Robots in Everyday Human Environments

On Platform Development and Behaviour Dependent Control

Master Thesis

Simon Kracht

Carsten Nielsen

Aalborg University · AAU

Department of Electronic Systems

Automation & Control

Automation & Control
Fredrik Bajers Vej 7C
DK-9220 Aalborg Ø
Phone.: +45 9635 8600
Web: http://www.control.aau.dk

TITLE:

Robots in Everyday Human Environments
- On Platform Development and Behaviour Dependent Control

SPECIALIZATION:

Intelligent Autonomous Systems

PROJECT PERIOD:

9th and 10th semester

2 September 2006 - 7 June 2007

PROJECT GROUP: 1030

GROUP MEMBERS:

Simon Kracht

Carsten Nielsen

SUPERVISORS:

Thomas Bak

Hans Jørgen Andersen

NUMBER PRINTED: 7

NUMBER OF PAGES: 130

ANNEXES: 9 on DVD

FINISHED: 7 June 2007

ABSTRACT:

This thesis concerns the development of an autonomous mobile robot intended for use in a human environment. From a primary focus on HRI, a novel robotic behaviour algorithm has been developed and validated through simulation as well as real world experiments. Furthermore, the behaviour of the robot is continuously adjusted to that of encountered people, by incorporating CBR based artificial intelligence, implemented by use of a *MySQL* database.
Besides interacting with people, the robot is capable of navigating and localizing itself in a given environment while avoiding unknown obstacles.
All robot software, including the implementation of the above algorithm, has been developed for use with the *Player* robot software framework, and implemented on a FESTO Robotino®. Consequently, due to the *Player* framework offering a wide range of robot features, the developed system can form the basis of future robotic research.
Experiments performed on the robot show promising results, in relation to both the behavioural capabilities and the developed system in general.

Preface

Targeting the issue of introducing robots in everyday human environments, this thesis documents the work of a master project on the specialization of Intelligent Autonomous Systems at the Section of Automation and Control, Aalborg University.

At present, the work conducted throughout this project is the primary part of the research conducted in cognitive robotics at Aalborg University. Furthermore, the part concerning behavioural human-aware robot control is at the forefront of the research in Human Robot Interaction in general.

Throughout the project period, the authors have made extensive use of the open source *Player* software robot framework, on which the developed solution is based. Furthermore, the free Unified Modelling Language (UML) toolbox BOUML developed by Bruno Pagès has been a valuable tool in the entire design process, while MATLAB© has provided much value in developing algorithms and illustrating the results of conducted experiments.

The thesis is intended for supervisors, examiner, students and others that might have interest in behavioural control of mobile robots.

Aalborg University, 2007

Simon Kracht

Carsten Nielsen

Reading instructions

References to literature are done by the Harvard method, where possible specific pages are added, e.g. figures, equations, and tables are numbered consecutively within each chapter. References to equations are in addition made in parenthesis e.g. (3.1). Acronyms are written in full length at first use and listed on page 112.

Matrices are written with bold upper-case letters e.g. $\boldsymbol{A}$, vectors with bold lower-case letters e.g. $\mathbf{a}$, while references to variables in source code are written as e.g. `foo`.

Furthermore, references will appear to a project Wiki:

`http://www.control.aau.dk/~tb/wiki`

References to annexes on the enclosed DVD, is done by e.g. Annex 1.
The DVD contains source code, videos, data sheets, etc.

Contents

CHAPTER 1

Introduction

Since the first industrial robot began its work at the Ford factories in 1959, research in robot technology has been performed intensively. The original focus of robotics research was certain human jobs involving repetitive tasks, identified as being well suited for industrial robots. Later on, the use of robots evolved as they became increasingly capable of performing more varied tasks, e.g. by the use of tele-operation allowing for human control of the robot.

Through the years, the focus of the robotics research has gradually broadened, and now covers not only industrial applications but also a wide range of other purposes. By moving from industrial repetitive tasks to more autonomous tasks, further interaction between robots and users has received increased attention. Thus, the Danish council for Technological Foresight under the Ministry of Science and Technology, has investigated the innovation possibilities related to robot technology in a report on Cognition and Robotics published in April, 2006 [The Danish Ministry of Science and Innovation, 2006]. Cognition stems from the Latin "cognoscere", which means to know, to recognize, to understand.

The report concludes, that the development of robots capable of carrying out more and more advanced cognitive demanding tasks, possesses *"a great potential for alleviating critical problems and promoting innovation in areas important to the society"*. These areas are; Industry, Agriculture, Experiences - play and learning, Service and care, and finally Hospitals and health. The common goal is an advanced robot, capable of participating in the everyday human life.

A good example of such development of robots with autonomous capabilities, is the present possibility for regular consumers to purchase a robot vacuum cleaner as e.g. an iRobot Roomba® at an affordable price.

Robots participating in the everyday life is a main part of the focus in the challenging research field of Human Robot Interaction (HRI), being at the intersection of other research fields ranging from psychology and social sciences to artificial intelligence, computer science, robotics etc. [Dautenhahn, 2007].

For a robot to navigate in a human environment, being highly dynamic and cluttered, requires a high degree of automaticity, as opposed to industrial robots for which the environment is static and known. Furthermore, the robot must be capable of both receiving signals by which humans interact and responding to these. Conducted research in the field of HRI has spawned a variety of robots targeted at different application domains ranging from museum and home tour guides [Kleinehagenbrock et al., 2004; Burgard et al., 1999] to various different service robots as e.g. the Care-O-Bot [Frauenhofer, 2004]. Probably the most renown result of these new kinds of human interactive robots is the Honda Asimo robot [HONDA, 2006].

At the Section of Automation and Control (SAC) at Aalborg University, research has been conducted in various parts of the robotics field such as development of autonomous control for helicopters, coordination of multiple autonomous robots and establishment of a humanoid robotic platform, intended for research in helping people with walking disabilities.

SAC has an interest in broadening the robotics research to embody elements of HRI and cognitive abilities. Postulating, that the primary criteria of success in obtaining rewarding interaction between human and robot, is the initial encounter between man and machine, it is of utmost importance to make this initial event as smooth and human-friendly as possible.

However, obtaining such a setting is still an open question in the research community. Hence, some researchers, that in order to establish the best possible foundation for a successful HRI solution, the given robot should be as human-like as possible, in terms of appearance as well as behaviour [Breazeal, 2002]. Contrary to this, other researchers argue that human-beings in general behave socially towards artifacts, and thus research should be concentrated on investigating those situations where this natural human ability is somehow disturbed [Dautenhahn, 2007].

This project is concentrated on the findings of [Butler and Agah, 2001], where the effect of different types of robot appearance and behaviour on human beings have been studied. Thus, the aim is to develop a robot with an appearance and behaviour proven to be valuable in the effort of establishing as pleasant a setting as possible. Thus, a new hardware platform (a FESTO Robotino®) has been acquired, featuring an omnidirectional drive, nine IR sensors, a bumper, a camera, and an additional range finder. Furthermore, is has been a desire to implement a robot software framework acclaimed by other researchers throughout the robot research community. Referring to framework the comparison presented in Appendix A, the open-source *Player project* [Collet et al., 2005] has been identified as the best choice. Based on a server/client principle, this robot interface allows for multiple instances of possibly different programming languages, to access robot hardware through predefined interfaces. Finally, many researchers have contributed to *Player* with various drivers covering everything from specific hardware, to outright navigation modules.

The long term objective of the research is a social assistive robot, capable of smooth interaction with people around it. Moreover, the robot should be able to extract features from these encounters and to learn from them, in the sense that the robot continuously becomes more aware of the people it approaches, and utilizes this knowledge to improve its integration in the environment. Applying such capabilities to the robot, would enable it to operate in a variety of applications.

1.1 Robotic context

With the above overall long term objective in mind, this thesis focuses on developing a robotic control system, which takes into account behavioural features of both the robot and encountered humans. By extending the Robotino® platform, it is made the foundation of a software framework offering improved HRI capabilities in terms of robot navigation.

According to [Isaacs and Walendowski, 2001], the process of designing such useful and usable technology, starts from a description of the fundamental relationship between user and technology. A relationship described with the analogy of a butler and his/her employer. Thus, a butler must always be prepared to assist his/her employer, and if in doubt of the task to perform or if problems should arise, the butler will ask as few and as relevant questions as possible. Furthermore, the butler does not ask how he can be at service, but instead figures out how his employer uses him, and thus develops a way to anticipate certain calls for service. In the same way, the employer gradually develops an idea of which tasks the butler performs well, and furthermore how he should be addressed in order for the employer to obtain what is wanted.

Developing technology able to enter into such a collaboration with its user, must according to [Isaacs and Walendowski, 2001] be governed by the rules of "negative and positive politeness". In short, new technology should at a minimum obey the rule of negative politeness, which basically means that the technology must not be rude to the user, by offering too many options, features, questions etc. For a technology to obey the rule of positive politeness, it must be able to collaborate with its user, in offering only relevant questions in an easy understandable way. Furthermore, abilities such as problem solving, error prevention and task prediction characterizes positive polite technologies.

With the above butler analogy in mind, a case has been set up as the foundation for this project and illustrated in Figure 1.1.

Imagine a person entering the foyer in search for room number 5. The person does

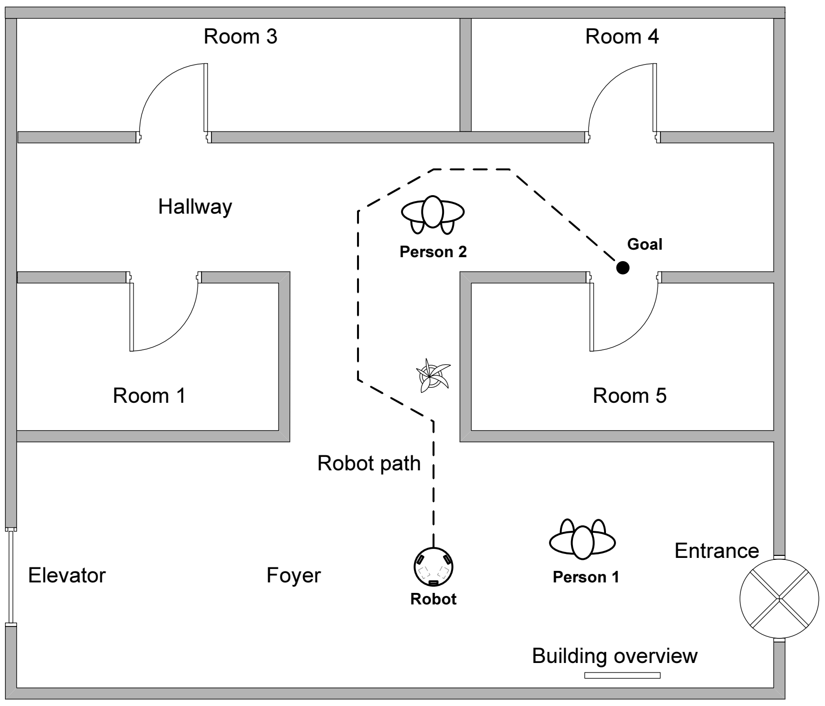


FIGURE 1.1: *Fictitious scenario for identifying sub-objectives.*

not know where to find the room, and might then act in several ways by e.g. going to the building overview or wander randomly around the foyer looking for the room. In the first case, the robot should not approach the person since the person will most likely find the way to the room by looking at the building overview. In the latter case however, the robot should approach the person and ask if any help is needed. If so, the person tells the robot where he/she wants to go. The robot then guides the person to the desired location and returns to the foyer. Upon every encounter with a person, the robot must evaluate the outcome of the interaction and adapt any similar future interaction based on the such previous experiences. By example, if the robot has approached persons in the vicinity of the elevator a number of times and each time the person has refused help from the robot, the robot must learn to approach people at the elevator less frequently.

1.2 Project objectives

From the above robotic context description based on the long term research objective, the following objective for this particular project can be formulated.

> A FESTO Robotino® equipped with a range finder, capable of conducting detection of and behaviour based interaction with people, while navigating in a

dynamic human environment. Furthermore, the robot is capable of learning from the various encounters, in order to constantly improve its knowledge of the behavioural characteristics of the people in its surroundings. All communication between software and robot hardware, is conducted through a *Player* server.

