Design and Implementation of a Human-Acceptable Accompanying Behaviour for a Service Robot

Alvaro Canivell García de Paredes

TRITA-NA-E04166
Design and Implementation of a Human-Acceptable Accompanying Behaviour for a Service Robot

Alvaro Canivell García de Paredes

Master’s Thesis in Computer Science (12 credits) at the School of Electrical Engineering, Royal Institute of Technology year 2004
Supervisor at Nada was Elin Anna Topp
Examiner was Henrik Christensen
Abstract

Robots have been already occupying the industry sector since the 70’s. Nowadays, robots are beginning to be introduced effectively into the service sector as well. Robotic pets, vacuum cleaners, nurses, tour guides have already been making their small place in the service sector.

When service robots are meant to share their working space actively with humans, the capability of dealing with the social constraints of movement are valuable. A service robot that is meant to follow a person in order to perform its service would benefit from this consideration.

The master thesis work presented in this paper was intended to design and implement an accompanying behaviour that could cope with the mentioned social constraints in a service robot. The problem is first delineated, and some related work done in the field is presented. After that, the problem definition is shown to justify the designed architecture, and the whole architecture is depicted. The implementation details of this architecture are later explained. Finally, the overall performance of the implemented design is studied, and conclusions and future improvement are posed.
## List of Figures

2.1 Three robots formation kept while avoiding obstacles ........................................ 6
2.2 Sugar and Kumar’s coordinated object handling robots ................................. 6
2.3 The three boundaries, left, right and furthest, that assure a robust trajectory planner .......................................................... 8
2.4 The selected zones of the image, based on colour matching ............................ 8
2.5 Feyrer and Zell’s stereo-vision system ................................................................. 9
2.6 Desired velocity calculation for maintaining an accompanying behaviour ................... 10
2.7 Accompanying behaviour when dealing with obstacles such as doors ............... 10
2.8 The CERO robot at the IPLab ........................................................................ 11
2.9 MINERVA’s smiling expression (left) and neutral expression (right) ............... 12
2.10 PEARL, the nursebot, in an elderly care facility ............................................. 13

3.1 The general architecture of the system .............................................................. 18
3.2 The architecture of the tracker module ............................................................ 20
3.3 The architecture of the follower module .......................................................... 22
3.4 The definition of the desired position of the robot ............................................ 23
3.5 The architecture of the main module ............................................................... 25
3.6 The module coordinator inside the main module .............................................. 27
3.7 The state sequencer module ........................................................................... 29
3.8 The architecture of the robot interface ............................................................ 30

4.1 Classic discrete time control loop ..................................................................... 32
4.2 Communication rate schema ............................................................................ 35
4.3 Region setting based on the Kalman Filter prediction ...................................... 37
4.4 A group of points detected as a line in a fragment of a laser range data set ....... 38
4.5 Leg hypotheses on a fragment of a real scan data set ....................................... 39
4.6 The complete laser range data scan and the position of the person legs .......... 40
4.7 ......................................................................................................................... 43
4.8 Transformation process of a point according to a fictitious rotation and translation of the robot .............................................................. 45
4.9 Possible sets of ($\alpha, \rho$) values according to the context identifier ............... 47
4.10 Absolute velocity calculation made in the velocity estimator component ........................................ 48

4.11 The two different definitions for the desired position ($\vec{a}$) of the person, according to the state of the robot. The figure on the left represents the $\vec{a}$ defined in the state approach, and the figure on the right the $\vec{a}$ defined in the follow state. ........................................ 49

4.12 Person’s position and velocity thresholds used to determine a probable visibility loss situation ........................................ 50

4.13 The threshold on the angle difference between person and desired position used to determine a probable visibility loss situation ........................................ 50

4.14 Relation between the module of the desired position vector and the velocity associated to this factor ........................................ 51

4.15 Internal behaviour of the component set desired velocity ........................................ 52

4.16 Internal behaviour of the component set desired velocity ........................................ 54

4.17 Five different regions where obstacles can be located determine how to react to them ........................................ 55

4.18 The three different regions where obstacles can be detected by the obstacle detection component ........................................ 56

4.19 A screenshot from the interface offered by the screen interface component ........................................ 58

4.20 The state machine inside the state sequencer module ........................................ 60

4.21 The architecture of the robot interface ........................................ 64

5.1 Person’s position and velocity estimations, in the $x$ axis ........................................ 67

5.2 Person’s position and velocity estimations, in the $y$ axis ........................................ 67

5.3 Evolution of the estimation process when the person is just found ........................................ 68

5.4 Evolution of the estimation process when the person is lost and later found again ........................................ 69

5.5 The estimated number of misses in 100 frames for each subject tested, and the mean value for all the subjects ........................................ 69

5.6 The mean recovery time after tracking loss for each subject tested, and the mean value for all the subjects ........................................ 70

5.7 Estimated and measured trajectories of the person ........................................ 70

5.8 Robot and human trajectories, being the robot in approach state ........................................ 72

5.9 Relative human robot distance and orientation, being the robot in approach state ........................................ 72

5.10 Definition of the desired position vector according to the position of the person ........................................ 73

5.11 Definition of the desired velocity according to the desired position ........................................ 74

5.12 Robot and human trajectories, the robot being in follow state, for a $(\alpha, \rho) = (0.7m, -\pi/2 rad)$ ........................................ 75

5.13 Relative human robot distance and orientation, the robot being in follow state, with a $(\alpha, \rho) = (0.7m, -\pi/2 rad)$ ........................................ 76
5.14 Comparison of the velocities of the person and the robot in follow state, with a \((\alpha, \rho) = (0.7m, -\pi/2\text{rad})\) ........................................... 77
5.15 Robot and human trajectories, the robot being in follow state, with a \((\alpha, \rho) = (0.7m, -\pi/4\text{rad})\) ........................................... 78
5.16 Relative human robot distance and orientation, the robot being in follow state .................................................. 78
5.17 Definition of the desired position vector ........................................... 79
5.18 Robot and human trajectories, the robot being in follow state .... 79
5.19 Relative human robot distance and orientation, the robot being in follow state .................................................. 80
5.20 Definition of the desired velocity vector ........................................... 81
5.21 Robot and human trajectories, the robot being in follow state .. 81
5.22 Relative human robot distance and orientation, the robot being in follow state .................................................. 82
5.23 Definition of the desired velocity vector ........................................... 83
5.24 One of the trajectory traced by the robot during the user oriented performance test .................................................. 85
5.25 One of the trajectory traced by the robot during the user oriented performance test .................................................. 86
5.26 Error means and standard deviations in the different experiments .. 86
5.27 The division of the complete application time into the six states ... 87
5.28 Application time and task time ................................................... 88
5.29 Number of misses or user special attention during the tests ....... 88
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Delineating a solution</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1 Service robots and mobility</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2 Service robots and interaction</td>
<td>3</td>
</tr>
<tr>
<td><strong>2 Related work</strong></td>
<td>5</td>
</tr>
<tr>
<td>2.1 Interaction among mobile robots</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Interaction between mobile robots and humans</td>
<td>7</td>
</tr>
<tr>
<td>2.2.1 Motion coordination, people following and sociability</td>
<td>7</td>
</tr>
<tr>
<td>2.2.2 Detecting and tracking people</td>
<td>13</td>
</tr>
<tr>
<td><strong>3 Architecture of the system</strong></td>
<td>15</td>
</tr>
<tr>
<td>3.1 Defining the specifications of the problem</td>
<td>15</td>
</tr>
<tr>
<td>3.2 The architecture as a solution for the problem</td>
<td>17</td>
</tr>
<tr>
<td>3.3 A closer look at the architecture</td>
<td>19</td>
</tr>
<tr>
<td>3.3.1 The <em>tracker</em> module</td>
<td>20</td>
</tr>
<tr>
<td>3.3.2 The <em>follower</em> module</td>
<td>21</td>
</tr>
<tr>
<td>3.3.3 The <em>main module</em></td>
<td>25</td>
</tr>
<tr>
<td>3.3.4 The <em>state sequencer</em> module</td>
<td>29</td>
</tr>
<tr>
<td>3.3.5 The <em>robot interface</em> module</td>
<td>30</td>
</tr>
<tr>
<td><strong>4 Implementation of the system</strong></td>
<td>32</td>
</tr>
<tr>
<td>4.1 Timing basis</td>
<td>32</td>
</tr>
<tr>
<td>4.2 The <em>tracker</em> module</td>
<td>36</td>
</tr>
<tr>
<td>4.2.1 The <em>set region</em> component</td>
<td>36</td>
</tr>
<tr>
<td>4.2.2 The <em>detect lines</em> component</td>
<td>38</td>
</tr>
<tr>
<td>4.2.3 The <em>choose person</em> component</td>
<td>39</td>
</tr>
<tr>
<td>4.2.4 The <em>filter odds</em> component</td>
<td>40</td>
</tr>
<tr>
<td>4.2.5 The <em>kalman filter</em> component</td>
<td>41</td>
</tr>
<tr>
<td>4.2.6 The <em>transformation</em> component</td>
<td>44</td>
</tr>
<tr>
<td>4.3 The <em>follower</em> module</td>
<td>45</td>
</tr>
<tr>
<td>4.3.1 The <em>context interpreter</em> component</td>
<td>46</td>
</tr>
<tr>
<td>4.3.2 The <em>velocity estimator</em> component</td>
<td>46</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

An office building in any given city, Bob, the 58 year old intern mailman of the company makes his daily route. As he walks along the halls, his automated mail cart accompanies him, carrying all the mails and parcels yet to be delivered, and freeing him from the burden of pushing a heavily loaded hand-cart. His mail cart, provided by the company, keeps a safe and comfortable position right beside our mailman, constantly adjusting its velocity to his. When Bob stops beside a desk to hand a parcel, the go-cart stops beside him, at the reach of a hand, allowing him to reach the delivery effortlessly. After a minute, Bob resumes the delivery route, the go-cart after him, catching up to him and adopting again the initial comfortable position. A few meters ahead, a pillar narrows the hallway, not making it possible for the go-cart to keep its position beside Bob. Automatically, it slows down and turns in order to occupy the space behind the mailman. Once they have both passed pillar, the go-cart goes back to its initial position, and the delivery route continues.

1.1 Motivation

Though this situation presented above is only a hypothetic scenario, it could describe reality within the next decades. The technological revolution filled the industries with robots decades ago, and might also fill our everyday lives in the close future. Existing robotic systems such as Roomba [10], a robotic vacuum cleaner, MINERVA [28], a museum tour-guide, or the Ohio State University Medical Centre [32], an automated delivering system which serves several hospitals in Ohio, are representative examples of this transition from the industry sector to the service sector. Step by step, the so called service robots are making their place in the sector. As defined by the ISRA (International Service Robot Association), a service robot is a machine that senses, thinks and acts to benefit or extend human capabilities or to increase human productivity [25]. Nurse assistants in hospitals, mail deliverers,
elderly carers, housekeepers, miners or construction workers are other illustrative examples of service robots.

Back to the imaginary scenario, the characteristic features of this go-cart service robot can be described. First of all, our service robot has to detect and track its target (Bob), once detected, it has to coordinate its motion with his, keeping an appropriate distance, velocity and relative position. In addition to this, it has to avoid still and mobile obstacles, and also be able to adapt its behaviour to different situations such as "Bob still", "Bob moving", "hallway narrowing" or "hallway crowded". A robot-user interface must be supported as well, in order to allow Bob to stop the go-cart when he doesn't want to be followed, or start it up whenever he needs its services again. All these features are indeed the common and necessary capabilities of any robot intended to offer its services by following users. They can be summarised in the following: person detection and tracking, navigation and motion coordination, obstacle avoidance, environment dependent behaviour, and human-robot interface.

1.2 Delineating a solution

According to the concepts explained above, this report will address the design and implementation of a following behaviour in a service robot. Though the whole follower will be implemented, a special stress will be set on the navigation, motion coordination and human-robot interface. Thus, obstacle avoidance, and person detection and tracking will show simplified designs, and thus, their limitations in performance will be assumed in the rest of the work. In this way, two key aspects will be emphasised in this thesis: service robot mobility and human-robot interfaces in the context of mobile service robots.

The goal of the overall design is creating a person follower behaviour that can cope with basic social constraints in an indoors context. The expression "social constraints" will be specified in terms of a human-acceptable following and adaptability to a changing context.

The first of these concepts, human-acceptable following, will be defined according to physical parametres such as human-robot minimum and maximum distances and relative human-robot velocity and position, set according to some human territoriality studies [19]. The second concept mentioned, adaptability to a changing context, will imply a dynamic accompanying behaviour according to the environment. The environment will be defined by factors such as the velocity of the target person, the obstacles surrounding him or the space to the nearest walls. The dynamic accompanying behaviour will involve a changing robot position and velocity relative to the human.

In this way, we will try to offer a versatile follower both able to adapt its behaviour to changing circumstances and able to perform a human-acceptable following. This will no doubt increase the effectivity of the services provided by the robot.
as will provide the human-robot interaction with an improved social dimension.

Once settled the aim of this thesis, a closer look on the previously mentioned key aspects will be offered: mobility and human-robot interaction.

1.2.1 Service robots and mobility

In many service robot applications, mobility is a key factor within the service provided. A mail delivery robot should be capable of navigating in an office environment, a remote explorer robot should be able to navigate in an unstructured environment, and tour guides should be able to guide visitors all through the museum. As explained in [30] the two main aspects concerning mobile robots include the perception of the environment and the path planning. The perception of the environment has to be focused on the recognition of obstacles and targets in the space, and the path planning has to deal with the task of reaching goals while avoiding obstacles, optimising travel-times, power consumption, etc. So the necessity of mobility determines the way the sensed information is treated and the way actions are carried out.

Although not all mobile robots need social capabilities to work effectively, those that are meant to share closely their working space with humans can benefit from using them. If we want to employ a robot in a hospital to help nurses carry beds from side to side, we should create a robot whose behaviour is consistent with the social rules of movement in crowded spaces, such as hospital corridors. The same requirement would be found when trying to use a robot to deliver mail in an office. Naturally, another example of a mobile robot that would also benefit from fitting into social rules would be an autonomous shopping-cart that follows a customer in a supermarket.

Though the social rules of movement are a quite complex field of study [15], in our limited framework we will assume that they are defined by the same factors that define the human-acceptable following behaviour, discussed in the beginning of this section.

As we can see, mobility in the framework of service robots is usually constrained by the social rules of space sharing. In the case of a person follower, this feature must be maximised as robot and human are constantly sharing a common space. By carefully observing this aspect in this thesis, the designed algorithm should make the robot perform so that the service provided is never declined.

1.2.2 Service robots and interaction

Interaction, as defined in [18] is a mutual or reciprocal action. In the case of humans and their environment, interaction is carried out thanks to the capacity of retrieving information from the environment, the capacity of analysing it, and the capacity to alter it. Nowadays, thanks to the fast technological revolution, the thought of autonomous machines able to interact with the world in a similar way humans do is no longer an illusion. Man-made devices able to sense the world around them
through sensors, process big amounts of information in short periods of time, and modify the environment through actuators, are becoming workable and cheap.

As the definition reads, service robots perform the mentioned sense-think-act cycle in order to serve humans. Therefore, the interaction between the robot and the human is crucial, and consequently an appropriate human-robot interface is necessary.

A distinction could be made within the available interfaces: interfaces implicit in the spatial relation human-robot, and those explicit in particular communication channels. The pose of the robot relative to the human, its orientation and its velocity conform an interface themselves, and thus, would be examples belonging to the first group. Examples belonging to the second group would be keyboard inputs, speech or facial expressions, as they make use of dedicated sign systems.

Both kinds of interfaces are important in the case of service robots. Particularly, in the case of a people follower, the interfaces of the first kind represent the main body of the application, while the others offer supplementary control. I.e. the robot must be able to interact with the human by coordinating their motions and interpreting the human’s movements. Simultaneously, the use of a keyboard, speech feedback or similar, should provide the user with an extra control over the robot, which will make it possible for him to stop it, resume the following, or choose another user. In the approach presented in this thesis, the first kind of interface will be implemented, plus a simple bumper-pressing interface used to stop and resume the following, and a monitoring screen. However, in a future work, a simple speech synthesiser could be added in order to improve the design and make the human-robot interface more natural. Eventually, the following behaviour could even be integrated in a higher level control system in order to provide a more sophisticated control.

