., 2012] Jung, J., Dan, B., and An, K. (2012). Real-time human tracking using fusion sensor for home security robot. *Consumer Electronics (ICCE), 2012 IEEE International Conference on*. [32]

[Kanda et al., 2004] Kanda, T., Hirano, T., Eaton, D., and Ishiguro, H. (2004). Interactive Robots as Social Partners and Peer Tutors for Children: A Field Trial. [24]

[Kanda et al., 2009] Kanda, T., Shiomi, M., and Miyashita, Z. (2009). An affective guide robot in a shopping mall. *HRI '09 Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*. [1]

[Kato and Billinghurst, 1999] Kato, H. and Billinghurst, M. (1999). Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Augmented Reality, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on*, pages 85–94. [69, 74, 75, 225]

[Katsuki et al., 2003] Katsuki, R., Ota, J., Tamura, Y., Mizuta, T., Kito, T., Arai, T., Ueyama, T., and Nishiyama, T. (2003). Handling of objects with marks by a robot. *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, 1. [68]

[Kaufman and Lord, 1949] Kaufman, E. L. and Lord, M. W. (1949). The discrimination of visual number. *The American journal of psychology*, 62:498–525. [2]

[Kepski and Kwolek, 2012] Kepski, M. and Kwolek, B. (2012). Human Fall Detection by Mean Shift Combined with Depth Connected Components. *Computer Vision and Graphics*. [32, 203]

[Kollar et al., 2012] Kollar, T., Vedantham, A., Sobel, C., and Chang, C. (2012). A Multi-modal Approach for Natural Human-Robot Interaction. *Social Robotics*. [21]

[Krumm et al., 2000] Krumm, J., Harris, S., and Meyers, B. (2000). Multi-camera multi-person tracking for easyliving. *Visual Surveillance, 2000. Proceedings. Third IEEE International Workshop on*. [101]

[Kuhn, 2006] Kuhn, H. (2006). The Hungarian method for the assignment problem. *Naval research logistics quarterly*. [150]

[Laptev et al., 2008] Laptev, I., Marszałek, M., Schmid, C., and Rozenfeld, B. (2008). Learning realistic human actions from movies. In *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. [197]

[Latta and Tsunoda, 2009] Latta, S. and Tsunoda, K. (2009). Gesture keyboarding. *US Patent App. 12/391,145*. [36, 224, 227]

[Laws and Cai, 2006] Laws, J. and Cai, Y. (2006). *A privacy algorithm for 3d human body scans*. [96, 119, 120, 121]

[Lee et al., 2005] Lee, K., Ho, J., and Kriegman, D. (2005). Acquiring linear subspaces for face recognition under variable lighting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. [103, 107]

[Lee et al., 2012] Lee, M., Forlizzi, J., and Kiesler, S. (2012). Personalization in HRI: A longitudinal field experiment. *Human-Robot Interaction (HRI), 2012 7th ACM/IEEE International Conference on*, pages 319–326. [24]

[Lee et al., 2009] Lee, M., Forlizzi, J., and Rybski, P. (2009). The snackbot: documenting the design of a robot for long-term human-robot interaction. *Human-Robot Interaction (HRI), 2009 4th ACM/IEEE International Conference on*. [1, 24]

[Leite et al., 2012] Leite, I., Castellano, G., Pereira, A., Martinho, C., and Paiva, A. (2012). Modelling empathic behaviour in a robotic game companion for children. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction - HRI '12*, page 367, New York, New York, USA. ACM Press. [32]

[Li and Jr, 1982] Li, K. and Jr, E. W. (1982). Text-independent speaker recognition with short utterances. *The Journal of the Acoustical Society of America*. [133]

[Lienhart and Maydt, 2002] Lienhart, R. and Maydt, J. (2002). An extended set of haar-like features for rapid object detection. *Image Processing. 2002. Proceedings. 2002 International Conference on*. [33]