Having outlined the overall project objective, the following section starts from a definition of the test environment, and furthermore outlines the functionalities needed for the robot to fulfil the objective stated above.

1.2.1 Experimental set-up

The environment in which the robot is to be functioning is illustrated in Figure 1.2.

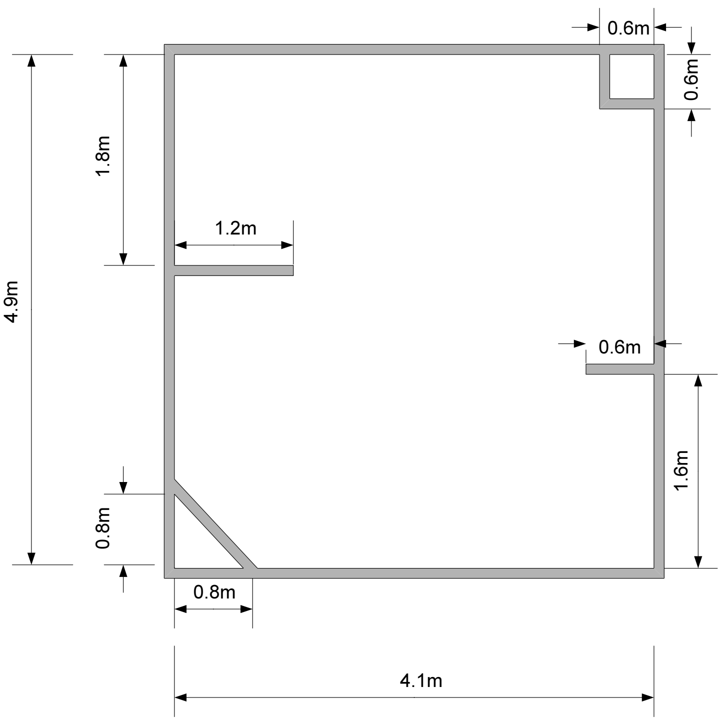


FIGURE 1.2: *Illustration of the test environment, in which the robot is to be functioning. The colour of the walls of the environment should be easily distinguished from that of a person entering.*

As denoted on the figure, the environment consists of a single room measuring 4.9x4.1 meters. The following restrictions governs the environment:

- No obstacles are present in the room.

- Only one person can enter at a time.

Obviously, the described environment is only a limited reflection of a real world crowded lobby, with people passing through on a daily basis. On the contrary, the test environment described is set up as the simplest environmental realization of such a lobby, still allowing for conducting satisfactorily experiments on the final robot solution.

In order for the robot to interact with a person in the room, the following functionalities should be developed. Notice, that the functionalities listed are intended as extensions on top of basic features provided by the Robotino® as e.g. robot mobility, network connection etc.

Navigator The robot must be able to navigate from one point to another, whether guiding a person to a zone or when moving from one point to another in the environment. Exactly as a GPS car navigation system tells the driver where to go, the Navigator tells the robot where to move. This implies that the Navigator must somehow know the environment in which the robot is moving, obtained by providing it a map of the above described test-environment.

Pilot When instructed to go to a certain destination by the Navigator, the robot Pilot is responsible for the essential robot movement. Thus, the primary objective for the Pilot is to avoid previously unknown obstacles encountered by the robot.

Localizer Due to the robot's intended function as a guidance robot it must constantly be aware of its own location in the environment, in order to guide people to their desired locations.

Person Detector When moving around in a given environment, the robot must be able to detect persons in its surroundings in order to evaluate their behaviour, and possibly initiate interaction.

Person Evaluator As described above, the robot must evaluate a detected person to determine certain behavioural characteristics. Recording these characteristics is tightly connected to the functioning of the Trainee functionality described below.

Trainee Upon every human encounter, the robot must posses the ability store and associate the Person Evaluator characteristics with the fact of whether the person is interested in assistance from the robot or not. Consequently, the Trainee functionality allows for the robot to be gradually taught the behaviour of the people in the environment, and thus to get increasingly better at judging whether interaction should be attempted or not.

Communicator Determined that a person is interested in interaction, the robot must be able to initiate a conversation, i.e. to pose questions and receive replies.

Behaviour As indicated in the introduction, behavioural control of the robot is an extremely important part of HRI, being one of the primary causes for a successful human encounter. Thus, to increase chances of successful human encounters, it is necessary that the behaviour of the robot signals courtesy towards encountered people. Lack of such human-friendly behavioural control, might indeed impose unease or even intimidation on people interacting with the robot. Furthermore, safety conditions must be taken into consideration, such that the robot does not get too close to people, or drive too fast when at close range.

Since this project focuses on the behavioural dependent control of the robot, some functionalities are identified as being of higher priority in in relation to the main objective. These are the Person Evaluator, Trainee and Behaviour. Thus, the crucial navigational functionalities of the Localizer, Navigator and Pilot will be based on existing solutions.

Thus, from the above introduction and description of the test environment, the following outlines the contributions to the HRI research, obtained through the work described in this thesis.

1.3 Contributions

The primary contribution from this project, is an algorithm providing robot control abilities, allowing for human encountering while obeying a set of rules obtained from human-robot studies [Butler and Agah, 2001] [Koay et al., 2006]. Furthermore, additional robot features have been developed, enabling the robot to judge and remember the interest of encountered people, based on a variety of situational specific parameters. Such knowledge of past experiences, allows for a gradually improving robot behaviour in terms of minimum unnecessary disturbance of detected people. The algorithm has been implemented as a *Player* driver, allowing for swift integration with any given *Player* compatible robot supporting the interfaces used.

The work conducted in order to reach the primary contribution described above, has spawned a variety of sub-contributions.

Firstly, through the use of the *Player* framework, Robotino® and range finder *Player* drivers have been developed in order to make the robotic platform capable of running a fully functioning *Player* server. Both drivers have been submitted to the *Player* community, for inclusion in the framework source.

Furthermore, a Wiki has been developed concurrently with the project solution, providing more detailed information on practical issues concerning the Robotino® platform, e.g. preparation and installation of *Player*.

To enable the robot with HRI capabilities, a robotic behaviour algorithm has been developed, extending present research in HRI [Sisbot et al., 2006]. This algorithm has been implemented as a *Player* driver allowing for other researchers to apply the algorithm on their respective robotic platforms, provided that they are compatible with the *Player* framework.

Lastly, Case-Based Reasoning (CBR) has been introduced in the HRI design, allowing the robot to act on the basis of past encounters, thus strengthening the possibility of a smooth introduction of a robot in a human environment.

The above mentioned contributions can be summarized as below, including other minor by-products of the development:

Robot interface

- Fully functioning Robotino® and range finder *Player* drivers.
- Fully functioning *Player* server installed on the Robotino®.

Robot control

- Robotic behaviour algorithm implemented as a *Player* driver.
- Navigator, Pilot, and Localizer for Robotino®. These will all be implemented on the basis of existing solutions available in *Player*

Robot interaction

- Person Detector.
- Person Evaluator.
- Trainee based on the principle of CBR.
- Communicator.

1.4 Outline

Below, the remaining chapters of the thesis are listed and briefly summarized.

Chapter 2 Analysis

Starting from a description of the FESTO Robotino® hardware platform along with the external range finder used, the Analysis also treats the *Player/Stage* framework to be applied on the robot. The remainder of the chapter covers the different functionalities judged to be necessary for the robot to fulfil the overall objective. Finally, an overview of the proposed system structure is presented.

Chapter 3 Design and Implementation

The Design and Implementation chapter presents a more thorough description of important issues related to the process of implementing the conceptual system solution of the Analysis.

Chapter 4 Experiments and Results

The Experiments and Results chapter embodies all experiments conducted in order to verify that the final system solution fulfils the stated demands. Thus, the functioning of all primary robot functionalities will, directly or indirectly, be documented in this chapter.

Chapter 5 Closure

Puts into perspective the findings of the Experiments and Results chapter, and concludes on the overall project solution.

CHAPTER 2

Analysis

Based on the above introduction, a system structure providing the Robotino® with functionalities, spanning from basic mobility to more advanced behavioural control is required.

Furthermore, the Robotino® platform is described, highlighting the need for both additions and modifications to be made. These alterations concern both hardware and software.

The analysis of the software to be developed is done by using UML use-cases which are presented in introductory of each section concerning software related analysis.

2.1 Robot platform

This section describes the Robotino® platform in terms of technical specifications, and establishes an overview of the features provided. Furthermore, additional sensors needed to make the Robotino® capable of the functionalities described in Section 1.2 are treated.

2.1.1 FESTO Robotino®

In Table 2.1 the technical specifications of the Robotino® are listed, and in Figure 2.1 the robot is depicted from below along with indications on sensor and motor placements.

The three omnidirectional drive units of the Robotino®, defines the robot as being holonomic, meaning that the controllable degrees of freedom equals the total degrees of freedom of the robot. Powered by Dunker DC motors equipped with optical shaft encoders the Robotino® can reach speeds of up to $10\,\frac{\mathrm{km}}{\mathrm{h}}$. Furthermore, interchangeable pinions allow for custom gearing from 1:4 to 1:16.

For the Robotino® to provide the functionalities described in Section 1.2.1, the sensors currently found on the Robotino® are not sufficient. Primarily, this discrepancy is found in the lacking possibility of the Robotino® to measure distances beyond $30\,\mathrm{cm}$ which is not sufficient for self-localization as described later in Section 2.3.1. This results in the need for applying an additional range sensor.

Robot:
Diameter of 370 mm
Height including housing without webcam of 210 mm
Three omnidirectional drive units each featuring a 3600 rpm Dunker motor
Overall weight of about 11 kg
Maximal payload of about 5 kg
Sensors:
Nine SHARP GP2D120 distance sensors
Analogue inductive sensor
Bumper with integrated sensor
Two optical sensors
Creative Live! colour webcam with USB interface
Three optical shaft encoders
Embedded controller:
PC104 MOPSlcdVE processor of 300 MHz
SDRAM 64 MB
Compact flash card (128 MB) with C++ libraries for accessing the Robotino®
Wireless LAN interface board
Interfaces: Ethernet, 2 x USB, 2 x RS-232, Keyboard and mouse, parallel port
I/O interface card:
outputs for controlling the three omnidirectional drive units
10 analogous inputs ($0 - 10$ V, 50 Hz)
Two analogous outputs
16 digital inputs (can be switched to outputs)
Three relays

TABLE 2.1: *Technical specification of the FESTO Robotino® hardware platform.*

Robotino® software

Bundled with the Robotino® platform, FESTO provides an Application Programming Interface (API) called *RobotinoCom* offering the functions listed in Appendix B.

The operating system of the Robotino® is divided into two layers, being a regular Linux layer and a Real Time Linux (*RTLinux*) layer. All hardware access goes through the *RTLinux* layer, while the Linux layer provides standard user space. Thus, *RobotinoCom*

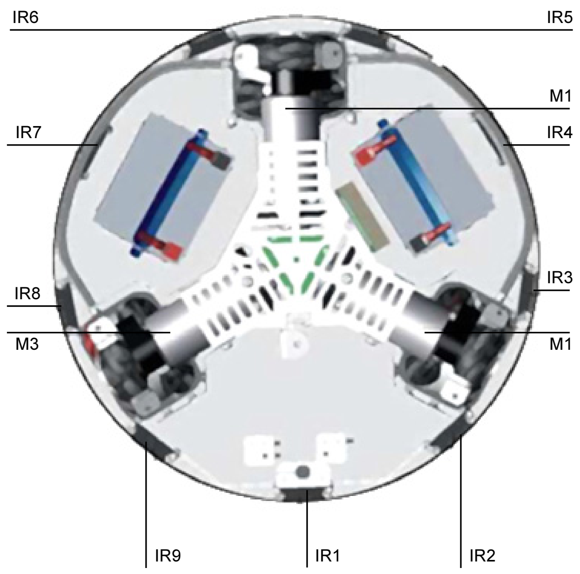


FIGURE 2.1: *Bottom view of the Robotino® where IR-sensors are indicated by "IR#" and motors by "M#".*

only supports interfacing of a limited number of additional FESTO sensors, since these are provided for in the *RTLinux* implementation. Therefore, in order to make the of an additional range sensor, measures must be taken to access the sensor through the *RTLinux* layer.

Such *RTLinux* integration is documented in Section 3.2.1, whereas the following section describes the additional range sensor used.