In this introductory chapter the motivation of this thesis has been presented, the proposed solution has been shortly sketched, and the key aspects within it have been posed.
Chapter 2

Related work

This chapter offers an overview of the existing approaches in the field of mobile service robots, stressing those with particular following features. Approaches are been grouped into two different sections: interaction among mobile robots and interaction between a mobile robot and a human or a group of humans. The first section will focus its attention on motion coordination among mobile robots. The second section will present solutions concerning human robot interaction, including people detection and tracking and motion coordination. Finally, a special remark will be set on social considerations and improved human-robot interfaces within motion coordination.

2.1 Interaction among mobile robots

Though this kind of interactions do not fit exactly with the service robot definition, they can conform a pretty illustrative framework for studying movement coordination in mobile robots. They can offer a simplified model of human-robot interaction, that can be helpful to:

- concentrate on motion models for the robots
- isolate and study the basic tools for following systems, such as tracking systems and motion control systems

Some publications in this context will be presented next, mostly focused on motion coordination and modelling.

In this direction, P. Ögren [21] presented a control system for multiple robots moving in formation. His thesis work was aimed to make a group of robots move in formation in a partially unknown environment while avoiding obstacles. The goal was achieved using accurate motion models for the robots and advanced non-linear control combined with convergent dynamic window for obstacle avoidance. The thesis offers a quite exhaustive study of the mathematical model of the coordinated motion and its stability, but does not consider the object tracking problem
at all. This approach shows that an accurate motion model and control can provide a stable moving formation, resistant to disturbances such as obstacles. A graph with the performance of the system in a three robot formation is shown in figure 2.1

In a similar line of work, T. Sugar and V. Kumar [31] designed a control system for coordinated robotic manipulators. The idea was to provide a group of two or three robots with the capacity of handling objects together, i.e., the capacity of cooperative transportation and formation marching. In this case, a lead robot is defined and communicates with the rest via WLAN, the followers are the ones responsible for the coordination and use a different control system for the manipulator (a robotic arm), the path planner and the platform controller. Again, the platform controller, responsible for the motion control of the robot, uses advanced non-linear control tools [23]. This approach introduces object handling coordination to the basic problem, a quite useful feature in service robots.

In these two cases, the motion coordination is shown as a key aspect and the focus is set on the basic theoretical process of coordinated motion. As the communication among the subjects is barely robot-to-robot, the social aspects of the following are not so evident. In a similar way, as being designed for interacting only with robots, they would not deal so well with the spontaneous traces, shapes, behaviours and movement patterns that characterise human motion.
2.2 Interaction between mobile robots and humans

The human-robot interaction represents a step forward from the interaction model presented above. When humans come to play an active role in the relation, some difficulties arise:

- human behaviour does not fit in a model as easily as robot behaviour does, which tends to randomise the situations a robot has to cope with when dealing with humans
- the shape, size and other physical features in a human are highly changeable, thus detection systems need extra flexibility
- a human has no serial port or wireless connection, which makes it necessary to create specific human-robot interfaces

In the following, several approaches will be presented, each of which emphasises different aspects in the interaction between mobile robots and people. Motion coordination and following systems will be shown, and some particular systems that engage exclusively detection and tracking systems will be posed. In the first section, a closer look to systems that provide either improved human-robot interfaces or improved social capabilities will be offered.

2.2.1 Motion coordination, people following and sociability

As explained above, in this section, solutions for motion coordination, people following and sociability will be delineated.

Ku and Tsai present in [13] an autonomous land vehicle (ALV) aimed to follow people using a detection system based in vision. As explained in the paper, the authors tried to implement a simplified tracking system that saves processing time compared to shape and colour recognition. The approach assumes that the person to be followed has a special rectangular-shaped pattern attached to the back. The robot motion commands are calculated according to how this square is perceived along time and space. A later improvement on this system made by the authors [12] adds a robust trajectory planner to this detection system. The main aim of the planner is to keep the visibility of the object at all costs. The task is accomplished by creating three visual constraints (right, left and furthest boundaries, see figure 2.3) and forcing the robot to assure that the person to be followed is inside of these three boundaries.

Another vision based person follower is proposed by LaValle et al. in [14]. In this approach, the control variable is set according to the motion constraints of the follower and to the optimisation process of two different variables. The first one is the probability that the object tracked is in the visual field in the next frame, and
the second one is the minimum time that it could take the object to get out of the visual field. Basically, it consists of a probabilistic forward looking planner able to deal with partially predictable targets.

Taroikh proposes in [33] an approach for a person follower able to track and follow a person in an unstructured environment. The main aim of this system is to create a simple visual tracking system coordinated with a fuzzy controlled motion algorithm. The person identification process is based in colour and shape featured plus a region growing technique. Only the zones in the image with a colour similar to some feature (e.g. shirt or jacket, see figure 2.4) of the tracked person are analysed. The shapes of these regions are subsequently processed to determine which one represents the tracked person. The person’s state is then determined according to the mass of the region and the position of its centre of mass. The motion coordination is achieved through the use of two independent fuzzy controllers, one for steering and the other for speed. Finally, a behaviour control is implemented in order to handle tracking loss, obstacle avoidance or endangered visibility. While the relatively simple visual system speeds up the algorithm, and may lead to misidentifications though, the fuzzy motion controllers present a robust performance when facing uncertain or complex situations.
Feyrer and Zell present in [5] a stereo vision based person follower. Stereo analysis, colour detection, motion detection and contour detection are used to detect and track the person’s face (see figure 2.5). Here, the person’s distance information can be extracted directly from the stereo vision analysis. The motion control is governed by the potential field method. Goals are defined as attractors, while obstacles are repulsors. The potential field is evaluated in the position of the robot, and the resulting motion direction is determined by the gradient of this field. The goal is not exactly the person to follow, but the most recent reachable position in the person’s remembered path. The robot stores the information about the history of the person’s positions, and tries to reach the newest position among them. Using this technique, the dead end situations can be avoided. While this approach offers a flexible person detection and an improved navigational capability, the computational load of the stereo vision processing makes slows down the system. In addition, the system is only able to follow people while their faces are facing the camera. That forces a person to walk backwards while being followed.

![Figure 2.5. Feyrer and Zell's stereo-vision system](image)

The cases above represent a bunch of basic people followers without any socially improved motion coordination. In the following cases, human-robot motion coordination will become a significant bias in the design of the mobile service robots.

Motion coordination itself is the main subject of the approach presented by Prassler et al. in [26]. In this paper, an accompanying behaviour is presented as an useful application for mobile service robots. A side by side following allows visual contact between the followed and the follower and gets closer to a normal pursuit behaviour among humans. The authors design and implement an accompanying wheelchair that follows a person side by side. The design is implemented and shows to be able to keep this relative position to the human while coordinating this behaviour with obstacle avoidance. The architecture of the system detaches the problem into layers. A lower level layer engages the obstacle avoidance and basic motion control services, offering to the top layer an obstacle free navigation. The top layer is responsible for creating the desired accompanying behaviour by determining the desired velocity of the robot in each instant (see figure 2.6). The formation person-robot is successfully kept while dealing with obstacles, such as
doors (see figure 2.7) or narrow passages, though visibility problems may lead the robot to tracking loss. As it will be shown in the next chapter, this approach addresses a very similar problem to the one concerning this thesis, and poses a similar solution as well.

Figure 2.6. Desired velocity calculation for maintaining an accompanying behaviour

![Figure 2.6. Desired velocity calculation for maintaining an accompanying behaviour](image)

Figure 2.7. Accompanying behaviour when dealing with obstacles such as doors

Nakauchi and Simmons present in [19] the a robot that is able to stand in line in a similar way to humans. The robot is able to detect people orientation in a queue, model the line, and find its place at the end of the queue. While queueing, the robot is able to keep its place in the line by moving up or down according to the person in front of it, and eventually, recognize the service point in the beginning of the queue. The design of such a system implies the analysis and implementation of the social rules implicit in the act of lining up. Personal space definition, a matter of study itself in cognitive psychology, is used to create a model of queue formation. Stereo vision analysis allows the robot to distinguish different individuals in the queue, determine their orientation and finally find its place in the queue. This approach shows a working system able to deal with the social rules of motion implicit in lining up. Though the problem addressed is not that of pursuing a person, the social considerations within the motion coordination make this implementation worth mentioning in our context.
Social rules of movement are also contemplated by Matellán in [22]. Matellán presents an improved navigation system that implements a preferred turning direction when crossing humans in a corridor. The idea parts from a existing indoors navigational method (Lane-Curvature Method - LCM), and modifies it so that a preferred side of the corridor is chosen when the robot needs to change the lane. In this way, the author tries to emulate the behaviour of people in crowded corridors, and succeeds in inclusion of basic social rules of motion in a mobile robot. This socially enhanced navigational capability is what makes this example interesting in the framework of this thesis.

The next works will also represent approaches to implement a socially improved motion coordination. However, they will also show the benefits of an improved human-robot interface.

The CERO project, presented by Hüttenrauch et al., emphasises this human robot interface in service robots[9, 8]. CERO is a service robot for light weight object transportation in office environments, aiming to assist motion-impaired people. The architecture of the system includes a speech interface, a graphical remote interface and a visual-characterised interface, in addition to the ultrasonic sensors and motor control that provide the navigational capabilities. Its design was intended to provide a base to make different studies about the usability of robotic interfaces. An interesting observation found in this work is the use of a visual interface provided by the animated figure which sits on the top of the robot (see figure 2.8). It acts accordingly to the state of the robot, and offers a reference point for its direction of movement. In the scope of this thesis, when mobile robots come to share their space with humans, a reference point in the robot can be useful to help humans notice its intentions, and thus make any motion coordination more natural for people.

![Figure 2.8. The CERO robot at the IPLab](image)

An interesting facial and speech interface can be found in the MINERVA project [28], developed by Schulte et al. MINERVA is a robotic museum tour-guide whose aim can be summarised in these three points: (a) attract people’s attention, (b) travel between exhibits during the tour and (c) engage people interest while explaining an exhibit. For this purpose, the robot is provided with navigational and
interfacing capabilities. In this case, the interface proposed consists of an animated face and a speech synthesis module (see figure 2.9). The face interface itself is intended to offer a reference point for the users, communicate the intentions of the robot through expressions, and attract users by adapting its movements to the observed reactions of the people. Though guiding users is not the same as pursuing them, both problems share similar difficulties in the scope of motion coordination. MINERVA should move slow enough to allow people to follow comfortably it, but fast enough not to bore the audience. In particular, it should overcome situations in which people may block its way. In such cases, the interface human-robot shows to be a valuable feature, as presents the robot intentions clearly to the human. Again, the interface itself can resolve motion coordination problems, making human-robot interaction more natural.

**Figure 2.9.** MINERVA’s smiling expression (left) and neutral expression (right)

In [24] Pollack et al. present the PEARL project. PEARL is a project that makes use of several interfacing modalities to provide assistance to the elderly. It is intended to remind users to take on daily actions (cognitive orthotic functions) and to help them navigate their environments. In order to achieve these goals, the main architecture of the system defines three different main states for the robot: remind, assist and rest. PEARL provides a quite complete human-robot interface, through speech synthesis and recognition, face recognition, and a graphical touch interface. Worth mentioning in the ambit of this thesis are the navigational constraints that this application has to deal with. First, a careful human-robot velocity regulation is necessary, according to the speed limitations of the elderly. In addition, considering the users’ physical handicaps, a safer navigation is required so as to avoid collisions with them, adapting in this way the navigational behaviour to the environment. In a similar way as MINERVA does, PEARL has to guide users, not follow them. However, as we pointed out before, the motion coordination problems, that both guiding and following face, share common points.

Some of the existing works concerning motion coordination, people following and sociability have been presented above. The connections between them and this thesis have been presented and, when not so obvious, they have been explicitly pointed out.
2.2.2 Detecting and tracking people

In this section, a few approaches aimed to improve the detection and following phases in a follower will be presented.

Nikovski et al. proposed sensor coordination in [4] to perform a better, more reliable person tracking system. While sonar sensors offer a quite wide sensing angle range, they often do not offer the means to distinguish people from pieces of furniture of similar size. On the other hand, cameras offer a constrained visual field, but provide means to differentiate people from furniture, and distinguish certain individuals, according to colours and shapes. The approach uses sonar to estimate the position of the person, and a camera to provide verification. In this way, sensor fusion is presented as a feasible option to minimise identification errors.

The detection and people tracking can be also enhanced considering the predictable motion that can be observed in people’s displacements in indoor environments. Bruce and Gordon present in [3] a learning process that enables the robot to create displacement patterns in a known environment. A set of training trajectories is presented to the robot while off-line, and spatial goals are identified out of them. The classic probabilistic models used in Bayesian filters are substituted by probability estimations for the goal each detected particle might have. This can definitely help the robot to overcome occlusion problems. However, the stronger assumptions made on human motion model can cause a worse performance as these assumptions might not be correct.

Tracking multiple targets simultaneously also helps robots to deal with mutual occlusions among people, and can be definitely useful in their navigation through populated environments. In [29], Schulz et al. show a solution for a multiple targets tracking system. Particle filters are used in a combination with a joint probabilistic data association filter (JPDAF), which engages the association of the observed features to the appropriate object. The particle filters offer a non-gaussian, more complete probabilistic model for the person motion, and the JPDAF system pro-
vides an answer to the feature-object association. The approach has been proved to successfully deal with mutual occlusions in office environment, using 2D laser range sensors. However, nothing is stated concerning motion algorithms for the robot platform.

All through this chapter a bunch of applications related in various ways to the workscope of this thesis have been presented. Works pertaining all the features present in a person follower have been delineated, in an attempt to offer a general idea of the existing techniques that can be used in this matter.

To conclude with, it is worth mentioning the fact that all these approaches show that the final design elements of any mobile social robot, including:

- the hardware used; such as sensors, actuators and robot platforms
- the control architecture; such as tracking systems, navigating systems and behaviour control
- the user interfaces; such as speech and visual feedback

are highly dependent on the kind of application to develop, and the environment where it is set. In addition, strong interdependencies bound these mentioned design elements.

As we will see in the next chapters, some of the ideas and techniques outlined in this chapter will be present in the work shown in this report.
Chapter 3

Architecture of the system

In this chapter, the architecture of a human-acceptable following behaviour is presented. Its design is justified, and its functional process is described. In order to do so, the problem is first defined considering goals and assumptions. Next, the problem is split into parts and the overall architecture described according to these parts. Finally, the functionality of each module and the relations established among them are posed, so as to show how the whole architecture supports the desired behaviour. The details on the implementation of each of the modules in the architecture will be presented in the next chapter.

3.1 Defining the specifications of the problem

The problem addressed in this thesis was delineated in the first chapter. In the following, a step forward in the problem definition is taken. Starting from the general goal of this thesis, a set of subgoals is be defined, in a way that their completion implies the completion of the general goal. These subgoals are achieved considering a specific context and setting a series of assumptions, so both, context and assumptions, are presented as well.

The goal of the overall design shown in this thesis is to design, implement, and test a person follower behaviour for a service robot that can cope with basic social constraints in an indoor environment. In order to reach this goal, the following set of subgoals must be accomplished:

1. enable the robot to detect and track the position and velocity of a person
2. create an algorithm that allows the robot to move autonomously in order to coordinate its motion with the person being tracked, providing the accompanying behaviour
3. make this accompanying behaviour algorithm dependent on the environment
4. provide the robot with obstacle avoidance capabilities
5. create a basic human-robot interface

These goals have to be accomplished according to a defined context, which is defined as follows:

- the robot and the person to be followed are placed in an office environment
- the robot is provided with a bi dimensional, 180° wide, laser range finder
- the robot has a non-holonomic platform, known as the robotic unicycle [11, 21] or differential drive model
- the environment is unknown

The completion of the goals according to the context is bound to the following design assumptions:

- in general:
  - the person to be followed behaves cooperatively
  - the person to be followed walks at a maximum speed of 1m/s
- concerning the tracking and detection:
  - only one person is to be detected and tracked
  - no occlusion problems occur
- concerning the obstacle avoidance:
  - no mobile obstacles are found in the environment
  - all possible obstacles are detected by the laser range finder
- concerning the accompanying behaviour dependency on the environment:
  - a given external module analyses the environment and outputs an environment descriptive variable to the presented system

In general, these assumptions play down the importance of some of the subgoals previously defined, while underline some others. The detection, tracking and obstacle avoidance are thus set as background issues, while the following behaviour, environment dependant, is underscored.
3.2 The architecture as a solution for the problem

In this section the whole system architecture is presented. First, the overall design is deduced from the specifications of the problem described in the previous section. Then, the general behaviour of the architecture is explained, using a real scenario example to support the explanation.