[Loomis, 1971] Loomis, A. (1971). *Figure Drawing for All Its Worth*. [95]

[Marton et al., 2011] Marton, Z., Goron, L., Rusu, R., and Beetz, M. (2011). Reconstruction and verification of 3D object models for grasping. *Robotics Research*. [59]

[Mikić et al., 2003] Mikić, I., Trivedi, M. M., Hunter, E., and Cosman, P. (2003). Human Body Model Acquisition and Tracking Using Voxel Data. *International Journal of Computer Vision*, 53:199–223. [32]

[Mittal and Davis, 2003] Mittal, A. and Davis, L. S. (2003). M2 tracker: A multi-view approach to segmenting and tracking people in a cluttered scene. *International Journal of Computer Vision*, 51:189–203. [94, 102, 111, 116, 122]

[Moghaddam and Yang, 2002] Moghaddam, B. and Yang, M. (2002). Learning gender with support faces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. [93]

[Montemerlo et al., 2002] Montemerlo, M., Thrun, S., and Whittaker, W. (2002). Conditional particle filters for simultaneous mobile robot localization and people-tracking. *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, 1. [141]

[Mooij, 2010] Mooij, J. M. (2010). lib{DAI}: A Free and Open Source {C++} Library for Discrete Approximate Inference in Graphical Models. *Journal of Machine Learning Research*, 11:2169–2173. [153]

[Mozos et al., 2010] Mozos, O. M., Kurazume, R., and Hasegawa, T. (2010). Multi-part people detection using 2D range data. *International Journal of Social Robotics*, 2:31–40. [74]

[Muñoz Salinas, 2009] Muñoz Salinas, R. (2009). People detection and tracking with multiple stereo cameras using particle filters. *Journal of Visual Communication and Image Representation*. [141]

[Mumm and Mutlu, 2011] Mumm, J. and Mutlu, B. (2011). Human-Robot Proxemics : Physical and Psychological Distancing in Human-Robot Interaction. *Design*, pages 331–338. [164]

[Munaro et al., 2012] Munaro, M., Basso, F., and Menegatti, E. (2012). Tracking people within groups with RGB-D data. *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. [117, 161, 226]

[Munaro and Menegatti, 2014] Munaro, M. and Menegatti, E. (2014). Fast RGB-D people tracking for service robots. *Autonomous Robots*. [117, 161, 226]

[Nagamine and Suzuki, 1964] Nagamine, S. and Suzuki, S. (1964). Anthropometry and body composition of Japanese young men and women. *Human Biology*. [102]

[Nielsen, 1993] Nielsen, J. (1993). A mathematical model of the finding of usability problems. *Proceedings of the INTERACT'93 and CHI'93*, pages 206–213. [212]

[Niinuma et al., 2010] Niinuma, K., Park, U., and Jain, A. K. (2010). Soft Biometric Traits for Continuous User Authentication. *IEEE Transactions on Information Forensics and Security*, 5(4):771–780. [102]

[Oberli et al., 2010] Oberli, C., Torres-Torriti, M., and Landau, D. (2010). Performance evaluation of UHF RFID technologies for real-time passenger recognition in intelligent public transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 11:748–753. [71]

[Olson and Delen, 2008] Olson, D. and Delen, D. (2008). Advanced data mining techniques. [46]

[Pang et al., 2011] Pang, Y., Yuan, Y., Li, X., and Pan, J. (2011). Efficient HOG human detection. *Signal Processing*. [35]

[Pereira et al., 2008] Pereira, A., Martinho, C., Leite, I., and Paiva, A. (2008). iCat, the chess player: the influence of embodiment in the enjoyment of a game. *AAMAS '08 Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. [215, 217]

[Poppe, 2010] Poppe, R. (2010). A survey on vision-based human action recognition. *Image and vision computing*. [197]

[Quigley et al., 2009] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, Kobe, Japan. [9, 41, 223]