2.1.2 Range sensors

As described in Section 2.1.1, an additional range sensor is needed for the Robotino® to function as intended. At SAC, two range sensors have been available for use in this project. These are a PBS-03JN and a URG-04LX, both manufactured by Hokuyo. Due to the future perspective of the use of the Robotino® at SAC, *Player* drivers have been developed, making the use of both range sensors possible. Thus, the URG-04LX driver, already provided through *Player*, has been modified to be compatible with the Robotino® *RTLinux* solution, while the PBS-03JN driver, on the basis of prior development [Mathiasen et al., 2006], has been developed to support both regular and *RTLinux* enabled use. The general PBS-03JN driver has been submitted to the *Player* community.

However, for the use of this particular project, the URG-04LX has been chosen due to its superior specifications compared to the PBS-03JN (data sheets for both sensors are found in Annex 9A and 9B, respectively. Hence, only the URG-04LX will be treated further, whereas details on the *Player* driver developed for the PBS-03JN sensor are found at the project Wiki.

Hokuyo URG-04LX

The Hokuyo URG-04LX is a scanning laser range finder developed for robotics applications. Selected performance specifications of the sensor is found in Table 2.2.

Property	**Value**
Distance range	0.2 - 4 m
Distance resolution	1 mm
Distance accuracy	± 0.01 m within 1 m
	$\pm 1\%$ otherwise
Angular range	240 °
Angular resolution	0.36 °
Power source	5 VDC
Interface	RS-232

TABLE 2.2: *Specifications of the Hokuyo URG-04LX range finder.*

The development of the *Player* driver for the URG-04LX is described in Section 3.2.3.

2.2 Software framework

As described in the introduction this project utilizes *Player* in interfacing the Robotino® robot. This entails, that the design of system solution is governed by the concepts utilized by the *Player* robot interface. To provide a comprehension of this interface, this section explains the basic concepts of the *Player* interface along with the *Stage* simulation software.

2.2.1 *Player* robot server

Player can essentially be described from three key concepts being [PlayerProject, 2006, Tutorials part]:

Interface: A specification of how to interact with a certain class of robotic sensors, actuators, or algorithms. The interface defines the syntax and semantics of all messages that can be exchanged with entities in the same class.

Driver: A piece of software (usually written in C++) that talks to a robotic sensor, actuator, or algorithm, and translates its inputs and outputs to conform to one or more interfaces. The driver's job is to hide the specifics of any given entity by making it appear to be the same as any other entity in its class.

Device: A driver bound to an interface, and given a fully-qualified address. All messaging in *Player* occurs among devices, via interfaces.

Player is implemented as one or multiple servers providing client access to the robot hardware through e.g. a TCP/IP connection. See Figure 2.2 for an example of a system set-up.

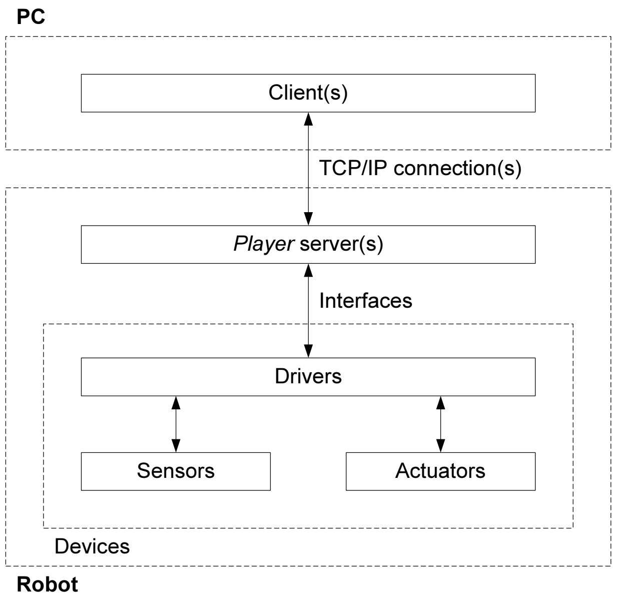


FIGURE 2.2: *The architecture of the Player software.*

Implementing the robot interface in this way has a number of advantages:

- The client software can be written in any language that supports the use of TCP sockets and having any structure the designer desires.
- Any number of clients can access the interfaces provided by the robot. Thus, one client can take care of navigation while another client processes e.g. data from a camera.
- Software developed for one particular robot can be reused for any robot providing similar interfaces.

- Simulating the robot control software can be done using the *Stage* simulator as described later in Section 2.2.2.

A robot device is implemented in *Player* by writing a configuration file containing linked drivers and interfaces complying with the hardware configuration of the given robot. Since no Robotino® driver is included in *Player*, new drivers has been be developed as described in Section 3.2.2. However, no new interfaces are needed, since the interfaces bundled with *Player* already support robot control (`position2d`), reading of IR sensors (`ir`), camera (`camera`) and bumper (`bumper`). Furthermore, the use of a range finder is supported through an interface for laser range finders (`laser`).

Due to the *RobotinoCom* API enabling control of motors and reading of sensors, the task of writing a Robotino® driver, is essentially a task of interlacing the *Player* driver structure with the desired API functions.

2.2.2 *Stage* simulator

Stage is the simulator part of *Player/Stage* providing the possibility of simulating robots, sensors, and objects in a two dimensional virtual environment, and furthermore to experiment with regular *Player* drivers. This is particularly useful when testing e.g. robot control algorithms, since the most widely used robot hardware and sensors are supported. A screenshot of the *Stage* window is seen in Figure 2.3.

When a specific piece of hardware is to be modelled, the parameters of a matching basic model is altered to represent the properties of the actual hardware (e.g. size and mobility characteristics). This specification is done through `.inc` files. Hereafter, control algorithms can be applied as if the hardware were indeed present. By using the *Player* plug-in `libstageplugin`, it is possible to access simulated devices as if they were normal *Player* devices.

The virtual environment (in *Stage* denoted "world") in which to perform the simulation is specified through a `.world` file. In this file, details of the environment are specified, such as which map is used to define the environment structure, and which models should be present at which initial positions.

When a *Player* device has been set up in *Stage*, the interfaces which the device provides can be accessed using the `playerv` or `playernav` utilities. `playerv` is a GUI client used to monitor the data flow from specific devices, e.g. to view the velocity of a robot or the distance measurements made by a range sensor. Furthermore, for some devices it is possible to provide commands as e.g. velocity input to a robot. The `playernav` utility is also a GUI client, however targeted on localisation and path planning features.

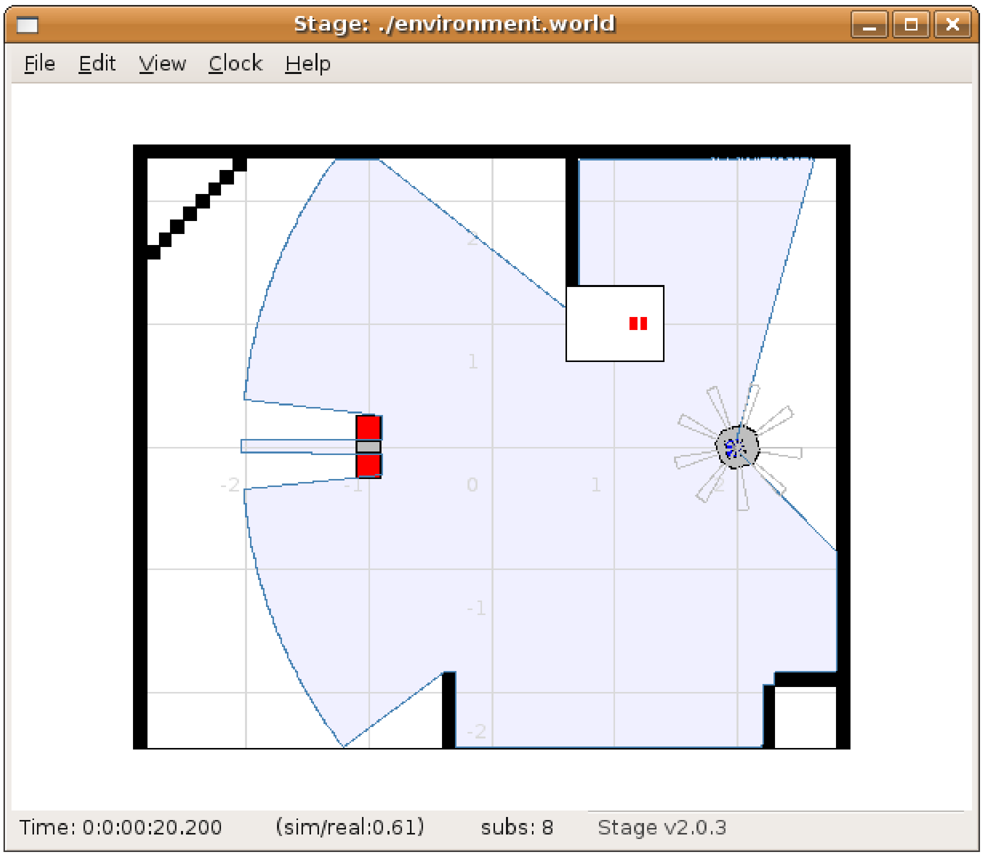

Stage: ./environment.world
File Edit View Clock Help
Time: 0:0:00:20.200 (sim/real:0.61) subs: 8 Stage v2.0.3

FIGURE 2.3: *Screenshot from the Stage simulator. The simulated robot (grey object) is equipped with a laser range finder for range measuring, blob finder for colour tracking, and IR sensors for close range detection. The red object represents a person.*

2.3 Required robot functionalities

This section treats the analysis of the robot functionalities as outlined in Section 1.2 to be necessary for fulfilling the project objective. The chapter starts from describing the robot controlling functionalities of the Localizer, Pilot and Navigator, while the remaining sections treat the interactive functionalities of the Person Detector, Person Evaluator, Trainee, Communicator and Behaviour. Finally, the development of a Controller interconnecting all the above functionalities is proposed.

2.3.1 Localizer

As described in Section 1.2 the robot must be able to localize itself in relation to a predefined map of the environment, as also indicated in the use-case diagram of Figure 2.4.

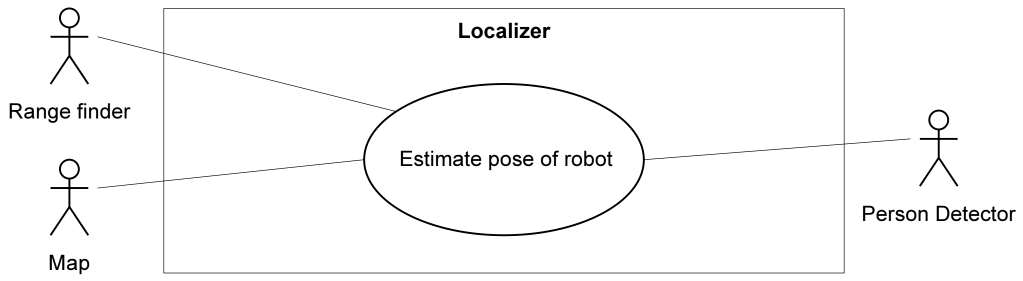


FIGURE 2.4: *Use-case diagram of the Localizer functionality, which estimates the pose of the robot from received range finder readings and a provided map of the given environment.*

This section introduces the necessary notions used in relation to localization. Furthermore, an existing *Player* driver intended for localization of mobile robots is described.

Since the Robotino® is to be used in a hybrid reactive/deliberative manner, it must be able to navigate in a known environment and at the same time avoid previously unknown obstacles.

A straightforward way, is to keep track of the robot's movements by simply using integrated velocity data (known as odometry) supplied by the Robotino® driver. However, such an approach is unreliable due to position errors continuously being accumulated. Hence, a method in which the odometry data is supplemented by other sensory information is needed.

The most extensive alternative would be to make the robot capable of drawing the map on-line as it moves around the environment while keeping track of its position. This method is known as Simultaneous Localization And Mapping (SLAM), and is described in e.g. [Garulli et al., 2005]. However, as mentioned in Section 1.2, the Robotino® will be provided with a map of the environment in which it is desired to operate, enabling the use of an existing *Player* driver intended for localization.

First some necessary definitions related to task of localization are introduced:

Location The robot's location in the two-dimensional world coordinate-system as described on the project Wiki.