A number of modules can be defined, according to the subgoals set in the previous section, and so that their coordinated function completes the primary goal. These modules are briefly presented below:

- **person tracker** → this module provides the position and velocity of the person to be followed, according to the laser range data

- **person follower** → this module engages the motion coordination between the person to be followed and the robot. It generates the commands for the motors of the robot according to the position of the person and the state of the robot. It covers these subgoals as well:
  - it has a behaviour dependent on the environment, specified by an external module, whose design is not addressed in this thesis
  - it has a simple obstacle avoidance submodule that modifies the output of the follower according to the nearest obstacles
  - it provides the interface implicit in the spatial relation human-robot, as it was defined in the introduction chapters

- **state sequencer** → this module determine the state of the robot. The states define the way in which the rest of the modules are used. The different ways of using these modules allow the robot to handle different situations in different manners. Standby situation, no person detected, person walking straight or person stopped are some of the situations that this module handles. In this way, it supports the human-robot interface offered by the follower. In addition, it offers a bumper-pressing interface, that affects directly the way in which the robot switches states.

- **robot interface** → this module offers to the rest of the modules a interface to control the robot and retrieve data from it. The module acts as a server: requests data from the laser, the bumpers, and the motors, and commands the motors according to what the follower module specifies. In this way, it provides access to all the devices used on the robot: laser, motors, bumpers, and batteries. The robot interface is not be implemented in this thesis, instead, an open source software project is used.

- **main module** → this module coordinates the communication among all the modules above according to the state set by the state sequencer. In addition, it offers a screen monitoring interface for the user, and supports the communication with the external module that defines the environment.
Once that the main modules of the architecture are defined, their coordinated way of working is shown. Figure 3.1 presents a general schema of the whole architecture, with the five modules previously defined.

The coordination and general way of working of the schemed system will be best understood within an example. The mailman Bob and his mail cart, presented in the introduction, will set the appropriate scenario. The mail cart initially stays still, waiting for a user to activate its motion. Tracker and follower stay inactive, and the main module laser range data, odometry and bumper readings from the robot interface. The system waits a signal from the robot interface. Bob comes around and pushes one of the bumpers of the cart, giving the system the signal that it was waiting. The state sequencer receives through the main module the information about the bumper pressed and switches its state, telling the main module the new state. According to this new state, the main module starts up the tracker sending the laser range data and the new state information to it. The tracker receives these scans and analyses them in order to detect movement, once detected, it informs
the state sequencer, which switches to the next state, telling the main module to start up the follower. In this way, the follower starts receiving the position of the person and the context identifier, plus the actual state and odometry of the robot. According to that information, the turn rate and speed for the motors is calculated and submitted to the main module, which finally sends this information to the robot interface. This module sends a request of speed change to the motors of the robot, which start moving consonantly. Now the robot is following Bob.

Therefore, in each time step new laser range data, new odometry information and bumper readings are retrieved from the robot interface. The main module diverts the laser range data, and actual state to the tracker. Using this information, the tracker calculates the position and velocity of the person and sends this back to the main module. The follower receives through the main module the position of the person, the odometry and state of the robot, and the context identifier. Considering all these factors, the follower gives back to the main module the turn rate and the speed, which is eventually directed to the robot interface. The robot interface will make the robot platform move, creating the accompanying behaviour.

When Bob stops, the tracker module outputs to the main module a person velocity equal to zero. The state sequencer is then informed about this fact, and it switches to a different state. That makes the follower module act accordingly, and calculate the precise turn rate and speed needed to make the cart stop at a comfortable distance to Bob. Once Bob resumes his walking, the process described one paragraph above is repeated until the robot is effectively accompanying him. A change in the environment, such as a pillar narrowing the hall, makes an external module input a different context identifier to the main module. The follower is informed subsequently and it calculates the turn rate and speed taking the new environment into consideration. The new commands computed by the follower module tend to place the cart just behind Bob, so that cart and mailman can pass the pillar comfortably. As soon as the pillar is left behind, the context identifier changes again and the follower reckons the appropriate motions commands to direct the cart to the right side of Bob.

This example intends to explain the general working basis of the architecture, by showing the coordination among modules, the kind of information they exchange, and how this is made within time. Detailed information on how each module functions will be presented in the next section.

3.3 A closer look at the architecture

This section goes through each of the modules in the architecture, offering a functional description of their internal processes. For each of the modules, inputs and outputs are pointed out and a general explanation of their behaviours is presented. Finally, their internal architecture is shown and commented.
3.3.1 The tracker module

As previously explained, the tracker module provides the whole system with the information about the position and velocity of the person to be followed. In figure 3.2 the architecture of the tracker module is presented. The figure shows how the module uses the inputs, laser range and odometry data, in order to output the estimated position of the target person.

Figure 3.2. The architecture of the tracker module

The main idea of the system can be summarised in the following steps. First, an angular region is set in the laser range data where to look for the person. Then, patterns in that region which could be the person’s legs are detected. Next, the choose person component chooses one of the detected patterns to be the person’s leg. Finally, the component filter odds filters out choices that may be mistaken, according to the distance to the previous choice. In the end, the position of the chosen pattern is input in to a kalman filter, which outputs the position and velocity of the person into the main module. In addition, the estimate error covariance is also output.

In order to understand the working basis completely a more detailed view of the module is presented next. Two of the formerly mentioned steps are crucial in the understanding of the whole process: defining the angular region where to
look for leg patterns, and choosing the most appropriate pattern among all the leg hypotheses. The approach presented here addresses these problems in two different ways, according to the state of the robot. When the robot is not moving, the region is set using the component labelled \textit{set region still}, and when the robot is moving, the component used for the same purpose is \textit{set region non-still}. In the first case (\textit{pre approach} state), the region is defined according to the difference between two subsequent laser range data sets, which gives a hint on movement. In the second case (\textit{approach} and \textit{follow} states), this scan differencing technique can no longer be used to determine useful movement information, as the robot itself is moving. Therefore, the region is set according to the prediction for person’s leg position in the next frame. This prediction is made by a Kalman Filter \cite{34} implemented in \textit{Kalman Filter} component resident in the \textit{tracker}, being the angular region set around this prediction.

This \textit{Kalman Filter} component plays also an active role in the choice of a single leg pattern among all the leg hypotheses. Again, this task is dependant on the state of the robot. In this case, the hypothesis closest to the robot is selected before the Kalman Filter is initialised (using the modcomponentextit{choose line closest}). Once the filter has been initialised, the picked hypothesis is the one closest to the predicted position for the person’s leg (using the component \textit{choose line kalman}). As it will be seen in the \textit{main module} subsection, the filter is initialised when the first person measurement is made, i.e. when the robot state changes from \textit{still} to \textit{pre approach}.

All in all, the Kalman Filter is used to predict where the person’s leg will be in the next frame, to choose a pattern among the leg hypothesis, and to estimate the position and velocity of the person. In fact, these uses involve the two well knowns steps in Kalman Filter usage: \textit{prediction} and \textit{correction}. More details on the implementation of the filter will be given in the next chapter.

The Kalman Filter usage involves another important aspect within the \textit{tracker} module. The Kalman Filter is always making predictions according to the last time step person position estimation. The reference system used by the robot is always attached to it. Thus, when the robot is moving, the estimation made in the last frame is not any longer expressed in the actual reference system. In this way, a transformation on the last estimation must be be done according to the odometry information, so that the prediction made by the Kalman Filter can be computed. This transformation is implemented in the component labelled as \textit{transformation}.

Summarising, the \textit{tracker} module presented above addresses the detection problem making use of movement information. Once movement is detected, a Kalman Filter is associated to the person. From then on, this filter sets the region where to look for the person, and gives an estimation of the person’s velocity and position.

### 3.3.2 The follower module

This is the module responsible for the motion coordination between the robot and the person to be followed. In order to make this coordination possible, the module
receives the position and velocity of the person, the actual state and odometry of the robot, plus a context identifier and the actual laser range data set. According to all these data, the module generates the turn rate and speed specification for the robot motors, that should make the robot coordinate its motion with the person. In figure 3.3 the internal architecture that supports this behaviour is shown.

![Diagram of the follower module](image)

**Figure 3.3.** The architecture of the follower module

In the following, the general working basis of this module is presented. The desired position for the robot is defined according to the actual state, a context identifier and the position and velocity of the person. This desired position is the position that the robot tries to reach, thanks to the commands that the module outputs. Once this position is defined, a desired velocity vector is computed, considering the actual state and the velocity of the person. This desired velocity represents the velocity that the robot tries to acquire in order to reach the position described before. Finally, the command motors component translates this velocity vector into turn rate and speed commands, taking into consideration some simple obstacle avoidance constraints, which will be detailed in the implementation chapter. The turn rate and speed is then output into the main module, which sends them to the robot interface, making the robot move in accordance.

A more detailed look at the module architecture is described in the following.
First of all, in order to understand the algorithm properly, three aspects are pointed out.

- the desired position for the robot is always defined in terms of the state of the robot, the velocity and position of the person, and the context identifier. The context identifier determines two key parameters for positioning the robot: \{\alpha, \rho\} \footnote{The angle \(\alpha\) is defined negative. The units used in \{\alpha, \rho\} are meters and radians, respectively}. As shown in figure 3.4, they define the desired position of the person relative to the human.

- special behaviours are be considered in order to cope with these two situations: probable obstacle collision and probable visibility loss. When an obstacle is within a threshold zone around the robot, a special flag (obstacle descriptor) is sent to the component command motors, which deals with the situation modifying the computed commands. In an homologous way, when the person being followed is close to get out of the angular range of the laser, a particular flag is raised (out of sight flag). That makes the definition of the desired velocity vector slightly different, and makes the component command motors define the commands in order to keep the visibility of the person.

- the velocity of the person perceived by the module tracker is relative to the robot. An accompanying robot should have the same velocity as the person that it is accompanying. Therefore, the absolute velocity of the human to be followed is an useful variable itself. In this approach, the robot estimates the absolute velocity of the person according to its own velocity, available through its odometry data. This calculation is made in the velocity estimator component, according to basic dynamics laws \cite{16}.

The way in which the listed aspects are integrated in the algorithm is described next. The first step in the algorithm engages the retrieval of the velocity of the
person and the reckoning of the absolute velocity of the person using the odometry information of the robot. At the same time, the context interpreter chooses the \( \{ \alpha, \rho \} \) that suit the given context identifier. Then, the set desired position component can compute the desired position, according to the state of the robot, the person velocity and the \( \{ \alpha, \rho \} \) parametres. This component also engages the detection of a probable visibility loss, and raises the corresponding flag if necessary. In case the state of the robot is approach, the \( \alpha \) parametre is despised, as this state usually implies that the person trajectory is not straight enough for the robot to reach the relative orientation implicit in \( \alpha \), as it will be noted when the state sequencer is detailed in the implementation chapter.

The next step in the algorithm is the calculation of the desired velocity vector. The component labelled set desired velocity engages such task making use of the information about the state of the robot, the desired position, and the person’s velocity, plus the out of sight flag. When the robot is in approach state, the desired velocity vector is defined so as to move the robot towards the desired position. If the robot is in follow state, the velocity is defined to make the robot reach the desired position and, once it is reached, keep up to the person’s velocity. When the out of sight flag is raised, the velocity is set to move the robot towards the person. In this last special case, the robot does not move towards the person, but only turn, trying to head towards the human. This is made possible thanks to a coordinated work with the command motors component, which detects the raised flag as well, and sets the speed to zero.

Finally, the component command motors translates the desired velocity vector into turn rate and speed commands, understandable commands for the motors in the robot. The robot interface does not offer heading control, only turn rate and speed control. Thus, the speed can be controled in a straight forward manner, while the heading needs a kind of control technique that provides effective heading control. Initially, two different approaches were cosidered: a fuzzy controller and a PID (Proportional, Integral and Derivative) controller. While the fuzzy controller can offer a more understandable design, its mathematical analysis and synthesis are limited. A PID controller can be fully described and synthesised using mathematical models, and the resulting dynamics can be explicitly controlled. In that way, the design can be based on the model of the turn rate system of the robot, estimated with classic plant identification experiments. Using this model and basic mathematical analysis, a nice approximation for the design parameters of the controller can be calculated. In addition, previous experiences with PID controllers suggested shorter design time and implementation.

The response to obstacles and visibility loss is also implemented in this last component. In the case of obstacles, the turn rate and speed are modified according to the position of the closest obstacle considered. This position is determined by the obstacle detection component according to the laser scan. The out of sight situation is handled by setting the speed to zero, thus making the robot turn around trying to place the person being followed in the centre of the visual field.
All in all, the follower module consists of a system that computes the desired position and velocity of the robot according to the state of the robot, the position and velocity of the person to be followed, and a context identifier. Once these desired parameters are created, they are transformed into commands for the motors. During the process, the system engages basic obstacle avoidance and keeps the visibility of the person. In few words, this is the way in which the module supports the accompanying behaviour.

3.3.3 The main module

This module represents the main frame of the whole system. It is the module in charge of the coordination among the rest of the modules. It receives all the outputs from the rest of the modules, and routes them to the appropriate destinations. Besides, it supports a screen monitoring interface and communicates with an external module that specifies the context. Its general schema is shown in figure 3.5.

![Figure 3.5. The architecture of the main module](image)

The idea behind this module is having a "brain" (the module coordinator) that coordinates in time all the outputs/inputs from/to the rest of the modules. Two
internal components provide the screen interface, and the communication with the external context specifier. They are coordinated by the module coordinator in the same way as the rest of modules.

Mostly, all the weight of this design lies on the module coordinator. Before going down into the details of this component, a couple of remarks will be done on the screen interface component and the context reader. The first of them offers to the human user information about the status of the robot, outputting into the screen some of the information retrieved from the rest of the modules, plus some internal information from the follower module and the tracker module. This internal information has not been explicitly included in the architecture diagrams for simplicity matters. Concerning the context reader, the external module that specifies the context is simulated using keyboard inputs from the console, being different keys associated to different contexts.

The module coordinator consists essentially of a sequential flow of communications with the different modules. In the figure 3.6 this flow chart is presented. The sequence of actions (communications) is determined by the state of the robot and the time. Depending on the state of the robot, a different branch of the tree in the figure is chosen. Starting in the point ** (see figure 3.6), it takes it roughly one sample time to go back to the same point.

The system starts in the starting box in the common branch of the tree. The robot interface is immediately initialised, and then a discriminant module decides which branch will be executed, according to the actual state. The states are defined in order to make the robot be able to handle different possible situations that it might encounter. The six different states are described below (detailed information about the transition among these states can be found in the state sequencer module subsection in the implementation chapter):

**Start.** This is the initial state, the first state after the robot is initialised. In this state, the robot is either waiting for a start up signal from a user, or recovering from a tracking loss. The odometry data, and the laser range data are retrieved from the robot interface. No other special action is required, but the actions common to the rest of the states.

**Still.** This is the next state after the start state. In this state, the robot is looking for a person to follow. Again, the odometry data and the laser range data are obtained from the robot interface. After this retrieval, the tracker module is accessed. The difference between the actual laser range data set and the previous one is sent to the tracker, which looks for movement there. So, the set region still component and the choose lines closest components in the tracker are used. Finally, the measured position of the person is retrieved. While this measured position is zero (no person detected), the robot states in this state.

**Pre approach.** This is the intermediary state between the still state and the approach state. This state is characterised for starting up the kalman Filter in
the tracker, preparing the robot to use the filter while it is moving. The robot does not move in this state. Laser range data and odometry data are retrieved from the robot interface. The tracker module is given the Kalman initialisation command if the Kalman Filter had not been initialised before. The range data set difference is then sent to the tracker, and the tracker modules set region still and choose lines kalman are used. Finally, the person estimation is obtained from the tracker.