[Rae et al., 2013] Rae, I., Takayama, L., and Mutlu, B. (2013). The influence of height in robot-mediated communication. In *ACM/IEEE International Conference on Human-Robot Interaction*, pages 1–8. [94]

[Ramey, 2011] Ramey, A. (2011). Playzones : A robust detector of game boards for playing visual games with robots. In *Libro de Actas del 3er Workshop ROBOT'11: Robótica Experimental*, chapter 7c, pages 626–638. ROBOT'11, Sevilla. [56, 215, 216, 217, 226]

[Ramey, 2012] Ramey, A. (2012). *Playzones: a robust detector of game boards and its application for games with robots*. Master thesis, University Carlos III of Madrid. [57]

[Ramey et al., 2011] Ramey, A., Gonzalez-Pacheco, V., and Salichs, M. A. (2011). Integration of a Low-Cost RGB-D Sensor in a Social Robot for Gesture Recognition. In *Proceedings of the 6th International Conference on HumanRobot Interaction*, HRI '11, pages 229–230, New York, NY, USA. ACM. [5, 32]

[Ramey et al., 2013] Ramey, A., Malfaz, M., and Salichs, M. A. (2013). Fast 3D Cluster-Tracking for a Mobile Robot Using 2D Techniques on Depth Images. *Cybernetics and Systems*. [32]

[Rehg, 1999] Rehg, J. (1999). Vision-based speaker detection using bayesian networks. *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on*. [34]

[Reid, 1979] Reid, D. (1979). An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24. [148]

[Rivas, 2007] Rivas, R. (2007). Robot skill abstraction for ad architecture. *6th IFAC Symposium on Intelligent Autonomous Vehicles*, 47(4):12–13. [12]

[Rong Li and Bar-Shalom, 1996] Rong Li, X. and Bar-Shalom, Y. (1996). Tracking in clutter with nearest neighbor filters: analysis and performance. *IEEE Transactions on Aerospace and Electronic Systems*, 32(3):995–1010. [148, 225]

[Rowley et al., 1998] Rowley, H., Baluja, S., and Kanade, T. (1998). Neural network-based face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. [34]

[Rusu, 2009] Rusu, R. B. (2009). *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany. [51]

[Rusu and Cousins, 2011] Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). *2011 IEEE International Conference on Robotics and Automation*, pages 1–4. [40, 62]

[Saldivar-Piñon, 2012] Saldivar-Piñon, L. (2012). Human Sign Recognition for Robot Manipulation. *Pattern Recognition Lecture Notes in Computer Science*. [101]

[Samaria and Harter, 1994] Samaria, F. and Harter, A. (1994). Parameterisation of a stochastic model for human face identification. *Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*. [103]

[Sanghvi and Castellano, 2011] Sanghvi, J. and Castellano, G. (2011). Automatic analysis of affective postures and body motion to detect engagement with a game companion. *Human-Robot Interaction (HRI), 2011 6th ACM/IEEE International Conference on*. [217]

[Satake et al., 2009] Satake, S., Kanda, T., Glas, D. F., Imai, M., Ishiguro, H., and Hagita, N. (2009). How to approach humans?: strategies for social robots to initiate interaction. *HRI 09 Proceedings of the 4th ACMIEEE international conference on Human robot interaction*, pages 109–116. [23]

[Schiele and Crowley, 1996] Schiele, B. and Crowley, J. (1996). Object recognition using multidimensional receptive field histograms. *Computer Vision - ECCV'96*. [101]

[Schüldt et al., 2004] Schüldt, C., Laptev, I., and Caputo, B. (2004). Recognizing human actions: A local SVM approach. In *Proceedings - International Conference on Pattern Recognition*, volume 3, pages 32–36. [197]

[Schulz et al., 2003] Schulz, D., Fox, D., and Hightower, J. (2003). People tracking with anonymous and ID-sensors using Rao-Blackwellised particle filters. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pages 921–926. [141]