Heading The heading is the robot's rotation relative to the world coordinate-system defined by θ (refer to the project Wiki for further details).

Pose The robot's pose contains both its location and heading.

Global localization Concerns the problem of turning all possible poses of the robot into one coherent and limited set of possible poses. If the set of possible poses is divided into multiple subsets (if e.g. it has been determined that the robot is located in a corner, but the specific corner has not been identified) the global localization problem has not been solved.

Local localization Concerns detailed tracking of the robot when global localization has been performed.

The `amcl` *Player* driver

Player offers a driver called `amcl` for localization. This driver is a so-called "abstract driver", meaning that it uses other drivers instead of hardware as sources of data. In this case, the `amcl` uses odometry data provided by the *Player* driver for the Robotino® through the `position2d` interface and range measurements provided by the range finder through the `laser` interface. The processing of the data in `amcl` is done using a particle filter, which is a variant of the Bayes filter, thus utilizing a probabilistic approach to position estimation. The particle filter is also known as a Monte Carlo filter. Furthermore, the amount of particles used by the filter is adapted dynamically to match the computationally capabilities of the system, resulting in a method known as Adaptive Monte Carlo Localization (AMCL).

Two different outputs are available from the `amcl` driver. One is a representative sample of the pose hypotheses weighted by likelihood, the other is the most-likely pose hypothesis. The latter is formatted according to the `position2d` interface, thus providing data which can be pretended to come from a perfect odometry system. In the following, a more detailed description is made of the AMCL method as well as the `amcl` driver.

The AMCL method

The AMCL method is by far the most widely used method for robot localization due to its simplicity of implementation and applicability [Thrun et al., 2005, p. 250]. Furthermore, compared to other Bayesian filters, which can also be used for localization purposes (e.g. the Extended Kalman Filter (EKF)), AMCL has a number of advantages. First of all, it takes raw sensor measurements with any noise distribution as input where, in comparison, the EKF assumes Gaussian noise distribution. Secondly, in contrast to the EKF, the AMCL can be used for global localization and is robust to e.g. robot kidnapping [Thrun

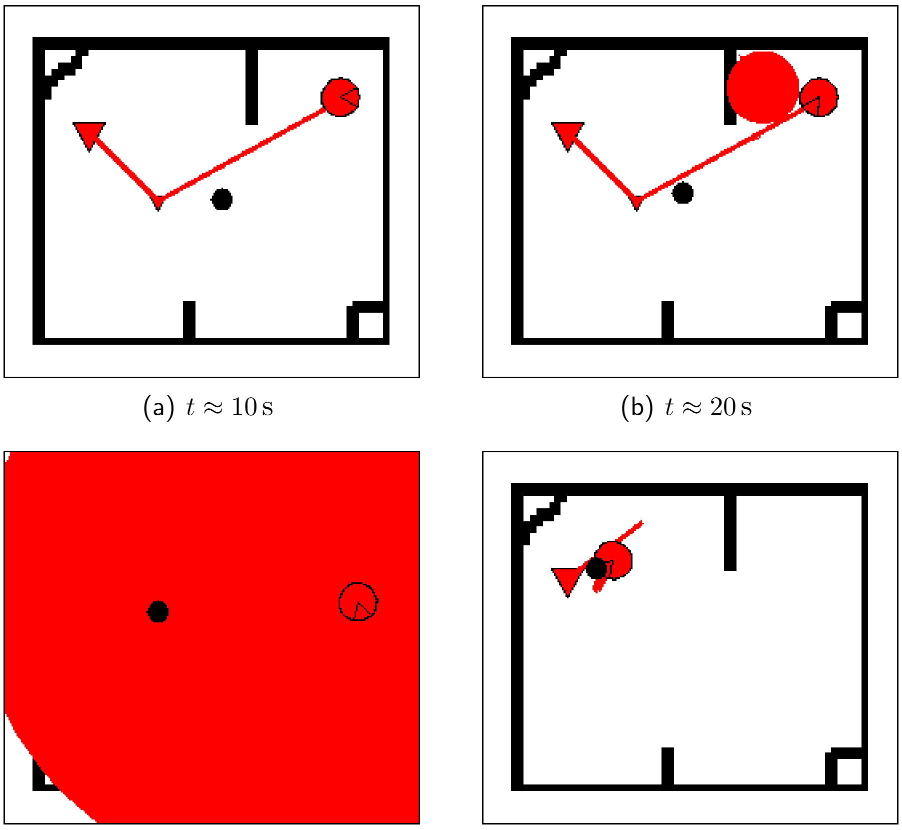

(a) $t \approx 10\,\mathrm{s}$
(b) $t \approx 20\,\mathrm{s}$
(c) $t \approx 40\,\mathrm{s}$
(d) $t \approx 60\,\mathrm{s}$

FIGURE 2.5: *Illustration of the* `amcl` *driver functioning, obtained by simulation in Stage and capturing of images by use of the* `playernav` *utility. The estimated position is marked by the red robot, particles by a filled red circle, and the robot by a filled black circle.*

et al., 2005, p. 274]. However, the AMCL has one disadvantage which is the amount of computations needed for the filter to work properly.

AMCL implementation

The overall implementation of the AMCL method in the `amcl` *Player* driver can be illustrated as in Figure 2.6 in which the function blocks and drivers are depicted along with their mutual information flow.

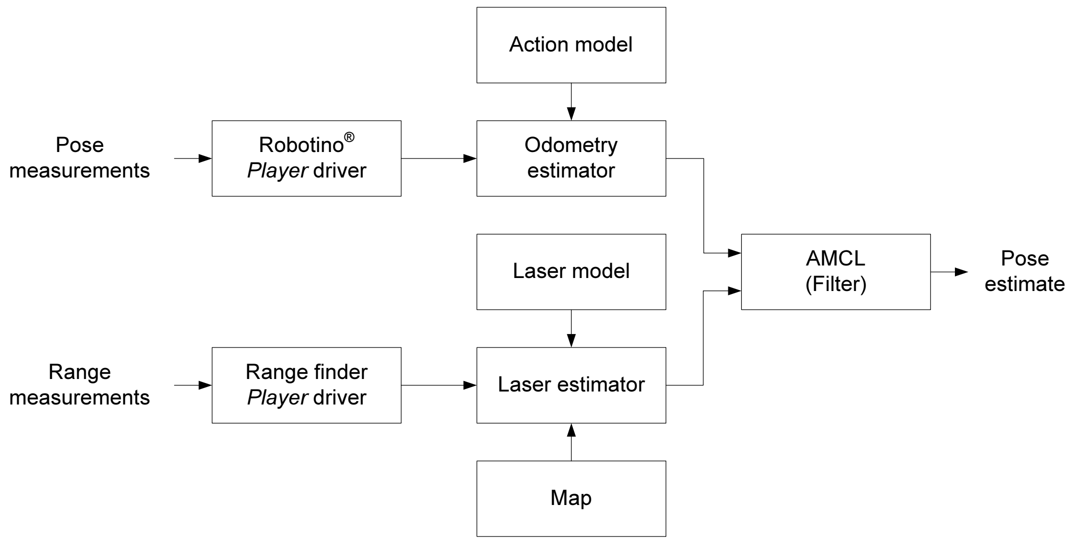


FIGURE 2.6: *The function of the* `amcl` *driver. Motion and laser measurements are used individually to provide temporary pose estimates, which are fused by the AMCL algorithm to provide a final pose estimate.*

Starting from the left of Figure 2.6, *Player* drivers for the Robotino® and range finder continuously collect data and publish these for subscribing client or drivers. The range finder driver measures distances to objects, whereas the Robotino® driver provides pose estimates through the `position2d` interface by use of the kinematics derived on the project Wiki.

When the Odometry estimator module receives odometry data from the Robotino® driver, the data is applied to the action model, time stamped and pushed onto a data queue to be processed by the AMCL module in which the actual filtering takes place. A similar approach is used by the Laser estimator module when receiving measured distances from the range finder driver. However, in this case, the laser model and map are used.

The particle filter is initialized by randomly distributing the initial number of samples (particles) over the entire map. At this point all particles are assigned similar likelihood weight since the robot could be anywhere on the map.

Next, the robot "senses" using a representative sample of the range readings . For each particle the range data is compared to the ranges which would theoretically be correct, if the robot's pose was equal to the pose of the particular particle. This comparison, results in a likelihood weight being assigned to each particle. The theoretically ranges are computed using the map of the environment and a probabilistic model of the range finder (denoted Laser model in Figure 2.6). If the robot has not moved, the sense step is performed again. By letting odometry data have priority over range data, it is ensured that the pose estimate will correspond to the latest received odometry data. However, this might result in range data being queued up, especially when the number of particles is large (e.g. in the initial phase of the filter's operation). The queued range data will be processed, when the computational demand decreases as a result of the increasing pose estimate likelihood and the resulting decrease in the number of particles.

If the robot has moved, the odometry data is used through a probabilistic motion model (denoted Action model in Figure 2.6) to update the filter. When updating the filter, a new set of particles is generated based on the likelihood weights of the prior set along with the motion model of the robot. The areas in which the likelihood weights were high are assigned a larger amount of particles than the areas, in which the likelihood weights were small.

The above described cycle is continuously repeated during AMCL filter operation.

Limitations of the `amcl` driver

The described `amcl` driver is designed for localization in static environments. However, in this project, the Robotino® will be functioning in a dynamic environment in the sense that a person will be walking about. This obviously affects the operation of the localizer due to the apparent inconsistency between actual range measurements and expected range measurements. One possibility of overcoming this problem is to simply regard the dynamics as noise. Alternatively, the state of the system may be augmented to embody the dynamics [Thrun et al., 2005, p. 269]. Yet another method, preferable due to its simplicity compared to the state augmentation, is outlier rejection, which basically preprocesses sensor measurements in order to eliminate measurements which are affected by dynamics.

The documentation of the `amcl` driver states that the driver is still evolving. Specifically, the likelihood models of the distance measuring device and odometry are characterized as simple and candidates for further work. Hence, it is necessary to test the function of the driver in the present hardware configuration to evaluate its performance. Experiments and results on the `amcl` driver are described in Section 4.2.

2.3.2 Pilot

From the use-case diagram of Figure 2.7. it is seen that Pilot's only task is to ensure that the robot avoids any unknown obstacles while moving.

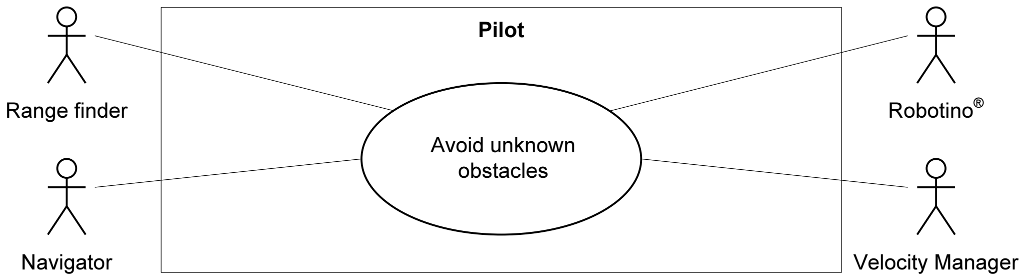


FIGURE 2.7: *Use-case diagram of the Pilot functionality. The Pilot prevents the robot from colliding with unknown obstacles, by using range finder readings. Possibly altered velocity commands are passed on to the Robotino® and the Velocity Manager.*

To implement this feature, an existing *Player* driver (`vfh`) implementing the Vector Field Histogram+ (VFH+) algorithm [Borenstein and Koren, 1991; Ulrich and Borenstein, 1998] is used.

The Vector Field Histogram+ algorithm

The different stages of the VFH+ algorithm are depicted in Figure 2.8.

When the VFH+ algorithm is invoked, a grid $\mathbf{C}$ containing the immediate surroundings of the robot is constructed and denoted as the active area. In this case the active area is the area between -90° and 90° with respect to the forward direction of the robot, due to this being the area in which the range finder operates (Figure 2.8(b)). When the range finder detects an object in a specific cell, the certainty value c_{ij} of the cell is incremented, and thus becomes a measure of the certainty of the cell being occupied by an obstacle. Based on the distance between the cell and the robot centre (length of the obstacle vector) along with the certainty factor c_{ij}, an obstacle vector is created for each cell. The active area is divided into a number of angular sectors of equal angular size.