**Approach.** In this state, the robot tries to reach a minimum distance to the person to follow, no matter their relative position. In order for the robot to be able
to occupy a relative position to the person to be followed, the person should "walk straight" during some time (the curvature of his trajectory should not be too high). If not, the robot won’t have the time to reach the desired relative position to the human. Considering this fact, in this state the robot tries to reach the closest point at a distance $\rho$ to the person. Once the human walks straight for some time, the robot switches to the follow state. As usual, the laser range data and odometry data are first retrieved from the robot interface. Then the tracker module is sent the laser range data, the odometry data, and this time, the components in the tracker labelled as set region non still and choose lines kalman are used. After the context is read from the context reader, the main module sends this context, the position and velocity of the person, the odometry data, the actual state and the laser range data to the follower. As a result, the follower outputs the pertinent turn rate and speed into the main module. Finally, these two variables are input into the robot interface so as to make the robot move.

**Follow.** If the robot is in this state, it tries to reach a relative position to the human according to $\{\alpha, \rho\}$. Once the person is tracing a trajectory with a low enough curvature, the robot is able to consider the $\alpha$ parameter when trying to reach a relative position to the human followed. The whole working schema is identical to the one in approach. In this case, the follower itself acts differently according to the state from which it is called. In this way, the system can support different behaviours in approach and follow, as it is explained in the follower module subsection.

**Terminate.** This state is aimed to close up all the modules and assure a clean exit from the application.

Once the branch of the corresponding state has been executed, the bumper readings are retrieved from robot interface. Afterwards, the state sequencer is given the triggers, the actual state and the previous state. The triggers include a variety of parameters that determine the next state to adopt. A detailed description of them can be found in the state sequencer description in the implementation chapter. The state sequencer returns then the next state for the system, and the flow is diverted to the point **(see figure 3.6), right before the state discriminant. This loop is repeated continuously, until the state is switched to terminate. In that case, the state discriminant leads the flow to the terminating box and the application is be terminated.

It is convenient in this point to indicate here the different use of the module tracker that is made in the different states. As it is explained in the tracker subsection, the set of the region where to look for legs, and the choice of the leg hypothesis depends on whether the robot is moving or not and if the kalman filter has been initialised or not. As a consequence, in the different states the robot makes uses of the module in slightly different ways. In contrast, the module follower is used in
the same way in the approach and follow states. However, this module itself performs different operations in both states according to the given information about the actual state. In this way, the module follower distinguishes the state from it is being used and acts consequently.

As explained above, the main module is the "brain" that coordinates all the modules in the architecture. It determines in each moment which modules to call, and how to call them, according to the state of the system. That provides the system with up to six different behaviours, each of them corresponding to one state, and aimed to deal with different situations. In addition, this module also offers a screen monitoring interface, and the communication with the external context specifier.

3.3.4 The state sequencer module

The state sequencer is the module that sets the state of the system. It uses the information about the actual and previous state, and a set of triggers retrieved from the whole system, in order to determine the new state. As shown in figure 3.7, the heart of the module consists of a synchronous state machine whose state is updated each time step.

![Figure 3.7. The state sequencer module](image)

The state machine component consists of what it is known as a finite state machine [17]. A set of states are defined (start, still, pre approach, approach, follow and terminate) and transitions among them expressed as a function of the retrieved triggers. An initial state (the start state) is set when the module is started up, and the mentioned triggers are continuously (each sample time) retrieved from the main module. The transition conditions for the actual state are always checked, and if satisfied, the particular transition is accomplished. When a specific state is reached (the terminate state), the main module halts the whole application.

As described in the main module section, each state is associated to a different behaviour, so the state sequencer provides different behaviours according to different situations, which are defined according to the triggers. A more detailed view of
transitions and triggers is presented in the implementation chapter.

On the whole, the state sequencer, together with the main module, holds the frame of the whole architecture. If the main module determines how to use the modules in each state, the state sequencer determines the way in which the states are interrelated, and how they are sequenced to handle different situations.

### 3.3.5 The robot interface module

The robot interface module offers to the rest of the modules an interface for retrieving information from the robot and sending commands to it. Basically, it collects the requests and commands coming from the main module and transforms them into understandable commands and requests for the robot. In figure 3.8 the structure of the module is shown. The module gives access to the four different devices used from the robot: the laser range finder, the motors, the bumpers, and the battery. The "heart" of the robot interface is the player interface component, which has not been designed in this thesis, instead the Player software [6], a open source project, has been used.

![Diagram of the robot interface](image)

**Figure 3.8.** The architecture of the robot interface

Thus, the main module can read from the robot interface laser range data sets, velocity and odometry from the motors, identifiers of the bumpers pressed, and battery levels. These conform all the sensory data sources available. On the other hand, turn rate and speed can be commanded to the motors, odometry can be reset,
and some devices can be configured (initialised). These two groups conform all the inputs and outputs available in the robot, and therefore play an important role in the architecture definition.

However, the robot interface is a semi-detached module from the architecture of the system. The way in which the robot’s actuators can be commanded and its sensors read, barely influences on the main architecture. I.e. it is not essential for the architecture presented to consider how this module works, but which information it can retrieve from the robot, and which commands it can deliver to the robot. So in a way, the main body of the architecture is independent from this interface. Therefore, some other existing interfaces for robots, such as Aria [2], could have been chosen instead of this one.

So far in this chapter the architecture of the system has been presented. The problem that the system was meant to solve was first defined, analysed and split into parts. The structure of the architecture was then deduced from the constituent parts. Finally, a broad description of the functionality of all the modules in the architecture and their interrelations was given. So, the way in which the architecture supports the solution to the problem (i.e. a human-acceptable following behaviour) has been explained. The way in which the architecture actually manages to carry out all the actions described above will be the main subject in the next chapter, the implementation chapter.
Chapter 4

Implementation of the system

This chapter presents a detailed overview of the modules described in the previous one. The main aim is now set on explaining how the functionalities associated to each of those module are implemented. In order to do so, each of the five modules shown in the last chapter (main module, tracker, follower, state sequencer and robot interface) is revisited and their internal modules depicted. First, a preliminary section will explain some common considerations concerning time issues bound to the implementation.

4.1 Timing basis

Time coordination is shown to be a critical matter [20] in the implementation of any real-time control algorithm. The algorithm designed in this thesis fits with the classic discrete time control loop in figure 4.1.

Figure 4.1. Classic discrete time control loop
The techniques that calculate the control signal for the robot assume a constant time $T_s$ between each time step. So in order to avoid miscalculations, the should assure that the algorithm makes a loop each $T_s$ seconds. At the same time, input from the robot will be received at a set frequency, dependant on the laser range finder and the robot interface configuration. This sets a new constraint to the algorithm timing. Finally, according to the real system (a moving person) that the robot has to deal with, an appropriate sampling frequency should be set, so as to be able to perceive it properly.

Therefore, the algorithm should provide time coordination in order to consider these timing aspects. Responsible for this time coordination in the system is the main module, particularly, the module coordinator component inside of it. In general, all the modules are involved in the coordination, because they are all time consuming tasks, and especially some of them, like the robot interface, as it controls the sensing and acting frequencies. This is the main reason why this timing section is not included in the main module section in this chapter, but in its own, independent section. However, the only module engaging this coordination in an active manner is the main module itself.

The time factors determined by the system are presented and analysed in the following. Afterwards, the solution adopted for the time coordination will be explained.

**The laser range finder** The laser range finder is configured to work with an angular range of 180° (the maximum range offered) and a resolution of 2 points per degree (sufficient for this application). The minimum time it takes to the laser range finder to make a scan with this angular range and resolution is 0.0267 seconds (37.5Hz). That means that the 360 points of each range laser data set are produced each 0.0267 seconds. In order to deliver range data sets at this rate and resolution, a 216Kbps connection with the PC on robot would be necessary (according to equation 4.1).

\[
37.5 \left[ \frac{data\_set}{s} \right] 360 \left[ \frac{points}{data\_set} \right] 2 \left[ \frac{bytes}{point} \right] 8 \left[ \frac{bits}{bytes} \right] = 216Kbps \tag{4.1}
\]

However, the connection from the laser range finder to the PC on robot is a serial port with a maximum capacity of 115.2Kbps. According to that capacity, the laser range finder should be able to deliver data at a rate of 20Hz (see 4.3).

\[
115.2 \left[ \frac{Kbit}{s} \right] \frac{1}{360} \left[ \frac{data\_set}{point} \right] \frac{1}{2} \left[ \frac{point}{byte} \right] \frac{1}{8} \left[ \frac{bits}{byte} \right] =
\]

33
Nevertheless, this rate (20 Hz) will never be reached due to the robot interface, as it will be explained below.

The **robot interface**. The player interface inside the robot interface communicates with the robot (with the PC on robot), and offers a complete set of data collected from all the devices to the proxies at a set frequency. This frequency can be set up till 100 Hz (the fastest Player can run on Linux, at least with a 2.4 kernel), so this is not a time limitation for the time coordination.

However, the driver offered in Player for the sicklms200 device allows only three different values for the baud rate of the laser: 9600, 38400 or 500000. That means that Player can only configure a communication with the laser at one of these three speeds. 500000 is an unreachable value according to the kind of connection laser-robot (serial port, RS232), which can reach a maximum of 115.2 Kbps. So, the maximum baud rate that can be configured with this laser-robot connection is 38400.

Consequently, the maximum frequency at which the player can offer complete data sets from the laser range finder is 6 Hz (5 Hz effectively), according to equation 4.3.

\[
38.4 \left[ \frac{Kbit}{s} \right] \times \frac{1}{360} \left[ \frac{data\_set}{point} \right] \times \frac{1}{2} \left[ \frac{point}{byte} \right] \times \frac{1}{8} \left[ \frac{bytes}{bit} \right] = 6 \left[ \frac{data\_set}{s} \right] = 6Hz
\] (4.3)

In future improvements, the laser range finder could be connected to the USB port in the PC on robot using a RS422 line with USB/RS422 converter, that allows a communication of 500000 baud with the laser. That would overcome the 38.4 Kbps limitation of the serial port, and would allow the player interface to configure the sicklms200 driver to a baud rate of 500000 baud. That way, data could be retrieved from the laser as fast as it is being produced (37.5 Hz).

**The algorithm.** The time it takes to the system to make a complete control loop (described in the module coordinator inside the main module) ranges from 0.018 to 0.02 seconds (50 Hz approximately). Considering the worst of the cases, the algorithm is updated each 0.02 seconds. According to this, the computational load of the algorithm is not a time limiting factor at all.

According to the points presented above, new data from the laser is available in the robot interface with a frequency of 6 Hz \(^1\). Considering a maximum human
walking speed of 1 m/s, the maximum distance a person would cover between two subsequent time steps is 0.16 m. This distance should be considered in the tracker module in order to help it discard detection errors.

The solution adopted here sets the laser range finder to work at a frequency of 37.5Hz using the player interface. The communication baud rate between laser and the PC on robot is set to 38400 baud (the default value), using the player interface as well. Naturally, not all the data sets produced by the laser are retrieved by the robot, according to the limitations in bandwidth presented before. Finally, the player interface is configured to retrieve information from the devices at a rate of 6Hz. The final configuration is shown in figure 4.2.

![Diagram](image)

**Figure 4.2.** Communication rate schema

The algorithm runs faster than sensory information is retrieved from the robot, so the algorithm should wait for the robot to have new information from the laser in each loop. The robot interface itself makes the algorithm wait until this new data is received, in this way, no old data from the laser is ever used.

Using old data from the laser can produce unexpected problems in the tracking system. For example, if the actual laser data set is identical to the previous one, due to old laser data usage, the tracking system may interpret that the person did not move at all after the last time step, and thus will output a wrong person velocity estimation.

However, a more strict solution would check the time stamps of the laser range data sets in each loop. Data sets with the same time stamp as the last loop’s time stamp would automatically be discarded. Though with the actual solution data sets are never repeated, this would assure that old data from the laser is never used,

\[\text{1} \text{This is the theoretical, maximum value for the frequency. In practise, data is received at a frequency of 5Hz.}\]
regardless of the speed configuration in the player interface, and the laser range characteristics.

All in all, the restricted bandwidth of the serial connection and the limited configurable baud rates for laser in the player interface force the whole system to work at a frequency of 6Hz. Setting the robot interface to work at this frequency, and running the module coordinator without any particular time coordination module shows to be a sufficient solution for the time coordination, as the player interface makes the program flow wait until new data is received from the laser. Naturally, the maximum displacement that a person can make between subsequent time steps must be considered for detection matters.

4.2 The tracker module

The aim of the tracker module is to come up with the person’s position and velocity in each time step. This position and velocity are in fact estimations of the real position and velocity of the person, made according the estimation process explained next. As explained in the previous chapter, the estimation process is basically held by these stages: first, an angular region where to look for the person is set, then leg hypotheses are identified in this angular region, subsequently, one of the hypothesis is chosen, and later a filter determines if it is an acceptable measurement of the person’s position. Finally, a Kalman Filter determines the estimation according to this measurement. The correspondence of these steps with the components inside the tracker module was depicted in the architecture chapter, in the following, their internal behaviour will be described.

4.2.1 The set region component

This component is designed to determine an angular region in the actual laser range data set where to look for the legs of a person. In order to do so, it makes use of the actual laser range data set, the previous one, and the prediction made by the Kalman Filter on the person position in this time step. The angular region limits are output as two indexes relative to the laser range data set vector (size $1 \times 360$).

As briefly explained in the previous chapter, this component performs two differentiated functionalities. When the robot is not moving (still and pre approach states), it uses the component set region still, and when the robot is moving (approach and follow states), the component set region non still is used.

Set region still $\rightarrow$ This component uses the difference between two subsequent laser range data sets. It looks for a pattern associated to movement in this scan difference and outputs the region limits in the scan where such a pattern is found. In case it is not found, the region limit values are set to the previous ones, which are initialised to zero each time the still state starts. In that way, the region limits are zero until movement is detected.
First, the scan difference is filtered, forcing to zero all the values lower than a set threshold, so as to make sure that any value different to zero in the difference is significant enough to be associated with a person’s movement. Afterwards, a simple algorithm examines the filtered difference looking for bulbs. A bulb is defined as a group of \( n \) points in the scan difference having the same sign. If a bulb of positive points if found after a bulb of negative points, or vice versa, the region occupied by both bulbs is output as the angular region where movement is located. If such sequence of bulbs is not found, an informative value is output signalling the absence of the particular movement pattern.

**Set region non still** → This component uses the actual laser range data set and the Kalman Filter prediction for the position of the person in the actual frame (defined in polar coordinates as: \( \{ \zeta_{predicted}, \gamma_{predicted} \} \)). In few words, it sets around the prediction an angular sector where the person might be.

![Figure 4.3. Region setting based on the Kalman Filter prediction](image)

The **size focus region** parameter defines the distance in meters around the prediction where the person is expected, and is set in order to define a region where both legs of a person are contained. In figure 4.3 the trigonometric relation between this parameter and the angular region is shown.

\[
\delta = 2 \tan \left( \frac{\text{size focus region}}{2} \right) / \zeta_{predicted}
\]  

(4.4)

Equation 4.4 defines the amplitude of the region where to look for the person. Once \( \delta \) is defined, it is centred around the angle of the predicted person \( (\gamma_{predicted}) \), and the limits of the angular region are defined consequently.

In both cases, the output consists of two values which delimit a region in the laser range data set where the person’s legs might be located. These region delimiters
are send to the component detect lines, which will try to find a "leg-like" pattern in that region.

4.2.2 The detect lines component

This component is aimed to determine a series of legs hypotheses within the angular region determined by the set region component. It outputs a matrix containing the start and end indexes of the all the leg hypotheses, plus their average distance to the robot. The number of lines detected is registered in a variable accessible by the choose person component.

The basic idea of the algorithm is to locate groups of consecutive points in the laser data set that could represent a person’s leg. The algorithm considers that a leg is a minimum number of points at a similar distance to the robot. The minimum size of the leg should depend on the distance. However, the implementation of such dependency showed no obvious improvement in the algorithm. Thus, a set minimum size of a leg was calculated according to the angular resolution of the laser and the maximum distance at which a person can be detected, and was later refined experimentally. The similarity in distance to the robot is defined according to the parameter person line threshold, whose value was set experimentally as well. In figure 4.5 the meaning of this value is illustrated.