[Semple and Kneebone, 1998] Semple, J. G. and Kneebone, G. T. (1998). *Algebraic projective geometry*. Oxford classic texts in the physical sciences. Clarendon Press. [216]

[Shanno, 1970] Shanno, D. (1970). Conditioning of Quasi-Newton Methods for Function Minimization. *Mathematics of Computing*, 24:647–656. [119]

[Streit and Luginbuhl, 1995] Streit, R. and Luginbuhl, T. (1995). Probabilistic multi-hypothesis tracking. [148]

[Thrun, 2002] Thrun, S. (2002). Particle Filters in Robotics. *Smithsonian*, 1:511–518. [141]

[Thrun et al., 1999] Thrun, S., Bennewitz, M., Burgard, W., Cremers, A., Dellaert, F., Fox, D., Hahnel, D., Rosenberg, C., Roy, N., Schulte, J., and Schulz, D. (1999). MINERVA: a second-generation museum tour-guide robot. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, 3. [17]

[Tian and Yuan, 2010] Tian, Y. and Yuan, S. (2010). Clothes matching for blind and color blind people. *Computers Helping People with Special Needs*. [101]

[Tira-Thompson et al., 2004] Tira-Thompson, E., Halelamien, N., Wales, J., and Touretzky, D. S. (2004). *Tekkotsu: Cognitive robotics on the Sony AIBO*. Citeseer. [215]

[Turk and Pentland, 1991] Turk, M. and Pentland, A. (1991). Eigenfaces for recognition. *Journal of cognitive neuroscience*. [91, 224]

[Varvadoukas et al., 2012] Varvadoukas, T., Giotis, I., and Konstantopoulos, S. (2012). Detecting human patterns in laser range data. In *Frontiers in Artificial Intelligence and Applications*, volume 242, pages 804–809. [74]

[Viola and Jones, 2001] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, volume 1, pages 511–518, Los Alamitos, CA, USA. IEEE Comput. Soc. [32, 33, 65, 104]

[Viola and Jones, 2004] Viola, P. and Jones, M. (2004). Robust real-time face detection. *International journal of computer vision*. [33, 34, 224, 227]

[Vogt, 2002] Vogt, H. (2002). Efficient object identification with passive RFID tags. *Pervasive Computing*. [71]

[Wan and Merwe, 2000] Wan, E. and Merwe, R. V. D. (2000). The unscented Kalman filter for nonlinear estimation. *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. [145, 146, 148, 225]

[Wiklund et al., 1992] Wiklund, M., Thurrott, C., and Dumas, J. (1992). Does the fidelity of software prototypes affect the perception of usability? *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. [211]

[Wolf et al., 2012] Wolf, C., Mille, J., Lombardi, L., and Celiktutan, O. (2012). The liris human activities dataset and the icpr 2012 human activities recognition and localization competition. [45]

[Wooden et al., 2010] Wooden, D., Malchano, M., Blankespoor, K., Howardy, A., Rizzi, A. A., and Raibert, M. (2010). Autonomous navigation for BigDog. In *2010 IEEE International Conference on Robotics and Automation*, pages 4736–4741. IEEE. [204]

[Yin, 1970] Yin, R. K. (1970). Face recognition by brain-injured patients: a dissociable ability? *Neuropsychologia*, 8:395–402. [2]

[Zhang and Suen, 1984] Zhang, T. and Suen, C. (1984). A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*. [109, 110, 111, 115]

[Zhao et al., 2003] Zhao, W., Chellappa, R., Phillips, P. J., and Rosenfeld, A. (2003). Face recognition: A literature survey. *ACM Computing Surveys*, 35(4):399–458. [91, 93]

[Zhou, 2004] Zhou, S. (2004). Visual tracking and recognition using appearance-adaptive models in particle filters. *Image Processing, IEEE Transactions on*. [141]

[Zhu and Yeh, 2006] Zhu, Q. and Yeh, M. (2006). Fast human detection using a cascade of histograms of oriented gradients. *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. [35]

List of Publications

[1] Fernando Alonso-Martin, Arnaud Ramey, and Miguel Ángel Salichs. Speaker identification using three signal voice domains during human-robot interaction. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction - HRI '14*, pages 114–115. ACM Press, 2014.