Based on the obstacle vectors a Polar Obstacle Density (POD) is calculated for each sector, thus becoming a measure of density of obstacles within each sector. Furthermore, the obstacle cells in the map are enlarged by the radius of the robot. Now, $\mathbf{C}$ can be transformed into a primary polar histogram $\mathbf{H}$ as depicted in Figure 2.8(c) containing the POD as function of the sectors taking into account the size of the robot. Sectors with high PODs are called peaks whereas sectors with low PODs are called valleys. The primary

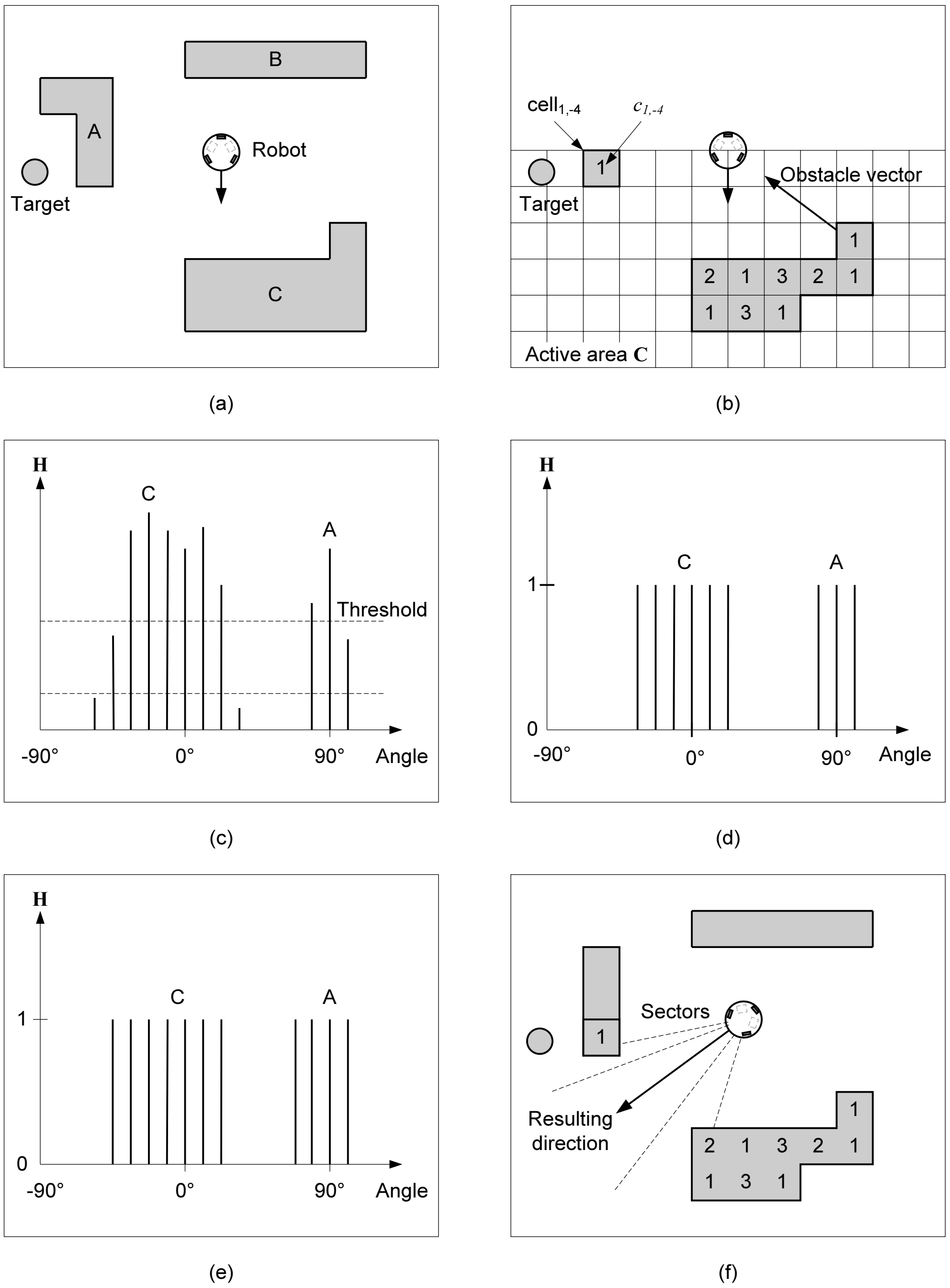


FIGURE 2.8: *Illustration of the VFH+ algorithm. The current situation (a) is transformed into a histogram grid in which the cells are assigned certainty factors (b). The histogram grid is then transformed into a primary polar histogram (c) which is transformed into a binary polar histogram (d). By taking the kinematics of the robot into account, the binary polar histogram is transformed into a masked polar histogram (e), which is used for determining the posterior bearing and speed of the robot (f).*

polar histogram is transformed into a binary polar histogram defining whether a sector is either “free” or “blocked” (Figure 2.8(d)). The binary polar histogram is transformed into a masked polar histogram taking into account the kinematics (steering capabilities and speed) of the robot (Figure 2.8(e)). However, the Robotino® being holonomic, this will have little or no effect in the present system.

When determining the steering angle, the valley, of the masked polar histogram, which most closely matches the direction to the target is chosen. Within the chosen valley the direction is chosen as the mean of the leftmost and rightmost obstacle free sectors, if the valley is characterized “narrow” (Figure 2.8(f)). If the valley is “wide” i.e. above a certain angular limit, the boundaries with which the direction is calculated, are determined by the obstacle on one side and the angular limit on the other. The speed of the robot is proportional to the distance to the nearest obstacle.

Limitations of the VFH+ algorithm

VFH+ is a local path planner which plans the path to a given target based on immediate sensor data. This entails, that the robot might get trapped if a dead-end situation occurs in which the sensory information provides no possible path for the robot to choose. However, the `vfh` driver handles this problem by letting the robot reverse its way out and invoke a global path planner for a new target. Furthermore, not being based on a global map, the VFH+ algorithm is not necessarily choosing the optimal path to the target.

The VFH+ is designed for avoiding static or slow moving obstacles. The present scenario in which a person might be present in the environment the VFH+ algorithm is not guaranteed to perform well, and must thus be subject to testing in order to measure its performance. An approach which might be considered in case an optimization of the VFH+ is needed, is described in [Huiliang and Ying, 2003].

Experiments by which the Pilot functionality has been verified are described in Section 4.4.

2.3.3 Navigator

Having described the reactive navigating functionalities of the robot, this section treats the analysis of the Navigator, providing the robot with the deliberative functionality of performing path planning towards a desired goal.

As in the cases of the Pilot and Localizer functionalities, it has been chosen to exploit the possibilities in path planning offered by existing drivers for *Player*. Thus, tying together the previously described `vfh` and `amcl` drivers with the `wavefront` path-planning driver, forms a "navigational trinity" offering a "global goto" ability to the system [PlayerProject, 2006]. This is illustrated in Figure 2.9.

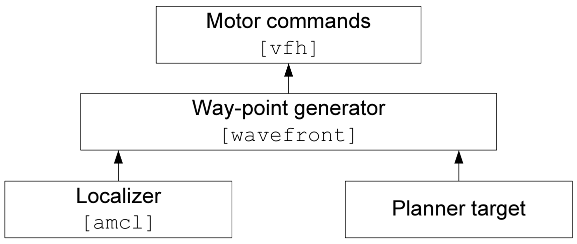


FIGURE 2.9: *Complete navigational solution in terms of Player drivers offering the abilities of a Localizer, Navigator and Pilot.*

The `wavefront` driver

A use-case diagram of the Navigator is illustrated in Figure 2.10.

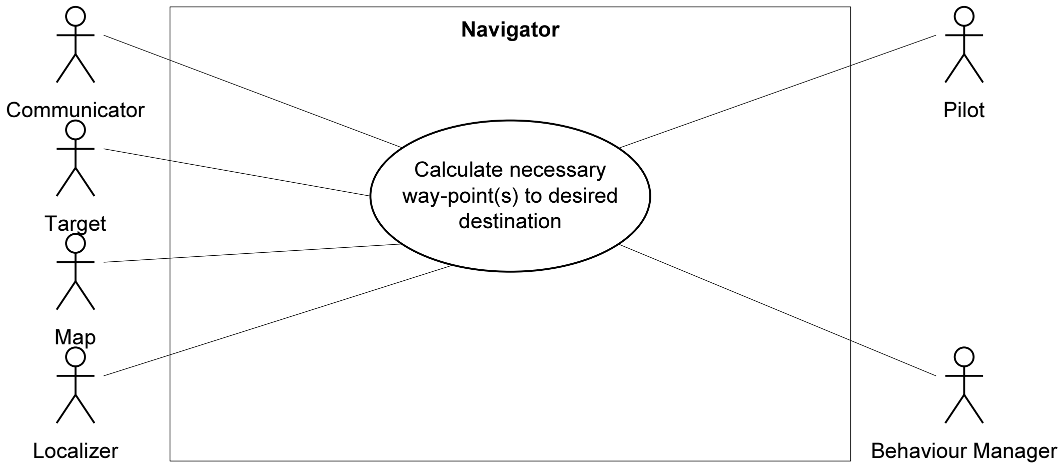


FIGURE 2.10: *Use-case diagram of the Navigator functionality planning routes from a map of the given environment. Notice, that the Navigator provides the Pilot functionality with way-points to follow. Furthermore, the way-points are delivered to the so-called Behaviour Manager, controlling the robot's behaviour as described in Section 2.3.7. The Target and Communicator inputs represent generated and user (obtained from communication) targets, respectively.*

As seen from the figure, the path-planning driver `wavefront` takes as input both the position of the robot provided by the `amcl` localizer and target locations. From these inputs, along with a provided map of the given environment, way-points are generated for the robot to follow, and applied to the robot through the obstacle-avoiding `vfh` pilot.

The `wavefront` driver generates these way-points, by utilizing the so-called Wavefront algorithm involving the steps as illustrated in Figure 2.11.

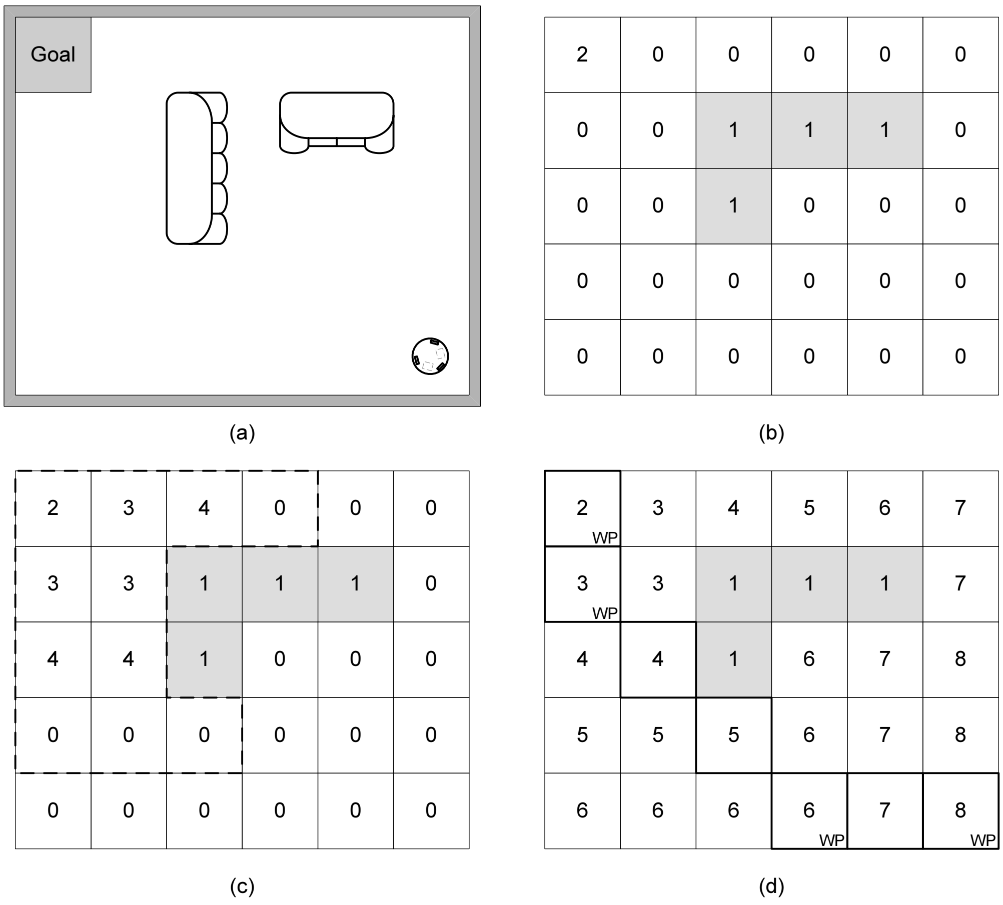


FIGURE 2.11: *Illustration exemplifying the different steps constituting the Wavefront algorithm, applied using an* 8*-point connectivity scheme. By initially assigning a fixed value for all known obstacles, the Wavefront algorithm starts from the robot's current position (cell value* 8*), and increments all adjacent free cells (cell value* 0*) according to the applied connectivity scheme. This procedure is repeated until all free cells have been altered. Finally, a trajectory is drawn by following decreasing cell values from the one currently occupied by the robot.*