![Figure 4.4. A group of points detected as a line in a fragment of a laser range data set](image)

Figure 4.5 shows some leg hypotheses made on real scan, the two lower hypotheses are the legs of the person. In figure 4.6 the complete laser range set data for the same frame is shown, where the real person’s legs can be noticed.
Figure 4.5. Leg hypotheses on a fragment of a real scan data set

All in all, the matrix with all the information about the lines found, plus a variable specifying the number of lines found summarises all the information about the leg hypotheses calculated.

4.2.3 The choose person component

This component is the responsible of choosing a line among all the hypotheses output by the detect lines component. First, it chooses a line, and then it calculates the index in the laser data set vector and the distance that represent the chosen leg hypothesis. This index and distance is considered to be the measurement of the person’s position.

Again, two different components can be used to decide which line hypothesis is the person’s leg. According to what was explained in the previous chapter, the component choose line closest is used in the still state, while the choose line kalman is used in the pre approach, approach and follow states. Below follows a description of both components:

Choose line closest → This component receives the person’s leg hypotheses and picks up the one which is closer to the robot. As long as it is only used in the still state and only in the brief transition from still to pre approach, this method is proved to be effective.

Choose line kalman → In essence, this component chooses the detected line closer to the prediction made by the Kalman Filter. It receives the prediction and its error covariance, and the whole set of hypothesis. It picks the line which is closer to the prediction, and checks its absolute distance to the prediction. Only if this distance is bigger than a number of $\sigma_{xx} \ (\sigma_{xx} = \sigma_{yy})$ the choice
The value of $\sigma_{xx}$ is directly deduced from the error covariance of the prediction: $\sigma_{xx} = \sqrt{P_{k+1}[0,0]}$. If this condition is not fulfilled, a miss is notified to the main module, which will temporarily consider the previous person measure as the actual measure. If a number of misses happen subsequently, the system is reset, as it is stated in the state sequencer module section in this chapter.

Once one of the components above has set a line to be the person’s leg, the index of the leg in the laser data set and its distance are calculated according to the parameters of the line. The distance to the leg is set as the average distance of the points in the line. Similarly, the index chosen for the leg is calculated as the mean index of all the points of the line. The position of the person is then assumed to be the position of the detected leg. Though this approximated detection works itself, future improvements should make this component look for two legs and estimate the position of the person according both.

Finally, the main module makes obtains the Cartesian coordinates of the person and sends to the filter odds component these coordinates as the person measurement.

4.2.4 The filter odds component

The use of this component is pointing out and rejecting person measures that might be mistaken. A mistaken choice can be identified as a measure of a person too far away from the last frame’s measure. In this way, this component receives the actual measured position of the person, compares it with the last measured position of the person, and determines if the actual measure should be rejected or not regarding a maximum person displacement parameter.
The working basis is as simple as follows: if the distance between the actual and
the last person measure is bigger than a threshold parameter, the actual measure
is despised, and replaced by the previous one. If the distance is smaller than the
threshold, the calculated measure is not touched. The threshold is set regarding
the calculated maximum displacement for a person at a velocity of \(1 m/s\) in \(0.2 s\), to
which a safety margin is experimentally added, resulting of a threshold of \(0.8 m\).

### 4.2.5 The kalman filter component

The Kalman Filter is a component aimed to predict where the person is going to
be in the next time step, and estimate the person’s position and velocity in the
actual time step. For these two steps, the component only needs as an input the
information about the estimation of the position of the person in the previous time
step and the measurement of the person’s position in the actual time step.

The working process of a Kalman Filter is extensively addressed in many pub-
llications [34] and shows to have many variations and applications. In this case, the
chosen option is the basic Kalman Filter (discrete Kalman Filter), and the appli-
cation field is 2D object tracking. No further details will be given concerning the
Kalman Filter basis, but the ones directly related to the model of the system used
in this thesis, and the implementation of the mathematical formulation.

The Kalman Filter used in this component assumes that the model of the
discrete system in question (the movement of the person to be tracked) is a linear
stochastic difference equation (4.5).

\[
x_k = A x_{k-1} + w_{k-1}
\] (4.5)

The state \(x_k\) is related to the measurement \(z_k\) according to the equation 4.6,
being the measure all that can be observed from the state.

\[
z_k = H x_k + v_k\] (4.6)

As usual in the basic Kalman Filters, \(w_k\) and \(v_k\) are considered to be white
noise. They are considered independent and normal distributed stochastic variables
with covariances \(Q\) and \(R\), respectively.

Considering this model, the model chosen to describe the position of the person
as a system is a second order model. This bounds the \(A\) and \(H\) matrices to the values
in equation 4.7, and the state vector and measurement to the shape presented in 4.8.
This definition makes the Kalman Filter provide the system with an estimation of
the position, velocity and acceleration of the person, using only a person position measure.

\[
A = \begin{pmatrix}
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix};
\quad H = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 
\end{pmatrix}
\quad (4.7)
\]

\[
x_k = \begin{pmatrix}
x_{x_k} \\
x_{y_k} \\
x_{dx_k} \\
x_{dy_k} \\
x_{ddx_k} \\
x_{ddy_k}
\end{pmatrix};
\quad z_k = \begin{pmatrix}
z_{x_k} \\
z_{y_k}
\end{pmatrix}
\quad (4.8)
\]

The model definition is completed once the process noise covariance \(Q\) and the measurement noise covariance \(R\) are set. According to the laser range finder specifications [27], the standard deviation of the laser is 0.005m. Considering the error added by the leg detection system, specially the fact that the tracker might change the leg that it is tracking in some cases, we can estimate a new standard deviation for the whole detecting system. If the maximum distance between two legs of the same person is 0.4 m, an error of that magnitude can occur if the tracking system loses the original leg and starts tracking the other leg. That leads to a standard deviation of 0.405m, which gives a first estimation for the diagonal matrix \(R\) \((2 \times 1)\) : \(R(0,0) = R(1,1) = 0.405^2 = 0.164\). The values for the diagonal process covariance matrix \(Q\) \((4 \times 4)\) are harder to determine. A first guess based on the model, the maximum speed of a person, and the time between steps, sets \(Q(0,0) = Q(1,1) = 0.36\). The rest of the elements in the diagonal are set to be one order of magnitude smaller.

These initial set of values for \(R\) and \(Q\) was later refined experimentally until a good enough estimation of the state of the person was reached. The final expression for the matrices are shown in equation 4.9.

\[
Q = \begin{pmatrix}
0.020 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.020 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.015 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.015 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.00035 \\
0 & 0 & 0 & 0 & 0 & 0.00035
\end{pmatrix};
\quad R = \begin{pmatrix}
0.361 & 0 \\
0 & 0.361
\end{pmatrix}
\quad (4.9)
\]

\footnote{What it is called velocity is in fact \((dx, dy)\), which lacks of the time differential \(dt\) to be velocity. In order to obtain the velocity \((dx, dy)\) must be divided by \(dt\), being \(dt = T_s = 0.2s\)}
As it is obvious to see, the covariance error of the process was made lower than the covariance error of the measurement, which was slightly raised. The performance showed to offer a more smoothed estimated trajectory (sequence of estimations within time) than the first approach. This can be understood as an effect of the increase of the reliability of the process model in comparison to the measurement, that is implicit in the new relation among their error covariances.

Once the model used for the Kalman Filter has been set, the mathematical implementation in the system will be presented. The four components which implement the whole filter usage are presented in figure 4.7. The component in brackets is an external component (transformation) that makes sure that the previous estimation of the person’s position is expressed in the adequate reference system before using it to make the prediction. The motivations for this transformation are stated in the architecture chapter, and the details about this transformation model will be given in the next subsection.

The rest of the components contain the classic mathematical formulation of the Kalman Filter. The kalman predict determines a predicted position for the person, together with the covariance error of this prediction. The kalman correct component takes these two values and computes them together with the actual measured position of the person. Out of this comparison, an optimal estimation of the person’s position is output. The formal expressions used in the mentioned predictive and corrective components are presented in 4.10 and 4.11. The component kalman next step simply updates the previous estimation with the actual estimation. Finally, the kalman initialise component is used only once, when a person is detected, and
determines if the output from the Kalman Filter component will be the measured position of the person (in case the filter is not initialised) or the estimation of the position of the person (in case the filter has been initialised). The Kalman equations as they are used in these components are depicted in 4.10 and 4.11.

\[
\begin{align*}
\text{Kalman predict} \\
x_{k\text{ predicted}} &= Ax_{k-1} \\
P_{k\text{ predicted}} &= AP_{k-1}A^T + Q
\end{align*}
\]

\[
\begin{align*}
\text{Kalman correct} \\
K_k &= P_{k\text{ predicted}}H^T(HP_{k\text{ predicted}}H^T + R)^{-1} \\
x_k &= x_{k\text{ predicted}} + K_k(z_k - Hx_{k\text{ predicted}}) \\
P_k &= (1 - K_kH)P_{k\text{ predicted}}
\end{align*}
\] (4.11)

All in all, the Kalman Filter component represents thus a crucial tool in the tracker module, as it smooths the measured trajectory of the person and estimates the person’s velocity. This position and velocity are indeed two decisive variables in the rest of the modules of the system.

### 4.2.6 The transformation component

As introduced in the last subsection, the transformation module is in charge of the reference system adaptation for the previous estimation of the person, necessary while the robot is moving. This task is carried out using the information about the rotation and the translation made by the robot since the last time step. The resulted values are velocities and positions expressed in the actual reference system of the robot.

What this component addresses is a simple reference system change. Base change matrices can be used for this purpose, however, in this thesis a simple two steps transformation is done. Given a point \( P \) expressed in the \( R_{old} \) (reference system of the last time step), and the rotation and translation that relate \( R_{old} \) with \( R_{new} \) (actual reference system), \( P \) can be expressed in the \( R_{new} \) by rotating and translating the \( P \) coordinates in the \( R_{old} \) reference system. Figure 4.8 illustrate the steps of this transformation in a simplified, not real situation, in which rotation and translation have been exaggerated.

When applying these two steps transformation to the last estimation of the person, a difference must be made when handling \((dx, dy)\) and \((ddx, ddy)\). Unlike the position of the person, they are not fixed to the origin of the reference system, instead, they are bounded to the position of the person. Thus, they don’t have to be translated, as their translation is implicitly done in the translation of the person position.

The transformation module is also used to transform the last person measurement, in order to be able to relate it to the actual person measurement in some
The transformation component engages a simple but necessary computation process concerning reference systems in basic Cartesian geometry. However, as the translations and rotations considered are relatively small, the effects of these calculations are not as obvious as one could think. I.e., in 0.2 seconds (the time step in the algorithm), the robot moves at a speed that never makes the translation and rotation values too high. Naturally, this does not mean that the component is not useful, but that its effects are not noticeable at first glance.

In this section, a series of descriptions of the internal functionality of the modules that conform the tracker. In this way, how this module manages to carry out the functionalities described extensively in the architecture chapter was cleared up.

### 4.3 The follower module

This section presents the implementation aspects concerning the follower module. As explained in the architecture chapter, this module is the responsible for the human-robot motion coordination. In order to achieve this coordination, the context and the person velocity are interpreted, and a desired position is defined.
accordingly. Then the velocity (the desired velocity for the robot) that should take the robot to that position is calculated, considering the state of the robot and the person’s velocity. Finally, this desired velocity is translated to turn rate and speed commands for the motors. In the whole process, two extra considerations will be taken: probable obstacle collision, and probable visibility loss.

In the following, how this sketched process is actually carried out by the components inside the tracker will be explained.

4.3.1 The context interpreter component

This is the component in charge of the definition of the relative human-robot position in accordance to the context in which the following process takes place. As stated in chapter 3, the desired relative human-robot positioning is defined by two parameters \((\alpha, \rho)\) (see figure 3.4). The context is defined by context identifier (a single integer) input to the follower.

This component simply defines different sets of \((\alpha, \rho)\) depending on the value of the context identifier. In figure 4.9 the different set values are shown as a function of the context identifier. The context interpreter is an external parameter to the whole architecture. The context it self is interpreted by some external module whose design is not addressed in this thesis. This external module should output an integer whose value identifies the kind of context.

As an example, the value 2 of context identifier might identify a narrow corridor, or a door walk-through proximity. Values like 1 or 3 could identify unconstrained environments, or obstacles incoming in one of the sides of the person.

This is the way in which the follower module is able to react to a changing environment. The default rho and default alpha are set offline according to the user preferences, the type of environment, or the kind of application (in the case treated in this thesis, default alpha = \(-\pi/4\), and default rho = 0.7m). These default values were chosen as temporal values which might be refined when testing the application with real users.

4.3.2 The velocity estimator component

The velocity of the person that the tracker calculates is relative to the robot. In the approach presented in this thesis, the follower needs to know the absolute velocity of the person in order to generate the commands for the robot that sustain the accompanying behaviour. Thus, this component calculates the absolute velocity of the person according to the relative velocity of the person and the odometry information of the robot’s velocity.
Figure 4.9. Possible sets of \((\alpha, \rho)\) values according to the context identifier

The basic dynamics laws for accelerated reference systems describe in 4.12 the relation between the velocities of a moving body expressed in two different reference systems. The moving body is \(P\) and it has position and velocity \((r_{RA}P, v_{RA}P)\) in the reference system \(RA\) and a position and velocity \((r_{RB}P, v_{RB}P)\) in the reference system \(RB\). The reference system \(RA\) is still, while the \(RB\) is moving relative to the \(RA\) at a translational velocity of \(V_{RBRA}\) and an angular velocity of \(\Omega_{RBRA}\).

\[
v_{RA}P = v_{RB}P + V_{RA}P + \Omega_{RBRA} \wedge r_{RB}P
\]  

(4.12)

In the framework of the follower, the \(RA\) is the absolute reference system, \(RB\) is the reference system attached to the robot and \(P\) the person to be followed. In figure 4.10 a graphical representation of the problem is presented. Thus, the inputs for this velocity estimator component are the position and velocity of the person output by the tracker \((r_{RB}P, v_{RB}P)\) and the velocity of the robot according to the odometry information \((V_{RA}P, \Omega_{RA}P)\), turn speed and turn rate of the robot, respectively. Evidently, the output is \(v_{RA}P\).

The term "absolute" refers in this context to the global reference system, opposed to the local reference system. The global reference system is attached to the floor on which the robot and the human are moving, while the local is attached to the robot itself, and moves with the robot.

\[3\]
This component offers thus the absolute estimated velocity of the person that the rest of the components use in order to create the accompanying behaviour. The resulting velocity will never describe the real velocity of the person, however, it offers an fair enough estimation of it. Basically, this component modifies the velocity output by the tracker trying to undo the effects of the robot's movements in this measured velocity.

4.3.3 The set desired position component

This component sets the desired position for the robot according to the position of the person to follow, its velocity (the output from the estimate velocity component), the set pair of parameters \((\alpha, \rho)\)\(^4\), and the actual state of the robot.

The idea underlying this component is defining two different positions according to the state of the robot, and determine if there is a probable loss according to this desired position definition. The desired position \(\vec{a} = (x_a, y_a) = \vec{a}\) is the closest to the robot point at a distance \(\rho\) to the person, in case of approach state. In case of follow state, the position is defined relative to the velocity of the person. The point \(\vec{a}\) is placed at an angle \(-\pi/2 + \alpha\) to the velocity of the person, and a distance \(\rho\) to the person. In figure 4.11 a imaginary situation illustrates both choices.

Concerning the probable visibility loss, the component addresses the detection problem analysing the desired position previously defined, the position of the person,

\(^4\)The units used for the \((\alpha, \rho)\) parameters are meters and radians, respectively
Figure 4.11. The two different definitions for the desired position ($\vec{a}$) of the person, according to the state of the robot. The figure on the left represents the $\vec{a}$ defined in the state *approach*, and the figure on the right the $\vec{a}$ defined in the *follow* state.

and its velocity relative to the robot. Two different configurations will be identified as leading to a probable visibility loss:

- the person is in an angular region set close to the limits of the visual field of the laser range finder and its relative velocity to the robot is inside an angular region set in the third and fourth quadrant of the robot’s reference system. In such cases, the person is considered to be close to get out of the laser angular range. In figure 4.12 these two thresholds are graphically shown.