[2] Arnaud Ramey and Miguel Ángel Salichs. Morphological gender recognition by a social robot and privacy concerns. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction - HRI '14*, pages 272–273, New York, New York, USA, March 2014. ACM Press.

[3] Arnaud Ramey, Maria Malfaz, and Miguel Ángel Salichs. Fast 3D Cluster-Tracking for a Mobile Robot Using 2D Techniques on Depth Images. *Cybernetics and Systems*, 2013.

[4] Arnaud Ramey and Miguel Ángel Salichs. Real-time recognition of the gender of users around a social robot: preliminary results. In *11th Workshop RoboCity2030: Social Robots*, pages 978–84–615–6787–4. University Carlos III of Madrid, Madrid, 2013.

[5] Arnaud Ramey, Javi F. Gorostiza, and Miguel Ángel Salichs. A Social Robot as an Aloud Reader: Putting together Recognition and Synthesis of Voice and Gestures for HRI Experimentation. In *HRI 2012*, Boston, MA, 2012.

[6] Fernando Alonso-Martin, Arnaud Ramey, and Miguel Ángel Salichs. Maggie : el robot traductor. In UPM, editor, *9 Workshop RoboCity2030-II*, number Breazeal 2003, pages 57–73. Robocity 2030, Madrid, 2011.

[7] Victor Gonzalez-Pacheco, Arnaud Ramey, Fernando Alonso-Martin, Alvaro Castro-Gonzalez, and Miguel Ángel Salichs. Maggie: A Social Robot as a Gaming Platform. *International Journal of Social Robotics*, pages 1–11, September 2011.

[8] Arnaud Ramey. Playzones : A robust detector of game boards for playing visual games with robots. In *Libro de Actas del 3er Workshop ROBOT'11: Robótica Experimental*, chapter 7c, pages 626–638. ROBOT'11, Sevilla, 2011.

[9] Arnaud Ramey, Victor Gonzalez-Pacheco, and Miguel Ángel Salichs. Integration of a Low-Cost RGB-D Sensor in a Social Robot for Gesture Recognition. In *Proceedings of the 6th International Conference on HumanRobot Interaction*, HRI '11, pages 229–230, New York, NY, USA, 2011. ACM.

[10] F Alonso-Martin, V Gonzalez-Pacheco, A Castro-Gonzalez, Arnaud Ramey, Marta Yébenes, and Miguel A Salichs. Using a Social Robot as a Gaming Platform. In *2nd International Conference on Social Robotics*, pages 30–39. Springer Berlin Heidelberg, 2010.

Abbreviations

AD	Automatic-Deliberative ... 120
COM	Center of Mass ... 54
ETTS	Emotional-Text-To-Speech ... 208
EKF	Extended Kalman Filter ... 142
FAE	Fundacion Alzheimer España ... 203
FOV	Field of View ... 140
GIS	Google Images Search ... 106
GUI	Graphical User Interface ... 210
HOG	Histogram of Oriented Gradients ... 224
HRI	Human-Robot Interaction ... 220
KF	Kalman Filter ... 225
KTP	Kinect Tracking Precision ... 161
LRF	Laser Range Finder ... 23
LBPH	Local Binary Patterns Histograms ... 91
ML	Machine Learning ... 79
PCA	Principal Component Analysis ... 91
PHS	PersonHistogramSet ... 225
`PP`	`PeoplePose` ... 224
`PPL`	`PeoplePoseList` ... 223
`PPLP`	`PeoplePoseList` Publisher ... 223

Index