Starting from Figure 2.11(a), an example environment is illustrated where the depicted robot is to guide the person safely, i.e. avoiding all obstacles, towards the goal zone. This is achieved by applying an occupancy grid of cells on the map as illustrated in Figure 2.11(b), and assigning e.g. ones in all occupied cells. Next, all cells are filled with values by starting a wave, hence the name Wavefront, from the goal destination towards the robot using either a 4-point or 8-point connectivity scheme (denoting the amount of surrounding cells embodied in the evaluation). Thus, starting from the cell currently occupied by the robot, all adjacent free cells according to the chosen connectivity scheme are incremented. An intermediate step of such filling of cells using the 8-point connectivity scheme is illustrated in Figure 2.11(c). Finally, as illustrated in Figure 2.11(d), when no free cells remain, a trajectory is drawn by starting from the current position of the robot and following cells with lower values. Way-points are placed where straight lines of cells are broken. Thus, one or possibly multiple eligible routes are provided by the Wavefront algorithm, ensuring that no deadlock situations can appear due to the wave of incremented cells being associated with a connectivity scheme. The *Player* implementation singles out a preferred route by associating a potential fields like cost grid on top of the one generated by Wavefront. By combining the two cost grids, an optimal path is found in terms of both distance to goal and distance to obstacles.

The functionality of the Navigator has been verified by experiments as described in Section 4.4.

2.3.4 Person Detector

While roaming a given environment, the robot must constantly be aware of its surroundings. Apart from the previously described functionalities of the Pilot, Navigator and the Localizer, enabling the robot to plan routes, avoid obstacles and keep track of its location, this section treats the functionality of detecting people. Due to the Person Detector functionality not being the main focus of this project, it is furthermore chosen to keep the implementation of this functionality simple.

A use-case diagram of the Person Detector is presented in Figure 2.12, illustrating the various functionalities and procedures involved in the task of detecting a person.

Two drivers in *Player* are usable for the task of person detection, namely a shape tracking driver `simpleshape` and a colour tracking interface `cmvision` based on the CMVision (Computer Machine Vision) algorithm [Bruce, 2006]. Both of these modules exploit the camera mounted on the Robotino®, and provide the *Player* interface `blobfinder`. "Blob" referring to regions in an image that are either brighter or darker than the surrounding.

For person detection, the functionality of blob detection is superior, since tracking a certain shape on a person would require the robot to be quite close. The `simpleshape` driver seems more suited for e.g. communicating with people, where close encountering is an obvious necessity. Hence, the `blobfinder` driver is chosen. Furthermore, for the `blobfinder` to be useful, the person will be marked with an easy distinguishable colour

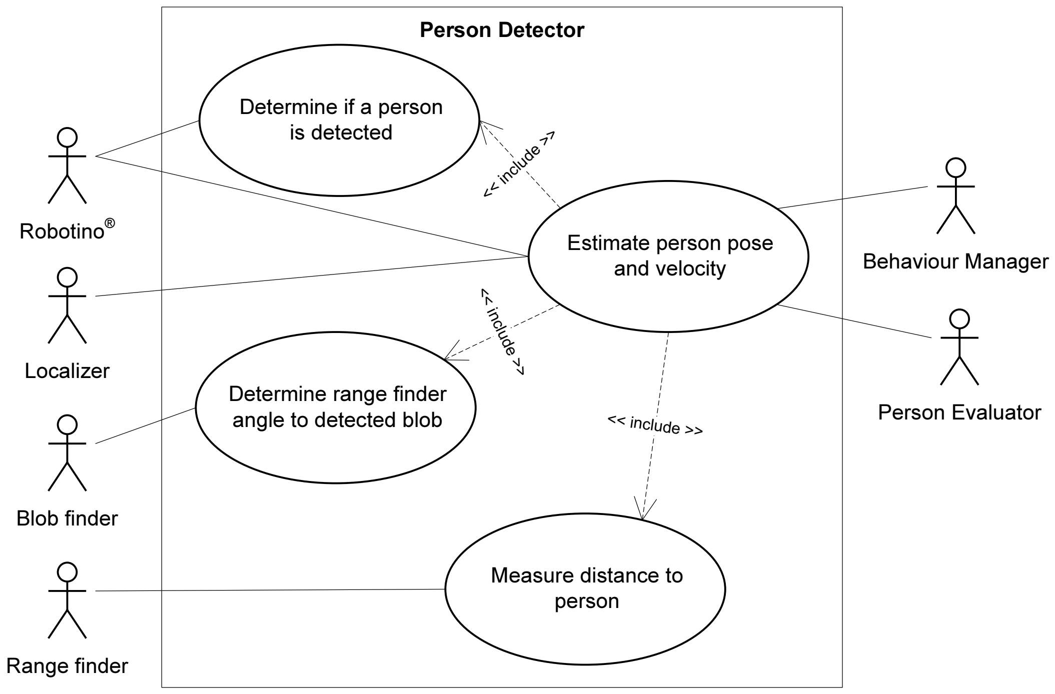

Person Detector
Determine if a person is detected
Estimate person pose and velocity
Determine range finder angle to detected blob
Measure distance to person
<< include >>
<< include >>
<< include >>
Robotino®
Localizer
Blob finder
Range finder
Behaviour Manager
Person Evaluator

FIGURE 2.12: *Use-case diagram of the Person Detector functionality. Inputs from range finder, blob finder, and the Robotino® (odometry data) are used in determining the pose and velocity of a detected person in the robot frame. Furthermore, the Localizer input is used to provide the person's position in world frame coordinates.*

marking.

Subsequently to detecting the person using the Blob finder, the pose and velocity must be determined as indicated in Figure 2.12. The pose and velocity must be determined in co-ordinates relative to the robot frame as well as in co-ordinates relative to the world frame. The reason for this, is that the world co-ordinates of the person is needed to determine his/her position in the environment, and the robot relative co-ordinates are to be used during interaction, in which precise mutual behaviour is important.

The co-ordinates of the person in the robot frame can be determined by fusing the sensor data from the camera and the range finder. Due to the desired simple implementation, traditional advanced approaches to fusing sensor data (as e.g. Kalman filtering) is replaced by more simple methods. Furthermore, the velocity of the person is calculated by simply differentiating the position estimates. However, when calculating the velocity of the person, the motion of the robot must also be taken into account. Doing otherwise, will make the robot relative co-ordinates error-prone due to the possible contribution from the robot's own motion.

To keep the estimation of the person's heading simple, it is chosen to regard the direction of the person's velocity vector as the person's current heading, rather than e.g. applying different colour markings to the person. The Person Detector has been verified by experiments described in Section 4.3.

2.3.5 Person Evaluator and Trainee

Having treated the actual recognition of a person in the environment, this section treats the robot interaction in terms of person evaluation and learning.

Starting from the Trainee, this functionality is, as described in Section 1.2, to be implemented on the basis of Case-Based Reasoning (CBR). What CBR does, is that it basically allows the robot to understand a new problem/situation, by referring to past experiences. Hence CBR, as the name infers, is implemented by defining a case resembling the experiences the given system, in this case the robot, would face. In addition to this, the ability to remember past experiences is implemented by the development of a case library.

Apparent from the above, the ability to evaluate the person detected is needed in order to define a case, while the ability of a trainee is implemented through the CBR property of improving performance based on past experiences. Figure 2.13 illustrates the task of the Trainee by a use-case diagram.

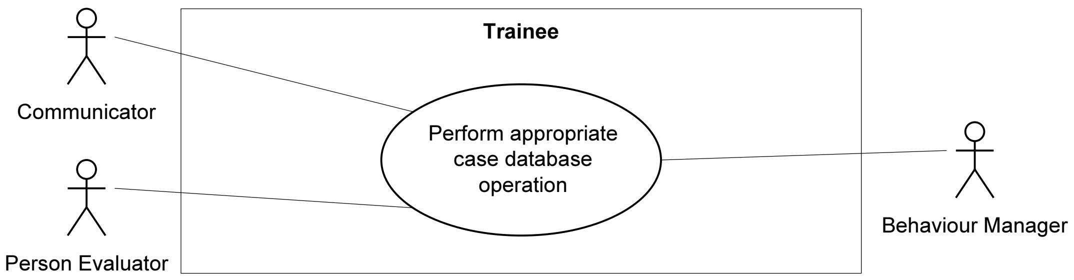


FIGURE 2.13: *Use-case diagram of the Trainee functionality performing case database operations according to the CBR method. Input from Person Evaluator denotes person characteristics used in the case definition, while the Communicator is used to receive the final outcome of whether the detected person is interested in interacting or not.*

Case-Based Reasoning

According to [Kolodner, 1993] a case is a "contextualized piece of knowledge representing an experience that teaches a lesson fundamental to achieving the goals of the reasoner". Thus, when e.g. the robot faces a new problem/situation, representing a case is basically the task of extracting those features of the problem identified as the most significant in relation to determining either the solution, the outcome or both. Having decided on the case representation, CBR essentially involves two parts, namely recalling and interpreting past experiences [Kolodner, 1993; Delany, 2006]:

Recalling or retrieving past experiences, is in CBR terminology defined as the indexing problem described as "the core of case-based reasoning". Thus, a satisfying solution to the indexing problem results in a system capable of retrieving experiences in an effective manner, when facing similar conditioned experiences.

Interpreting or reusing a recalled case, is done by comparing it with the situation currently faced. If a good match or matches exists in the case library, there should be no immediate reason for interpretive processes. A situation referred to as null adaptation. On the other hand, when facing a problem difficult to match in the library adaptation of cases is necessary.

When the solution output from the reuse process is known, further case processing must be made. Also known as revision and retention, such further processing is essentially what constitutes the learning capability of a CBR system. Revision is partly the process of evaluating the solution output, and partly the process of diagnosing and repairing the possible discrepancy. The output of the revision process is a revised case, which is to be retained in memory of the system. Thus, the process of retention is basically to archive the results of an applied case if proven useful.

The above description of the CBR method, is summarized in Figure 2.14

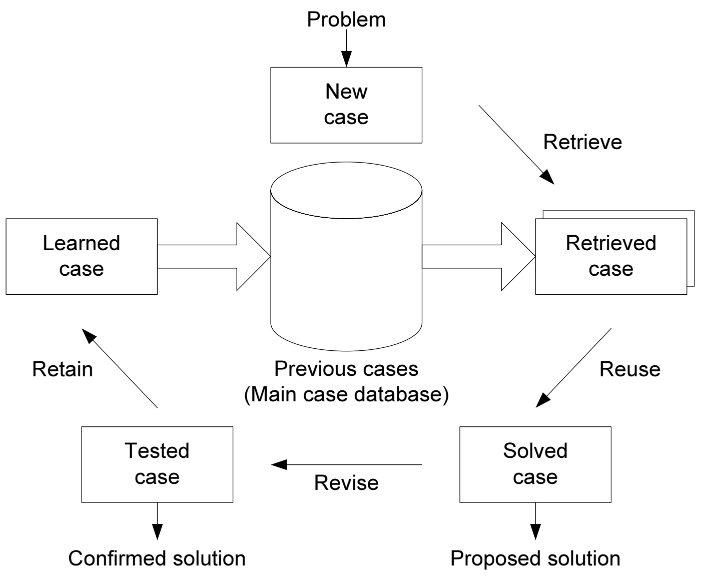


FIGURE 2.14: *Illustration of the (4R) CBR cycle involving the processes retrieve, reuse, revise and retain [Aamodt and Plaza, 1994].*

Person evaluation

Apparent from the above, an essential task of CBR is the determination of a case definition. As also indicated, since the Trainee is to improve the robot's behaviour towards detected persons, the case must somehow reflect certain characteristics of those people detected. This is essentially the task of the Person Evaluator, illustrated by a use-case diagram in Figure 2.15.