- the desired position vector and the person’s position vector define an angular region of an amplitude bigger than a threshold. This situation implies that trying to reach the desired position will probably make the person get out of the visual field of the laser. Figure 4.13 illustrates this angle definition and the threshold itself.

In both cases, a special action is required from the robot that assures the visibility of the person in the next time step. This action will be demanded by raising the *out of sight* flag, and will be carried out by the *set desired velocity* and *command motors* components.

---

5The velocity of the person relative to the angle range limits of the laser is what matters in this case, so it is the relative velocity of the person that is used in this case.
Once defined the desired position where the robot should be located, and distinguished any probable case of visibility loss, the velocity that the robot should acquire can be determined.

### 4.3.4 The set desired velocity component

This is the component that defines the velocity vector that the robot should have in order to perform the accompanying behaviour. According to the velocity and position of the person, the state of the robot, the desired position and the *out of sight* flag, this component comes up with a desired velocity vector that is output to the *command motors* component.

The desired velocity vector is a direct function of the desired position and the velocity of the person. The robot state and the *out of sight* flag determine this function. The main idea of the function is to make the desired velocity be the sum
of the distance to the desired position ($\vec{a}$) and the velocity of the person itself, as proposed in [26]. This formulation should tend to make the robot reach a certain point with a certain velocity.

Some experiments showed that the vector $\vec{a}$ needs to be scaled before being added. I.e., if the desired position is 5m away, the robot will try to go 5m/s if no scaling is used, and won’t slow down fast enough when reaching the point. In this way, the scaling factor provides the sum with a factor proportional to the desired position vector, as shown in equation 4.13.

$$v_{desired} = v_{person} + \text{scale\_factor} \cdot \vec{a} \quad (4.13)$$

The scaling factor is a function of $\vec{a}$ and is represented by the slope in the graph in figure 4.14. The sizes of the regions and the values for the slopes were experimentally tuned.

![Figure 4.14. Relation between the module of the desired position vector and the velocity associated to this factor](image)

The state of the robot and the out of sight flag play an active role in the function definition as well. When the state is approach the $v_{person}$ won’t be considered in equation 4.13, so as to make the robot simply approach the desired position (see 4.15). When the out of sight flag is up, the $v_{person}$ is despised as well. In addition, the $\vec{a}$ vector is substituted by the person’s position (see 4.16), and the scale\_factor incremented in its value so as to make the robot more sensitive to distance and thus react faster to the event. In that way, the robot is forced to head towards the person. In a coordinated way, the command motors component, will also detect the flag and block the forward velocity of the robot, only allowing it to turn.

All in all, there are three possible functions that define the desired velocity vector. The three of them are shown in equations 4.14, 4.15 an 4.16.
case follow
\[ v_{desired} = v_{person} + \text{scale\_factor} \cdot \vec{a} \quad (4.14) \]
case approach
\[ v_{desired} = \text{scale\_factor} \cdot \vec{a} \quad (4.15) \]
case out of sight
\[ v_{desired} = \text{scale\_factor\_incremented} \cdot r_{person} \quad (4.16) \]

The flow chart of this component is presented in figure 4.15. When the component starts, the module of \( \vec{a} \) is analysed in order to define the scaling factor. The out of sight flag is then checked, if it is not activated, the component will use the functions 4.14 or 4.15 according to the state. In case the flag is up, the component will use the equation 4.16. Finally, the vector calculated is output to the next component, the command motors component.

Figure 4.15. Internal behaviour of the component set desired velocity

On the whole, the desired velocity vector defines how to reach the desired position according to the state of the robot, and considering probable visibility loss cases.

4.3.5 The command motors component

This component is the responsible for the creation of the turn rate and speed commands of the robot as a function of the desired velocity generated by set desired
velocity, the *out of sight* flag, and the *obstacle descriptor*.

The main idea is to generate turn rate and speed commands that tend to make the robot acquire the desired velocity. Once the turn rate and speed commands are generated according to the desired velocity vector, they are modified considering the *out of sight* flag and the *obstacle descriptor*. Figure 4.16 shows the working basis of the component.

In the following, a detailed explanation of the schema in figure 4.16 is presented. First, turn rate and speed are calculated to make the robot have the heading defined by the argument of the desired velocity vector, and move with a speed equal to the module of this vector. As described in the architecture chapter, the robot interface offers only speed and turn rate control. Thus, the speed can be set to the module of the desired vector (see 4.17), and the turn rate must be set through a system that makes the robot try to acquire the heading of the desired vector. The chosen system for this purpose is a PID. The error is defined as $\angle v_{desired} - \pi/2$, and input into the PID, which tries to reduce the error to zero using the turn rate command as the control signal (see 4.20).

<table>
<thead>
<tr>
<th>Speed command</th>
<th>definition</th>
<th>(4.17)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>speed</em></td>
<td>$</td>
<td>v_{desired}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Turn rate command</th>
<th>definition</th>
<th>(4.18)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_k$</td>
<td>$\angle v_{desired} - \pi/2$</td>
<td></td>
</tr>
<tr>
<td>$I_{e_k}$</td>
<td>$I_{e_{k-1}} + e_k$</td>
<td>(4.19)</td>
</tr>
<tr>
<td><em>turn rate</em></td>
<td>$K_P e_k + K_D (e_k - e_{k-1}) + K_I I_{e_k}$</td>
<td>(4.20)</td>
</tr>
</tbody>
</table>

The values for the $K_P$, $K_D$ and $K_I$ were calculated in two steps. First, a mathematical model from the system was built according to system identification experiments. A PID controller was then designed using computer aided analytical methods \(^6\). Finally, the controller was used in the real system and the $K_P$, $K_D$ and $K_I$ were re-tuned until the performance of the controller was acceptable.

After the turn rate and speed are defined, their values are limited to the physical constraints of the robot, expressed in terms of maximum turn rate and speed supported. In order to avoid wind-up problems \(^7\) in the PID, the integral term is reset in case the turn rate is limited.

The turn rate and speed generated are modified then according to the *obstacle descriptor*. This variable is output by the *obstacle detection* component into the *command motors* component, and gives information about the obstacle distance and approximate angular location, if such obstacle was detected. The distance is given in meters, and the approximate angular location is specified with an identifier of the angular region where the obstacle is located. As explained in the *obstacle descriptor*.

---

\(^6\)The tool used for this purpose was the Control Toolbox from Matlab ©
detection component, the space around the robot is divided in three angular regions (see 4.18). In addition, the command motors component parts the same space in two rings centred in the robot. The figure 4.17 shows the resulting division of the space around the robot.

If an obstacle is detected, a corrective action is applied on the calculated turn rate and speed values. These corrections are only applied if the robot is not stopped or the person is not inside the shaded zone in figure 4.17. The corrective action consists on different modifications made on the turn rate and speed commands depending on the region where the obstacle is detected. If the obstacle is in region 1, the turn rate command is modified accordingly. If the obstacle is in region 2, the speed command is set to zero. The subregions labelled 3 require a more drastic modification of the turn rate command. Equations 4.21 to 4.25 depict the different modifications made on the commands according to the location of the
Figure 4.17. Five different regions where obstacles can be located determine how to react to them.

obstacle.

Region n.2 : Obstacle in the front
\[
\text{speed} = 0 \quad (4.21)
\]
Region n.1.1 : Obstacle on the right
\[
\text{turn rate} = \text{turn rate} + \text{corrective constant}_1 \quad (4.22)
\]
Region n.1.2 : Obstacle on the right, close
\[
\text{turn rate} = \text{turn rate} + \text{corrective constant}_2 \quad (4.23)
\]
Region n.3.1 : Obstacle on the left
\[
\text{turn rate} = \text{turn rate} - \text{corrective constant}_1 \quad (4.24)
\]
Region n.3.2 : Obstacle on the left, close
\[
\text{turn rate} = \text{turn rate} - \text{corrective constant}_2 \quad (4.25)
\]

Once the correction concerning obstacle avoidance is done, the component checks the \textit{out of sight} flag and acts consequently. Only if the flag is up, the speed command is set to zero, making the robot stop on the dot, and turn in order to head the person, as it was explained in the component \textit{set desired velocity}.

Finally, the generated speed and turn rate commands are output to the \textit{main module}, which will forward them to the \textit{robot interface}, making the robot motors react accordingly.

5 Assuming in this way that, according to the desired velocity definition, the robot will never hit the target that it is following.
All in all, the steps described above are aimed to transform a desired velocity into turn rate and speed commands for the motors of the robot, so as to make the robot acquire this precise velocity. In addition, the defined commands are subsequently modified in order to take into account obstacle avoidance and probable visibility loss.

4.3.6 The obstacle detection component

This component is in charge of the identification of possible obstacles. It receives the actual laser range data set, and outputs an obstacle descriptor that specifies if an obstacle has been found, and if so, describes the position of the obstacle.

The component performs a simple identification of obstacles according to a minimum safety distance to the robot. Whatever is closer than this distance to the robot is considered to be an obstacle. In that case, an obstacle is notified, and its position relative to the robot is described.

The laser range finder data set determines the distances to all possible obstacles. Thus, the minimum value of the data set is calculated in each time step. If this value is lower than a threshold, an obstacle is notified. The position of the obstacle is specified with its distance to the robot and an identifier which determines the angular sector (see figure 4.18) in which the obstacle was found.

![Figure 4.18. The three different regions where obstacles can be detected by the obstacle detection component](image)

As explained before, it is the command motors component which decides how to react to the obstacle detected. Thus, this component simply detects and describes the obstacle situation.

---

8 According to the assumptions stated in the beginning of the architecture chapter
The component follower is the component that determines the commands for the robot’s motors in order to create the accompanying behaviour. In this section, a complete description of the implementation details of the components involved in this particular task has been presented.

4.4 The main module

As explained in the architecture section, the main aim of this module is coordinating the rest of the modules in the architecture. The module coordinator inside of the main module engages the coordination itself, while additional components offer a screen monitoring interface and communication with an external module that specifies the context.

The implementation part of this module is far more simple than the implementation of the modules described above. The module coordinator is barely a sequential series of instructions for the rest of the modules, that was completely overseen in the architecture chapter. The screen interface uses existing graphical libraries, and the context reader basically communicates with a standard input device. Therefore, no more details will be given on the module coordinator. Concerning the other two components, a brief explanation of the tools used for their implementation will be given.

4.4.1 The screen interface component

This component is meant to offer through a graphical display useful information about the performance of the algorithm while the robot is running. It should ease the interaction robot-human by providing feedback about the general state of the robot to the person being followed.

For that purpose, the screen interface makes use of the X11 libraries, developed by the X.org Foundation. These libraries offer a versatile write access to a screen device working in Linux, which is the system in which the robot application runs.

The information that is considered to be useful for the monitoring of the robot functioning is listed below:

- The complete laser range data set in each time step offered by the robot interface
- The angular region in the scan where the tracker looked for the legs of the person
- The measured position of the person offered by the tracker
- The estimated position and velocity of the person offered by the tracker
- The absolute estimation of the person’s velocity calculated by the follower
The chosen values for \( \{\alpha, \rho\} \), retrieved from the follower.

The desired position and desired velocity defined by the follower.

When detected, the angular region where the follower detected the obstacle.

The state of the robot, as specified by the state sequencer.

The battery status, proportioned by the robot interface.

The actual time it takes the algorithm to be updated, as measured in the module coordinator.

All these figures offer a fair complete status information of the robot. Figure 4.19 shows the whole interface while working.

![Figure 4.19](image_url)

**Figure 4.19.** A screenshot from the interface offered by the screen interface component.

The robot itself lacks of a screen, so this interface runs on a wireless terminal of the robot. Therefore, it offers no direct feedback to the user, unless the terminal is transported by the user. In addition, some of the information displayed is only useful from the point of view of a developer, not an end-user. However, the information
about the state of the robot is understandable and helpful even for unexperienced users. Any kind of user finds it valuable to know if the robot has detected him, if it is waiting for the bumper to be pressed, or if it has considered the service finished. Future improvements should implement part of this information (the state information, specially) into a different interface such as a speech synthesiser.

4.4.2 The context reader component

This component engages the communication with a module external to the algorithm that specifies the kind of context in which robot and human are located. The kind of context is specified, as explained in the context interpreter section, by a single integer.

The external module is meant to analyse the context in some way and output a context identifier that the context interpreter can recognise. The external module is nonexistent, therefore, a simple keyboard input simulates the output of the context specifier. A keyboard connected to a wireless terminal of the robot is used for such purpose. Four different values for the context identifier are considered, and input from the keyboard. In order to access this extra device without interfering the execution of the main algorithm that control the robot, threads are used. The main module launches a thread when the algorithm starts. This thread scans the keyboard continuously, captures keyboard inputs, and writes the corresponding value in the context identifier variable, that is later sent to the follower module.

As the real module that defines the context identifier is simulated with keyboard inputs, the real purpose of the context identifier is not fully accomplished. However, the analysis of the context in order to define a context identifier is out of the scope of this thesis. The ”simulated” context specifier simply offers a way to test the context dependency of the algorithm designed.

Above lies the description of the components that conform the main module. The focus has been set on how they manage to implement the functionalities depicted in the architecture chapter.

4.5 The state sequencer module

This section offers a closer look to the state sequencer that reveals the nature of its algorithm, and its implementation’s details. As shown in the architecture chapter, the state sequencer consists of a single component named state machine. In the following, a description on its implementation is presented.
4.5.1 The state machine component

The "heart" of the state sequencer is a component labelled state machine. The functionality of component was presented in the architecture chapter. In this chapter, a description of the finite state machine that conforms the state machine component will be posed. The states, the transitions, and the triggers in which the transitions depend will be detailed in the following. The figure 4.20 shows the structure of this finite state machine.

![Figure 4.20. The state machine inside the state sequencer module](image)

To start with, the set of triggers will be presented, later the complete set of transition conditions for each state will be explained. Below, the triggers that the main module sends to the state sequencer are listed:

- **estimated person velocity** → output from the tracker module into the main module.

- **estimate error covariance** → output from the tracker module into the main module.
- **number of misses** → internal variable from the *main module* which counts the number of consecutive times that the person has not been found in the laser range data, i.e. the number of times that the tracking system was unable to find the legs of the person in the expected angular region.

- **bumper readings** → output from the *robot interface* module into the *main module*.

- **person measure** → internal variable from the *tracker* module that represents the measured, not estimated, position of the person. It will be zero until a person is detected.

- **previous** and **actual absolute person velocity** → internal variables from the *follower* module that represent the estimated absolute velocity of the person to be followed, according to the velocity of the robot.

Some of these triggers are internal variables from the modules *tracker* or *follower*, and are not explicitly available in the *main module* according to the diagrams presented (*actual* and *previous absolute person velocity* and *person measurement*). These variables are indeed available in the *main module*, however, for simplicity matters they were not included in the diagrams.

The states and their transition conditions are explained below:

**Start.** A transition is made towards state:

- **Still** → if any bumper has been pressed and either the transition to the start state was due to the person being still for a long time or the previous state was start. I.e., if the person was still for a long while, the robot will assume that the service is over and will wait for the starting signal from the user (bumper pressing) to start up again. If the robot has just been started, the previous state is start, and thus, the bumper pressing will also be required in order to start the service.

- **Still** → if the transition to the start state was due to a tracking loss, and a safety time has passed since the start state was adopted. I.e., when a tracking loss happens, the robot will wait in start for a very short time. Afterwards, it will automatically switch to the still state and look for movement in the laser range data in order to find the person again. The safety waiting time is set in order to assure that the robot is still before it starts looking for movement in the laser scans.

- **Terminate** → if the OFF bumper is pressed. I.e., if the robot has not initiated any service and a particular bumper is pressed, the robot will assume that its service is no longer required, and thus, the application will be terminated.

- **Start** → if none of the conditions above are fulfilled. I.e., the robot will wait indefinitely in this state until some bumper is pressed.
Still. A transition is made towards the state:

Pre approach  $\rightarrow$ if a person has been detected (person measure different from zero). I.e., the robot will go to the pre approach state when a person has been detected (movement and legs detected) by the tracker module.

Still  $\rightarrow$ if the condition above is not fulfilled. I.e., the robot will stay in this state until it detects movement and gets a measurement of the position of the person.