Since the only data available on detected persons is the output of the Person Detector, the evaluation possibilities of the Person Evaluator are limited to only concern motional characteristics. However, according to [García-Rojas et al., 2006] a lot of information on a human's state of mind, can be extracted from the way he/she behaves/moves. Based on this statement, it seems reasonable to believe that case inclusion of a person's motional characteristics, provides for satisfactory differentiation in terms of whether a detected person is interested or not.

The implementation of the Person Evaluator is described in Section 3.4.2, whereas the verification of its functioning is included in the experiments documented in Section 4.3.

The complete case definition is presented in Section 3.4.3.

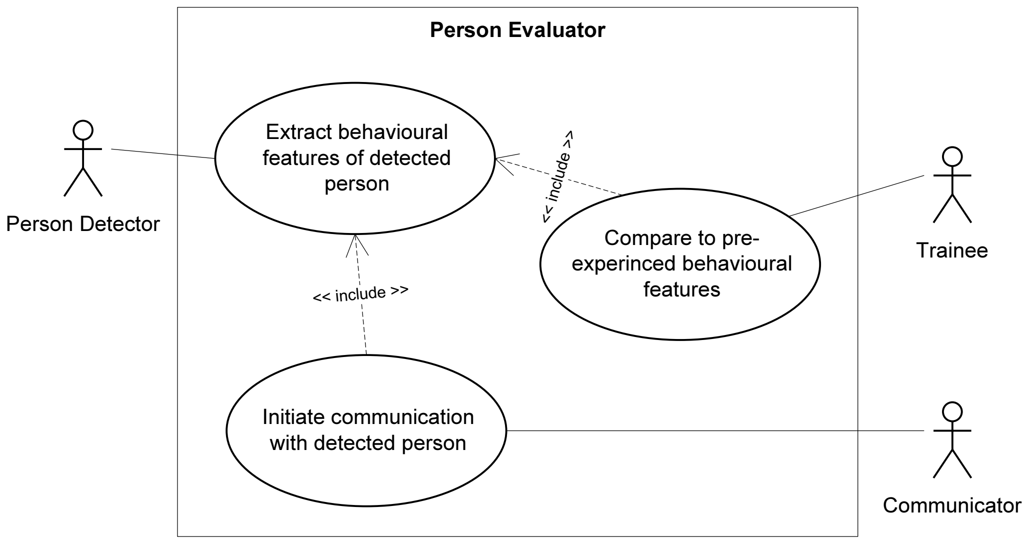

Person Evaluator
Extract behavioural features of detected person
Compare to pre-experinced behavioural features
Initiate communication with detected person
<< include >>
<< include >>
Person Detector
Trainee
Communicator

FIGURE 2.15: *Use-case diagram of the Person Evaluator functionality. Using input from the Person Detector, the Person Evaluator determines certain characteristics of the detected person to be included in the case description for the Trainee. Furthermore, the Person Evaluator must notify the Communicator, whenever the person is within a certain range allowing for a conversation to be made.*

2.3.6 Communicator

As indicated on the use-case diagram of Figure 2.16, the purpose of the Communicator functionality is to enable the robot to engage in conversations with encountered persons. Most importantly, the robot must be able to communicate in a basic and instinctive human language.

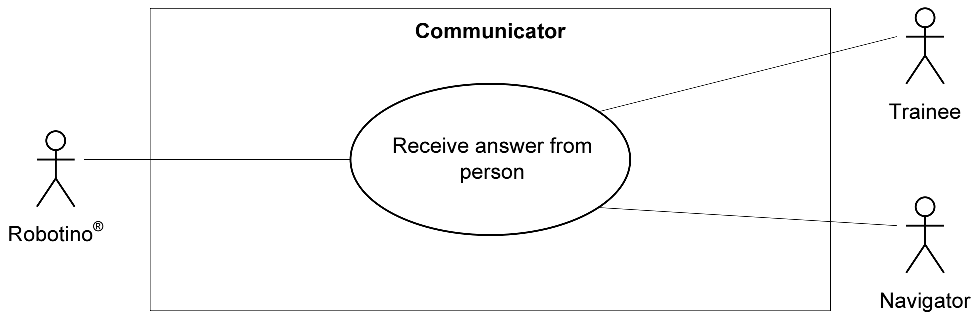


FIGURE 2.16: *The functionality of the Communicator illustrated by a use-case diagram. The task of the Communicator is to enable the robot to conduct two-way communication with encountered persons.*

In the robot research community, a variety of different techniques for communication are investigated, ranging from recognition of simple symbols, to more advanced recognition solutions involving human gesture and speech [Breazeal, 2002]. Starting from the discussion on the profitable aspect in the ongoing endeavour to make robots as human alike as possible, robot communication must naturally be exposed to the same degree of disagreement. Thus, one could argue, that a limited set of gestures or spoken vocabulary, does not necessarily improve the conditions for successful communication, since the set of choices does not conform with the advanced communicative skills of human beings. Thus, the gestures to be performed and the language to be spoken, would seem unnatural and mechanical to a human being. However, in some scenarios of e.g. tour guiding robots, especially spoken communication would undoubtedly be of great value.

To provide the Robotino® with human understanding capabilities, it has been chosen to utilize the IR sensors mounted on its rim. Such realization of the robot's communicative skills naturally introduces some restrictions on the conversation to be conducted, since the sensors only provide means for receiving answers like "yes" or "no". Consequently, in order to establish a two-way communication, the robot requires a functionality to pose questions for the encountered person to answer.

Such functionality has not been considered in the scope of this project, since the focus is aimed at understanding a persons behaviour. Thus, an implied condition for the person when engaging in robot communication, is to tell the robot whether he/she in fact did want to communicate or not.

2.3.7 Behaviour

In order for the robot to function in a human environment it is crucial, that the robot is accepted by humans. Otherwise, if e.g. humans are scared by the robot, it will not be able to perform its assistive tasks. According to prior research [Butler and Agah, 2001], the behaviour of a robot can advantageously be adapted to exhibit a human-like behaviour, if the robot is to be accepted by humans. In other words, if the robot acts like a human it is more likely to be accepted as a social entity in a human environment. A number of studies have been made on this particular subject and are described in e.g. [Butler and Agah, 2001], [Koay et al., 2006], [Huettenrauch et al., 2006], and [Walters et al., 2005]. Based on these studies, a number of rules regarding the robot behaviour in relation to spatial relationships can be stated. The rules relate to the proxemics of human interaction and are defined by using the so-called Hall zones, which along with the principle of proxemics are described in [Hall et al., 1968]. The Hall zones, which originally arose in studies of human-human interaction, have also proven eligible in HRI [Koay et al., 2006] and is shown in Figure 2.17.

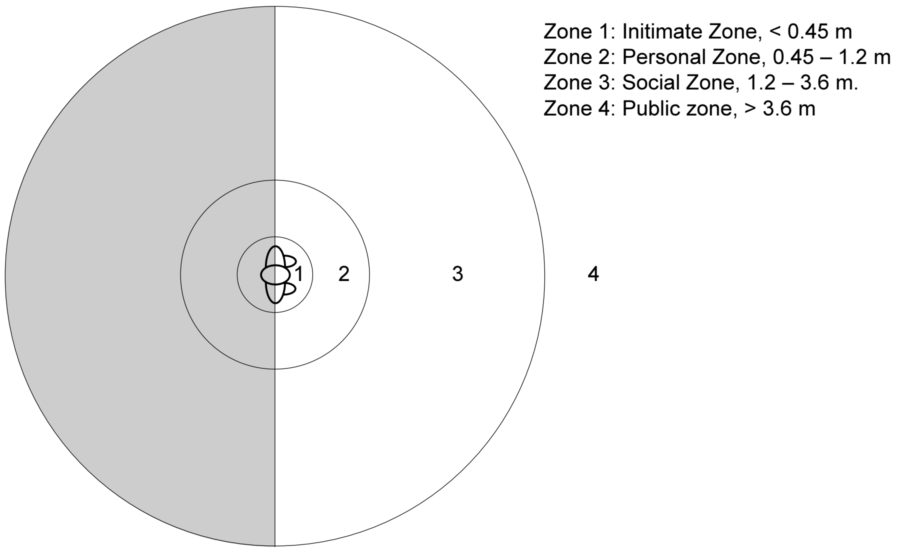

Zone 1: Initimate Zone, < 0.45 m
Zone 2: Personal Zone, 0.45 – 1.2 m
Zone 3: Social Zone, 1.2 – 3.6 m.
Zone 4: Public zone, > 3.6 m
1
2
3
4

FIGURE 2.17: *A person with his associated Hall zones. The grey-coloured area is off-limits for the robot due to a resulting human discomfort.*

The rules governing the spatial behaviour of the robot are listed below:

- The grey-coloured area is off-limits to the robot due to people generally feeling discomfort when the robot moves behind their back.

- Zone 4 (Public Zone): The robot can move freely
- Zone 3 (Social Zone): The robot must adjust the speed and keep inside the person's field of view. The robot is not allowed to approach the person directly. Instead an indirect route must be chosen. The speed must not exceed $0.5\ \frac{\mathrm{m}}{\mathrm{s}}$.
- Zone 2 (Personal Zone): The robot is only allowed to be inside this zone if communication with the person is to be carried out. When communicating, the robot must stop in front of the person.
- Zone 1 (Intimate Zone): The robot is not allowed to be inside this zone. If the person moves, such that the robot enters this zone, the robot must move outside again.
- The behaviour of the robot must not appear too machine-like. Hence, its movements should be smooth and sudden changes in speed and direction should be avoided.
- Wherever possible, the robot must avoid moving through zones in which the person cannot see it.
- The robot must never be on a collision course with the person.
- A small robot is preferred rather than a human size robot. According to [Butler and Agah, 2001] this is due to a human size robot having a more intimidating effect on humans.
- To strengthen human perception of the robot, it should be able to express its mood. Preferably both in terms of an emotional display and by adjusting its behaviour to the current mood.

The spatial behaviour is closely related to the navigation of the robot. Hence, the above specified spatial rules are to be implemented as *Player* drivers. It has been chosen to develop two drivers being Behaviour Manager and Velocity Manager. One for handling the navigation while the robot interacts with a person, and one for handling the robot's velocity in general. Use-case diagrams of the two drivers are seen in Figures 2.18 and 2.19, respectively.

The Behaviour Manager is must take over the navigational features of the robot, when a person is detected. Otherwise, the Navigator and Pilot will regard the person as an obstacle, thus avoiding him/her.

As seen from the listed rules, the robot must adjust its speed according to its distance to the person. This limitation must be performed whether the robot is interacting with the person or not. Hence, it is necessary that the Velocity Manager can be invoked independently of the Behaviour Manager.

For the developed *Player* drivers to function, data regarding the spatial features of a detected person is needed. Furthermore, the drivers are not to be invoked at all times.