Pre approach. A transition is made towards the state:

Approach $\rightarrow$ if the estimate error covariance is low enough and a safety time has passed after the detection. I.e., after initialising the Kalman Filter (in the pre approach state), the robot will wait until the tracking system offers some certainty on its estimation and a safety time has passed after the detection of the person. In this way, the robot will not move (change to approach state) until the estimation of the position of the person is good enough. The error covariance is a significant value of the quality of the estimation, and the safety time avoid certain transitory effects in the first steps of the Kalman Filter.

Start $\rightarrow$ if the tracking system failed to track the target (a certain number of misses happened). I.e., the robot will stop whenever the tracking system misses the person a number of times. In this way, it should be able to find the person according to movement information.

Pre approach $\rightarrow$ if none of the conditions presented above is fulfilled. I.e., the robot will not start moving until the estimation offered by the tracker is not good enough.

Approach. A transition is made towards the state:

Follow $\rightarrow$ if the person is walking straight for some time. I.e., the robot will start to consider the $\alpha$ parameter in the following behaviour only if the trajectory of the person makes it possible. The walking straight definition is given in terms of the relations between the previous and actual absolute person velocity, and the module and orientation of the actual absolute person velocity.

Start $\rightarrow$ if the tracking system failed to track the target or if the person was still for some time. I.e., the robot stops whenever the tracking system misses the person a number of times, or the person was still during a time of a magnitude of 10 seconds, approximately. As explained before, in the first of the cases, the robot should be able to find the person again according to movement information. In the second of the cases, the robot will consider the service to be over and wait until a bumper is pressed to start it again.
Approach → if none of the conditions above is fulfilled. I.e., the robot will try to approach the closest point at a distance \( \rho \) to the person if his trajectory is not "straight" enough, no tracking loss happens, and the person is not still for a long time.

**Follow.** A transition is made towards the state:

*Approach → if the person slows down significantly. I.e., when the actual absolute person velocity is not high enough, the robot will no longer consider the \( \alpha \) parameter in the following behaviour. That will make the robot head the person when he is about to stop, and eventually, stop the robot at a distance \( \rho \) to the person once he is completely stopped. In future improvements, the following behaviour could be held even when the person has stopped. However, in this level of implementation, it is dangerous to keep the follow state when the person slows down, because the estimated actual absolute person velocity when robot is approaching a still human is not reliable.*

*Start → the conditions and the consequences are identical to the ones explained above, in the approach transitions.*

*Follow → if none of the conditions above is fulfilled. I.e., the robot will move in order to reach a relative position defined by \( \{\alpha, \rho\} \), while the person keeps a minimum velocity, no tracking loss happens, and the person has not stopped for a long time.*

**Terminate.** No transition is made towards any state. This is the exit state of the state machine. The main module will simply terminate the application when detecting that the state is terminate, as explained in the main module subsection.

Above, the way in which the states of the robot are coordinated was presented. The complete formal description of the finite state machine that underlies the state sequencer was presented.

### 4.6 The robot interface module

The robot interface, is the intermediary module between the robot and the rest of the system. As posed in the architecture chapter, it is a semi-detached module from the whole architecture. The implementation of the module was not addressed in this thesis, Player, an existing software, was used instead.

In this way, the implementation details that are offered in this section are no more than an overview of the internal architecture of the features of this software that are used in the thesis.
4.6.1 The player interface component

Figure 4.21 shows the structure of the player interface as it is used in the scope of this thesis.

![Diagram of the architecture of the robot interface]

The component is subdivided in four different proxies, each of them associated to a different device in the robot, plus a base client that communicates all the proxies with the robot itself. A closer perspective to these components will be offered next:

- **Player client.** This is the basic communication component with the robot. It communicates with the robot at a set frequency, sending and retrieving packets with requests and commands for the pertinent devices in the robot. The initialisation of this component must be done when the application starts, and sets the frequency in which data will be received and sent to the robot. This frequency is a crucial parameter, as it is in any discrete time control system [20]. The module coordinator should coordinate in time with this player client module.

- **Laser proxy.** This is the server that offers the laser range data sets retrieved from the robot in each sample time. The laser range data is offered in the
shape of two vectors, one containing the distance information and the other containing the angles corresponding to each distance measurement. The initialisation process is also a crucial step to be made before the robot starts to use the information retrieved. The laser must be configured to scan a range of $180^\circ$, with a certain angular resolution distance resolution.

- **Position proxy.** This server offers to the main module the possibility to command the motors of the robot and to read or reset the odometry information of the wheels. The proxy offers control of the speed and turn rate of the robot, and information about the heading, position, speed, and turn rate of the robot. This odometry information is calculated by the robot according to the movement of the wheels since the last odometry reset. In the architecture presented in this thesis, the odometry is only used to know the velocity of the robot and to calculate the position and heading change from one frame to the next. Thus, the odometry is reset in each time step.

- **Bumper proxy.** This server communicates with the bumpers in the robot. It offers the possibility to read from one particular bumper or from all of them. This allows the definition of a OFF bumper, whose function is explained in the state sequencer section in the implementation chapter.

- **Power proxy.** This server offers readings from the battery level of the robot. These measures do not play any active role in the architecture, they are merely informative figures to be displayed in the monitoring screen supported by the main module.

The player interface itself offers many more capabilities, such as access to different other devices, multirobot control, or simulation interface. This thesis work has included the usage of the interface that the Player provides to control laser, motors, batteries and bumpers, plus the simulation features. However, future work could include other device interfaces offered by the Player such as the speech interface, which makes use of the Festival speech synthesis system.

Implementation details for all the modules used in the architecture have been offered in this chapter. The tools used in this implementation, such as PID controllers, threads programming, Kalman Filters, X11 graphical libraries, finite state machines, basic Cartesian geometry or basics dynamics laws, have been described, and their application in the system depicted.
Chapter 5

Experiments

The performance of the system is presented in this chapter. The analysis of the performance has been divided into different sections, each of them focused in different aspects of the overall system behaviour.

First, a brief analysis of the performance of the tracking system (the tracker module) is presented. The performance analysis of the follower module follows. Finally, a description of the system as a whole while it is interacting with different people is presented, in order to analyse the usability and effectiveness of the robot as an end-user oriented device.

In all the figures presented, distances are expressed in meters, angles in radians, and time in frames.

5.1 The tracking module performance

This section addresses the performance of the tracking system. The tracking system produces estimations of the position and velocity of the person according to laser range finder data. Though the tracking system is not the main issue on this thesis, it influences the performance of the follower, as the follower bases its calculations on the estimations provided by the tracker.

The overall behaviour of the tracker will be presented, ranging from the movement search phase to the tracking phase, and considering tracking losses. Individual experiments were made to illustrate the performance of the system, and a reduced group of people were tested in order to estimate the effectiveness of the tracking.

The figures 5.1 and 5.2 compare the estimated position and velocity, with the measured position and velocity of the person. The velocity is not measured, but for comparison matters, it is deduced from the differences between subsequent measured positions. The estimation is expressed in $x$ and $y$, coordinates in the reference system that is attached to the robot. A more intuitive representation of the estimations is presented in figure 5.7.

\[ T_s = \text{number of frames}, \quad T_s = 0.16s \]

1 The time in seconds would be calculated as follows: $T_s = \frac{\text{number of frames}}{60}$.
The error covariance of these estimations during the same time period is stabilised in a constant value 0.23.

Figure 5.3 shows the covariance evolution when the detection and tracking start. The uncertainty of the first estimations is shown in the higher value of the
covariance of the error in the first two seconds, approximately, after the Kalman Filter initialisation. When a measure is detected (first value different to zero), the Kalman Filter is initialised, and the first estimation different to zero takes place. It is also evident that the Kalman Filter is started in the pre approach state.

A tracking loss produces almost the same sequence of events in the tracker. Figure 5.4 shows a tracking loss followed by the re-start of the tracker module.

A group of four people where tested with the system in order to estimate the effectiveness of the tracking system. Figure 5.5 shows the relation between the number of misses and the total tracking time for each of the people tested (expressed in number of tracking misses in 100 frames).

As explained in previous chapters, when the tracking system loses a person, the robot is stopped, and the tracker starts looking for movement. Figure 5.6 shows the recovery times for the sample group. Recovery times are measured from the time that the track loss happens, until the person is detected again.

The main cause of tracking losses is related to detection problems. In some cases, the considered minimum size of the leg of the person is sometimes too small for the real size of the leg (when the person is approximately further than 3.5 meters from the robot). However, the mean values for tracking loss (0.9 misses/100frames) and recovery time (8 frames) show that the tracking system works acceptably well.

A plot from the estimated trajectory of the person and the measured trajectory shows the smoothing qualities of the Kalman Filter 5.7. Again, the x and y coordinates are the coordinates of the reference system attached to the robot (the
The degree of smoothness of the estimated trajectory depends on the values set for $R$ and $Q$. The trajectory could have been smoothed even more by making $Q$ smaller relative to $R$. However, this showed to produce mis-detections in some cases. According to the line detection and line choosing methods, the system could
Figure 5.6. The mean recovery time after tracking loss for each subject tested, and the mean value for all the subjects.

Figure 5.7. Estimated and measured trajectories of the person.

consider that a line which is not a leg (a piece of wall or a table leg), but is closer to the predicted position of the person, is the person itself. In addition, the stationary
value for the covariance error of the estimation is not as low as it should be. Though retuning the $Q$ and $R$ values could make this value decrease, the estimated position and velocity are good enough to be used by the follower.

Nevertheless, it is worth mentioning that, as the position and velocity of the person are considered to be one of his/her leg’s velocity, the velocity estimations might differ slightly from the real velocity of the person. Once again, the detection system could be improved in order to consider both legs, but the performance of the actual tracking system is acceptable, and, in addition, is not the main point on this thesis.

5.2 The follower module performance

The follower is the module responsible for the human robot motion coordination. As posed in the previous chapters, two main ways of functioning can be distinguished in the follower, depending on the state of the robot: approach and follow.

This section offers an overview of the performance of this differentiated behaviour, plus a performance study of the relative positioning system implemented in the follower.

5.2.1 Approaching the person

The absolute velocity of the person draws the line between approach and follow behaviours. The approach behaviour should be kept in case this velocity is not high enough, or does not keep a certain orientation. Figure 5.8 shows the robot as it follows the person in the approach state. In this case, the trajectory of the person is not straight enough (as defined in the implementation chapter), so the robot just tries to keep a determined distance to the human.

All the experiments are made with a distance $\rho$ of 0.7m, the $\alpha$ value, according to chapter 4, is irrelevant in the approach state.

Figure 5.9 represents the relative positioning human robot within time. The distance to the person tends to be $\rho$, while the orientation simply reaches $\pi/2$ after some time, as the robot tries constantly to keep the person in the centre of its visual field.

It is interesting to identify in the graph 5.9, the frame numbers in which distance and orientation are closer to the desired values. Those points represent the situations in which the person is still or almost not moving, thus the robot has time to reach the position and orientation desired. Frame number 100 in 5.9 and 5.8 illustrate this situation. While the person is moving, the robot can not reach the desired $\rho$ distance to person, but stays around 1.5m from the person. The $scale\_factor$ parametre in equation 4.15 could make the robot reach the distance $\rho$ faster. But a robot moving so fast to a distance 0.7m to a person proved to be threatening for the person itself.
**Figure 5.8.** Robot and human trajectories, being the robot in *approach* state

**Figure 5.9.** Relative human robot distance and orientation, being the robot in *approach* state
The definition of the desired position and desired velocity is crucial in the behaviour of the following system. In figure 5.10 the module and angle of the desired position is shown, together with the module and angle of the relative position human robot. It is obvious to see the constant relation between them: the desired position module is the distance to the person minus $\rho$, and the angle is the same as the person’s. As expected, this definition makes the robot head the person, and try to reach a point at a distance $\rho$ of the person.

![Graphs showing desired position and desired velocity.](image)

**Figure 5.10.** Definition of the desired position vector according to the position of the person

In figure 5.11 the desired velocity is represented together with the desired position. In this case, the desired velocity is proportional to the desired position, according to equation 4.15.

The general behaviour of the follower in the approach state is acceptable. The desired position is properly defined, and the desired velocity calculated accordingly. Equation 4.15 proves to tend to place the robot at a distance $\rho$ to the person.

### 5.2.2 Following the person

The follower behaviour (follow state) starts whenever the trajectory of the person is straight enough, as explained in page 62. This section tests first the transition from approach to follow, and afterwards, the effect of context changes within the follow state.
Figure 5.11. Definition of the desired velocity according to the desired position

In all the experiments, a value of \((0.7\ m, -\pi/2)\) is set for the parameters \((\rho, \alpha)\). However, context changes eventually make these values change, as it is explained further on in this section.

From approach to follow

Figure 5.12 illustrates the situation in which the follow state is started. The trajectories of robot and person are plotted in order to give an idea of the kind of trajectory that started the follow state, and differentiate it from the trajectory seen in the previous case (figure 5.8).

According to the design of the system, the robot should try to reach a relative position to the human. This position is defined by the data set \((\alpha, \rho)\). For a value such as \((0.7, -\pi/2\text{rad})\), the relative position and orientation of the robot relative to the human are presented in figure 5.13. The robot maintains the relative orientation to the person, but can not reach the position behind the person. However, it succeeds in keeping up the velocity with the person, so that the distance between human and robot does not grow bigger.

Figure 5.14 shows the progression of the velocity of the person and the robot with the time. The robot’s velocity reaches its maximum \((0.5\text{m/s}^2)\) before reaching the desired distance to the robot. Unless the person walks slower, the robot will never keep up with him. That obliges the user to limit his speed when using the robot. In addition, the calculated module of the velocity of the person is higher.
than the velocity of the robot during the follow state. That should make the distance person robot grow with the time. As this distance is not growing according to figures 5.13 and 5.12, it is the estimated velocity which might not be very accurate.

Another example performs a similar behaviour. The \((\alpha, \rho)\) parameters are now set to \((0.7m, -\pi/4 rad)\). Figure 5.15 shows the trajectories described by the robot and the human. According to the \(\alpha\) angle, the robot starts turning slightly to its right in order to keep the person on its left. However, once reached the desired angular orientation relative to the human, it finds difficulties to turn slightly to the left so as to match its direction with the person’s direction. It is only when the person slows down, making the robot go back to approach state, that the robot turns definitely to the left. Reasons for this behaviour are outlined at the end of this point.

The relative position human robot plotted in figure 5.16 shows how far from the ideal behaviour the robot’s performance is. The position is never reached, though the robot manages to keep up with the person’s velocity.

A significant matter in the following algorithm presented in this thesis is the

\footnote{The maximum velocity of the robot, according to its manual \cite{1}, is 0.9m/s. The default configuration limits the velocity to 0.5m/s. This limitation can be overridden by accessing low level configuration routines on the robot}
desired position definition. The desired position definition should change significantly when switching from approach state to follow state. Figure 5.10 shows how the desired point definition has the same argument as the person. When the follow state is started, the desired point definition starts to differ from the person’s argument (see figure 5.17). This means that the robot no longer determines a desired position on the imaginary line that links robot and human, just like it does in the approach state. Instead, the new desired position lies slightly shifted to the right of the robot compared to the position of the person (positive angles decrease from left to right in the reference system of the robot). That should make the robot turn right in order to reach this desired point, and thus, leave the person on its left, according to the $\alpha$ value $-\pi/4$ rad.

The tendency to adopt the desired relative angular position in the follow state is considerably slow. Among the possible reasons, the following can be pointed out:

- in the equation 4.14, posed in page 52, the weight of the term assigned to the desired position is not significant enough, when compared to the term related to the velocity of the person. In fact, as it will be explained later, the argument of the desired velocity does not change drastically when entering in follow state (see figure 5.20). That could mean that in the follow state, the positional terms are shadowed by the term that considers the velocity of the person.
Figure 5.14. Comparison of the velocities of the person and the robot in follow state, with a \((\alpha, \rho) = (0.7 m, -\pi/2 \text{rad})\)

- the motion control model, with detached speed control and turn rate control, does not allow the robot to describe the necessary trajectories to adopt the certain angular relative position. I.e., the motion model is deficient.

This problem described above is revisited in the next section, as it affects also the experiments in the next section.

**Changing the context while following**

The architecture chapter explains that the context adaptability of the following algorithm is achieved thanks to a set of \((\alpha, \rho)\) parameters dependant on a context identifier. The context identifier is supplied by keyboard inputs, which emulates a real, changing context. The adaptability to changes in the context identifier is only supported by the follow state.

This section shows the results of a variable context identifier while the robot is in follow state.

Figure 5.18 presents the trajectory of robot and human in this situation. The context identifier changed from 0 to 3 in the frame number 111. According to figure 4.9 in page 47, this changes the \(\alpha\) value from \(-\pi/2\) to \(-\pi/4\). The robot should then try to occupy the space on the right side of the individual.
Figure 5.15. Robot and human trajectories, the robot being in follow state, with a $(\alpha, \rho) = (0.7m, -\pi/4\text{rad})$

Figure 5.16. Relative human robot distance and orientation, the robot being in follow state
Figure 5.17. Definition of the desired position vector

Figure 5.18. Robot and human trajectories, the robot being in follow state
The figure shows that the robot slowly tends to occupy the space on the right of the person. The robot, as in previous experiments, is able to keep up with velocity, but shows poor performance in direction control.

In figure 5.19 the **relative position** and angle of the robot relative to the person is shown. The problem explained some lines above is more evident in this graph. Taking into account that the trajectory of the person is not completely straight (see figure 5.18), the speed with which the robot approaches the new desired angular position is rather low.

![Distance Control](image1)

**Figure 5.19.** Relative human robot distance and orientation, the robot being in follow state

A closer look to the desired velocity (see figure 5.20) can offer some hints on the cause of this problem. In order to make the robot leave the space behind the subject and drift to his right side (in this particular case), the angle of the desired velocity vector should decrease. The context changes in frame number 111, and the angle of the desired velocity suffers no significant changes. This supports the first of the two possible causes mentioned in 5.2.2.

A different experiment with a similar configuration to the one presented above corroborates the reasons proposed in page 77 for this poor performance in the relative positioning of the robot. In this case, the context changes from value 0 to value 1, which changes the $\alpha$ parameter from $-\pi/2$ to $-3\pi/2$. Figure 5.21 represents the trajectory of both human and robot during the experiment. The context change is made in frame number 33.
Figure 5.20. Definition of the desired velocity vector

Figure 5.21. Robot and human trajectories, the robot being in follow state
The relative position study shown in figure 5.22 reveals that the system is hardly responding to the context change. Figure 5.23 represents one the main responsible for the behaviour of the robot: the desired velocity vector. Once the follow state is reached (frame number 55), the module of the desired velocity is incremented, and its argument grows slightly bigger than the person itself. The argument of the desired velocity determines how the robot it turns. The module increment is enough to make the robot acquire a similar speed to the human, however, the change in the argument is not significant enough to make the robot turn and acquire the desired orientation relative to the person.

Again, the problem is that the robot is not turning enough to acquire the new relative position. The argument is not significantly higher than the person’s argument, so the robot-person relative turn is not significant either. Only a noticeable change on direction of the velocity of the person (see the end of the person’s trajectory in figure 5.21) makes the desired velocity argument change significantly. This supports again, that, according to the reasons stated in page 5.2.2, the velocity of the person is more substantial in the definition of the desired velocity than the desired position.

![Distance Control Graph](image1)

**Figure 5.22.** Relative human robot distance and orientation, the robot being in follow state

All in all, the performance of the follower module is not as good as it was expected according to the design. The approach state fully accomplishes its aim, i.e. it drives the robot to a position approximately $\rho$ meters close to the person. The out of sight flag generated avoids visibility loss successfully, modifying the
Figure 5.23. Definition of the desired velocity vector

commands output to the robot. The forward velocity is properly tuned to fit the person’s velocity, thus keeping a constant distant to the person. However, the follower module lacks of a proper turn control that can make it reach the desired angular position relative to the human. In future improvements, possible solutions to this problem could be drawn from the reasons outlined in 77.

5.3 User oriented performance

The performance of the decisive components in the robot have been analysed as partly independent components of the whole system in the previous sections. In this section, the overall performance of the design and implementation will be analysed as an end-user product.

The main aim of the algorithm is to create a following behaviour that can be of use in a service robot. A simple task, that involves a following behaviour, is defined in this section as the service to be provided by the robot. The performance of the algorithm is measured according to how this task is carried out. In this way, the effectiveness of the algorithm as a user oriented tool can be studied.
5.3.1 Defining the task, the sample group and collecting the data

A simple task involving the robot usage was proposed to a reduced group of people. The task was set in a big room, almost free of obstacles, in an office building. Users were given instructions about the usage of the robot, were informed about its limitations as well, and finally, they were told about the task to perform.

Within the task, the users where asked to start the robot, collect an object from a certain point in the room and return to the starting point with the object. They were informed that the robot should accompany them during the whole task. The starting point was always fixed in the same place in the room and the robot was initially placed in this point.

Four users conform the whole sample group. The sample group lacks of heterogeneity and number, but users availability and time itself were big constraints at the time of the experiments.

The data collected consisted on annotations on particular events happening, user’s opinions, plus a complete record of the internal variables of the robot, that includes all the data collected from laser and motors. According to the reduced sample group, the experiments were carried out in only one day.

5.3.2 Analysing the data

The data collected has to be analysed in order to draw useful conclusions based on that. The following factors have been considered determinant on the effectiveness of the tool:

- Time to complete the task. An effectively following behaviored robot should not delay the completion of the task.

- Number of times the robot misses the person. Each time the robot cannot follow the person anymore, the service itself is deteriorated

- Number of times the person has to stop to wait for the robot, or go back to the robot in order to be detected after a tracking loss. Each time the robot does not go fast enough, or requires for some reason that the person stops, the quality of the service is degraded.

- Average error relative to the desired position determined in the follower with \((\alpha, \rho)\), expressed in function of:

  - the error in the distance to the person being followed
  - the error in the relative orientation robot-person

The bigger this error, the further away the robot is from the original design idea, and the worse it will perform its behaviour, in detriment to the general service provided.
• Percentage of the total time of each experiment in which the robot is in each state, specially in follow state and in approach state. Despite of the follower problems commented in previous sections (see page 77), when the robot is in follow state, it manages to keep up with a person velocity better than in the approach state.

• Observations on the users and opinions retrieved from users. The anotations made on the behaviour of the subjects, and the observations verbalised by the subjects are indeed descriptive of the utility of the tool.

In the following, a couple of the trajectories traced by the robot while following the person are plotted (see figure 5.24 and 5.25), so as to present an approximated idea of the kind of movements in which the robot was involved during the experiments.

![Figure 5.24. One of the trajectory traced by the robot during the user oriented performance test](image)

The next four sections offer some comparative results concerning the factors formerly mentioned in relation to the four tests carried out.

1. Error of the relative positioning system. The graph 5.26 shows the error mean and standard deviation. Two errors are considered: the difference between the desired distance to the robot and the real one, and the difference between the desired human-robot relative orientation, and the real orientation.

According to the figures, even if the error mean in some experiment is quite low (e.g. −0.14), the high values of the standard deviation make the values close to this mean not so probable. The figures will be thus more useful to compare the performance of the different experiments. As explained in previous chapters, the system has difficulties on acquiring a certain orientation
Figure 5.25. One of the trajectory traced by the robot during the user oriented performance test.

Figure 5.26. Error means and standard deviations in the different experiments.

relative to the person. That would make the system have a higher error mean in the orientation that in the distance error. In these experiments, $\alpha$ has been set to $-\pi/2$, precisely behind the person, the easiest to reach orientation for the robot. That is the reason why the error mean of the distance and the orientation are of the same order.

2. Division of the total time among the different states. In figure 5.27, the state-time distribution in the four tests is presented.

The preferred state in the four tests is the approach state, which requires no special movement characteristics on the person being followed, unlike the follow state, which is often the second or third most preferred state. The still state is not usually prolonged in time, as any movement in the visual field of
the robot is meant to change from the still state to the pre approach state, a temporary, “unstable” state.

Going back to the figure 5.26, the tests with a higher rate on the follow state have slightly lower error means for the position and the orientation. That means that, despite of the fact that the follower performance is not as good as expected, it improves the average behaviour when compared with the approach state. Though in the previous section it was stated that the follower is not effective enough, in average it shows to perform better than the state approach.

3. The application time and the task time. The application time is considered to be the time that the application has been running. The task time is considered the time since the robot starts following the person towards his target, until the robot is ”brought” back to the initial point. Figure 5.28 shows this time division for the four tests.

The interest of this measure is set on knowing how long it really took robot and person to perform the task, not taking into consideration dead times before the activation of the robot, or after the task is completed. A faster completion of the task would often imply a more effective following, as it will usually involve few tracking losses or no necessity for the person to wait for the robot, for instance. In this case, while the task time is quite similar for all the cases, the application time is, evidently, bigger in those tests that showed to spend more time than the rest in the start state.

4. Number of tracking losses or user special attentions. The figure 5.29 tries to give an idea of how demanding was for the user to perform the task being followed by the robot. Normally, a tracking loss is associated to the person stopping and moving slightly, waiting to be noticed by the robot again. In

Figure 5.27. The division of the complete application time into the six states
the worst of the cases, the person has to go back on his steps in order to be noticeable for the robot again (what it is called here a user special action). Both cases were separated only for informative matters, as they imply different actions from the user.

According to figure 5.29, it is sensible that those tests with higher score are the ones which do not spend so much time in the follow state. To enter into the follow state, a certain continuity in the velocity of a person is required. Therefore, a break in this continuity such as a tracking loss would prevent the robot to switch to follow.

5. Observations made on the subjects, and observations made by the users. The following observations where made on the subjects behaviours:
users where often forced to wait for the robot, or reduce their walking speed to adapt it to the robot. Too much attention was required from the users

- when robot and user where walking at the same speed, the distance robot-user was often too big (3m), and in some cases detection loss would happen due to this distance increasing

- the usage of the robot was perfectly understood

- when the robot’s tracking system confused a piece of furniture with the person, the user was capable to reset the robot and make it track him/her again

- users never showed to be intimidated by the robot following them

The main observations made directly by the users during the experiment can be summarised in the following:

- users pointed out that when they were being followed at a comfortable speed, the robot was too far away from them

- users pointed out that the robot was often too slow to keep up with their speed

- users pointed out that the usage of the tool was easy in general

- users pointed out that some feedback from the robot would be useful in order to know if they have been detected by the robot

This observations show that the robot as a tool was easy to use, but did not perform as good as the users expected. Users expected the robot to be faster and stay closer to them while being followed.

The five points presented above conform an attempt to analyse the performance of the application as a whole while interacting with common users. In general, when dealing with real users, the system shows to be too slow for the users, and to follow them at a too big distance. The reasons for this might be found in the not fixed error (in probabilistic terms) for the distance and orientation error presented in figure 5.26. When it is reached in the robot, the state follow, as shown in figure 5.27, makes this error decrease. However, according to the problems pointed in section 5.2.2(page 77), the follow state is not fully efficient. Thus, the desired relative positional relation robot-human is never reached.

This chapter has presented a performance study of the system based in three aspects: the tracker, the follower, and the end-user utilization. The tracker showed to have an acceptable performance. In contrast, the follower did not perform as good as expected, as the positional control in the follow state was shown to be deficient according to section 5.2.2. This deficient performance in the follower made the user oriented performance be not fully satisfactory. Solutions for this problem are presented in 6.2.
Chapter 6

Conclusions and future improvements

The last two chapters depicted the implementation phase and the experimental phase of this thesis. This chapter presents the conclusions extracted from that work in relation with the problem definition, described in the introduction chapter. In addition, future improvements on this work will be presented.

6.1 Summary and conclusion

The starting point of this thesis was the goal of ”creating a person follower behaviour for a service robot able to cope with basic social constraints in an indoors environment”. According to this main goal, the architecture chapter defined five goals (page 15) to be completed.

The architecture of the system was first designed out of the requirements of these goals. The system was then implemented into a software application which was tested first in a simulator, and later moved to a real robot. Changes in the design were necessary in order to engage the complexity of the real scenario.

Finally, the system was tested with real users in order to check the general performance, and conclusions were drawn out of the results.

The five goals are revisited in the next lines, in order to summarise the achieved set of conclusions:

1. Detection and tracking of position and velocity of a person. It is necessary to implement two different behaviours in the tracking system that can deal with two different situations: the robot is not moving and a target has not been detected yet, and the robot has detected the target. In addition, the usage of a Kalman filter for tracking needs the process of tuning its parameters to provide proper output.

2. Motion coordination between the robot and the human. The approach chosen in this thesis consists on setting a relative position to the person, and command the robot to reach this position, and stay there with the same velocity of the person. A reliable estimation of the heading of the person to follow is precise
in order to define this position. In addition, the absolute velocity of the person
must be deduced from the velocity estimated by the tracking system, which is
relative to the robot. If the subject being followed describes a trajectory with
high or very changing curvature, reaching a relative position to the person has
been proved to be disturbing for the person, and hard for the robot. Thus,
an approach state is created in order to handle these situations. The choice
of the control system used to generate the commands for the motors is crucial
for the performance of the system. The motion coordination system should
consider visibility matters, as reaching the desired position and keeping the
person in the visual field might conflict sometimes.

3. Accompanying behaviour dependent on the environment. The parameters
that define the relative position of the person mentioned in 2 can be made
dependent on an external module.

4. Obstacle avoidance implementation. A static obstacle avoidance system based
in threshold zones around the robot is enough for the scope of this thesis.

5. Basic human robot interface. The interface explicitly offered by the robot does
not have to be very complex. Simple bumper pressing for starting and shutting
down the application are fair enough. However, more complex interfaces could
ease the interaction.

In addition, concerning all the aspects above in some way, the implementation
of a state machine that regulates the behaviour of the system proved to be effective.
The different behaviours are supported by another module (main module), while
the state machine determines the sequence of behaviours that fits better with the
context.

6.1.1 Conclusion

All in all, the goal set in the beginning of the thesis work has not been completely
accomplished. The algorithm implemented in the robot makes it follow people,
but can not deal properly with context variations, and finds difficulties in making
the robot acquire a relative position to the person followed. Initial work with the
Stage simulator proved the algorithm to successfully implement all these features,
however, reality proved to stay far from simulated worlds. Definitely, some of the
improvements explained in this chapter could bridge this gap between simulation
and reality.

6.2 Future improvements

A number of modules present in the architecture of the system can be improved in
some way. In the following, these modules will be presented and their improvements
suggested.
The tracker module. The actual tracker module starts looking for movement information subtracting subsequent laser scan data sets and looking for at least two bulbs on that difference, one positive and one negative. That makes the module sensitive to transversal movements to the heading direction, but not so much to moving objects moving in a radial velocity respect to the robot. Looking only for one bulb in the difference should improve the sensitivity to this other kind of movement.

The last version of the choose lines module detects only one leg and determines that the person’s position is the detected leg’s position. A step forward in the detection would be looking always for two legs in the laser data set, and determine the person’s position as a function of both legs.

In addition, the detect lines algorithm could be slightly modified in order not to detect only straight lines, but to detect straight lines fairly separated from the background.

The follower module. As explained in the experimentation chapter, in page 77, the desired velocity definition lacks of sensitivity on the desired position when the robot is in the follow state. This could be solved by retuning the values for the scale_factor, mentioned in the implementation chapter, until the changes in the desired position are significant enough to make the robot react faster to context changes.

A control technique for non linear systems called feedback linearization could be used to determine the speed and turn rate commands for the motors. The actual control system considers speed and turn rate control separately, despising non linear dynamics. A more complete and complex model could improve the motion control of the robot.

The robot interface module. The laser range finder is offering a new data set to the system with a frequency of 6Hz approximately according to the actual configuration of the robot and the player interface. The robot configuration could be changed in order to be able to get data at a frequency higher than 6Hz, which is slow in situations such as obstacle avoidance.
References


[12] C. H. Ku and W. H. Tsai. Robust trajectory planning in person following for vision-based autonomous land vehicle guidance by visual field model and visual


[22] Vicente Matellan Olivera and Reid Simmons. Implementing human-acceptable navigational behavior and a fuzzy controller for an autonomous robot.


[26] Erwin Prassler, Dirk Bank, and Boris Kluge. Key technologies in robot assistants: Motion coordination between a human and a mobile robot.