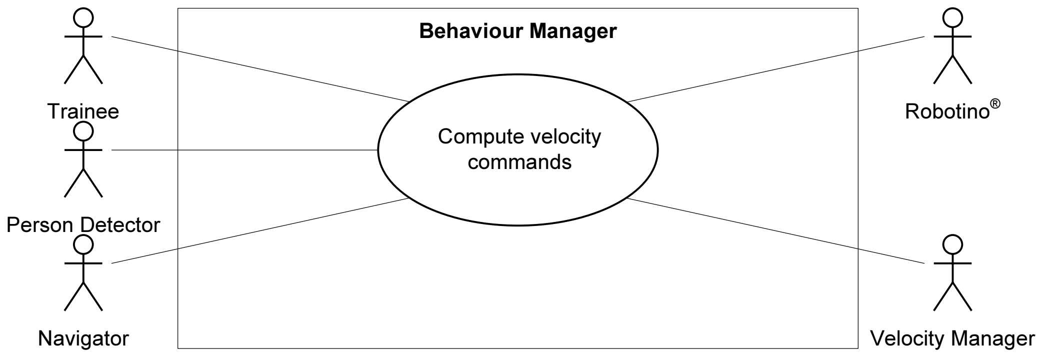


FIGURE 2.18: *Use-case diagram of the Behaviour Manager which handles the navigation while robot is interacting with a person.*

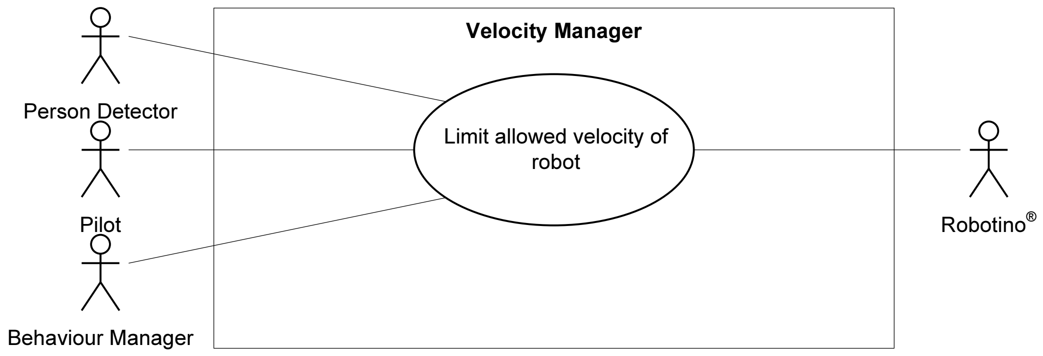


FIGURE 2.19: *Use-case diagram of the Velocity Manager. The Velocity Manager handles the velocity of the robot according to the distance between the robot and the person.*

Hence data must be provided to the drivers indicating invocation. The part of the system, which handles the communication is presented in the following section.

2.4 Controller

2.4.1 Controller objectives

The main objective of the Controller is to interconnect the functionalities described in the preceding sections.

It has been chosen to implement such an interconnecting functionality by the use of a *Player* client, thus allowing the development of the Controller to be performed with the only requirement of supporting the client interface. Since *Player* comes bundled with a C++ client library `libplayerc++`, this library has been chosen as the base upon which the Controller will be developed.

As indicated from the use-case diagram of Figure 2.20, all of the above described functionalities operate at different robot stages, e.g. when the robot is roaming, or communicating. Thus, the Controller must contain a method for supervising (monitoring and controlling) the different states as listed below.

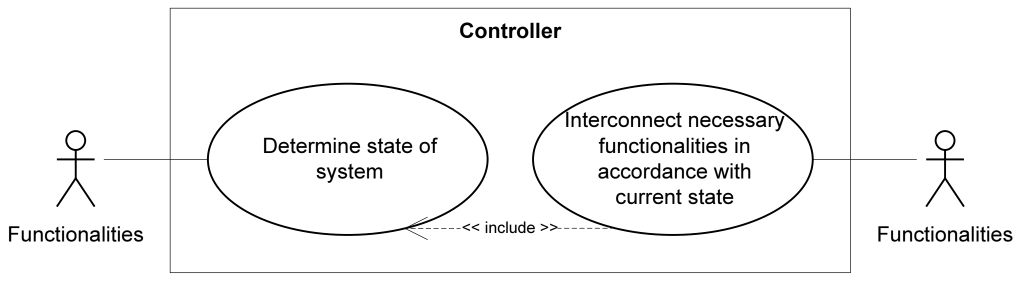

Controller
Determine state of system
Interconnect necessary functionalities in accordance with current state
<< include >>
Functionalities
Functionalities

FIGURE 2.20: *Use-case diagram of the Controller, used to interconnect all the functionalities described in Section 2.3.*

- Localizing - when the `amcl` driver reports unsatisfactory levels of certainty.
- Roaming - when the robot is moving around the environment looking for people to investigate. Note, that in the state of roaming, the Controller must be able to distribute targets inside the given environment, laying more around the positions where it has encountered the most people exhibiting interested behaviours.
- Approaching - upon person detection the robot moves closer to the person in order to verify his/her heading.
- Evaluating - having verified the heading of the person, and still able to get closer, the robot evaluates the person's behaviour.

- Communicating - if the person exhibits interested behaviour and "allows" the robot to come close, the encounter results in a communication between person and robot.
- Guiding - if the person needs guidance, the robot must be able to move to the desired destination.
- Returning - in case of failure or low battery, the robot must return to base.

State-supervision

Considering the Controller as a finite state machine, requires definition of the various edges/conditions controlling the legal transitions of the machine. Studying Figure 2.21, the states listed above have been depicted along with the identified transition conditions to be described in the following.

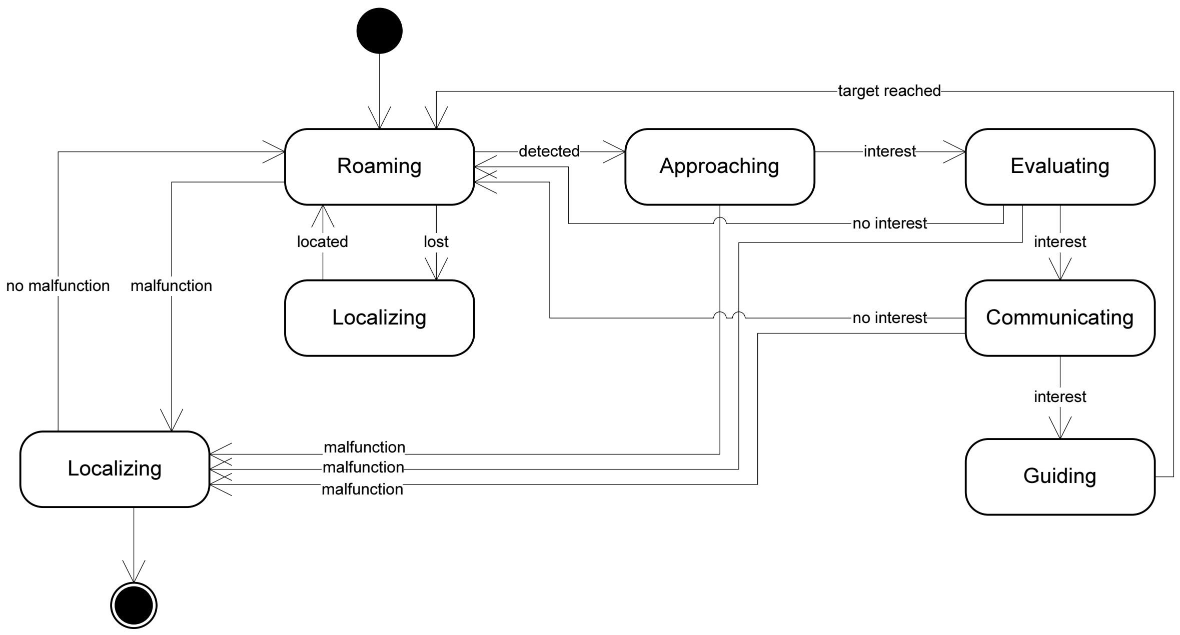


FIGURE 2.21: *Illustration of the supervisory state monitoring to be performed by the Controller. Note, that the transition from state Returning to Roaming requires external robot assistance, in order to cope with the given malfunction, e.g. recharging the batteries, or fixing a sensor.*

lost

Covers the situations where the Localizer (`amcl` driver) reports significant pooled pose variance values, signalling that the current robot pose estimates are influenced by a certain degree of uncertainty. The reason for only including the lost transition condition in the state of roaming, is that the states of Approaching, Evaluating and Communicating all are handled in robot frame coordinates, and thus not vulnerable to errors in the global localization. Furthermore, during Guiding it is assumed that the `amcl` driver, while the robot moves through the environment, will be provided proper differentiating laser measurements to maintain a relative good position estimate.

located

Having performed localization and again reached an acceptable level of pose certainties, the robot fulfils the located transition condition, and returns to roaming the environment.

detected

This condition covers the situation where a person is detected, and thus must be investigated further by the robot.

interest

If the detected person, while the robot is Approaching, Evaluating and Communicating, exhibits signs of interest, the "interest" transition condition is fulfilled.

target reached

Whenever the robot has finished guiding a person to his/her desired destination, the "target reached" condition is fulfilled, and the robot returns to the state of roaming.

no interest

Contrary to the above, the transition condition of "no interest" is fulfilled when the person exhibits no signs of interested behaviour towards the robot.

malfunction

If e.g. one of the motors fails, or if the battery level gets dangerously low, the condition of "malfunction" is fulfilled. Hence, the Controller should be able to monitor the robot's current health, and to react if any abnormalities are discovered.

no malfunction

If no sign of malfunctioning is showing, the robot is ready for normal operation. By example, if the robot has returned to the base due to a low battery level, the transition "no malfunction" is applicable when recharged.

Having analysed all of the functionalities needed in order to fulfil the overall objective, the following section describes the design of a system structure, tying together the individual robot functionalities.

2.5 Software structure

Having treated the analysis of the various individual elements constituting the overall system solution, this section provides a system overview by presenting an overall system description. The proposed system structure is presented in Figure 2.22.

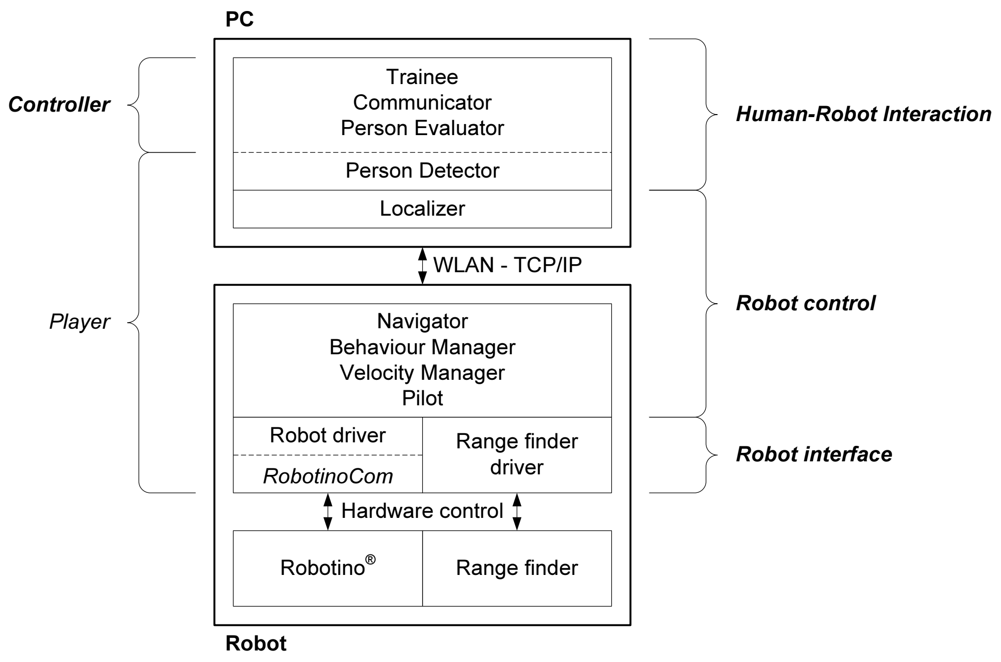


FIGURE 2.22: *Proposed system structure, illustrating the various layers needed to link the Robotino® with the top-most interaction abilities implemented in the client. Note that a player server is implemented on the PC to make use of its computational capabilities in relation to the demands of the* `amcl` *localization driver.*

As indicated on the figure, the system structure is divided into four parts, being ***Robot interface***, ***Robot control***, ***Human-Robot Interaction*** and ***Controller*** which all will be described in the following.

Robot interface

Embodies the *Player* part of the system structure needed for interfacing the Robotino® hardware and the range finder. Thus, in providing hardware interfaces, the ***Robot interface*** lays the foundation,for the remaining elements of higher layers to be built upon.

As seen from the figure, the task of interfacing the Robotino® with *Player* is narrowed down to developing a bridge connection between the *Player* server and the drivers shipped with the Robotino®.

Robot control

The ***Robot control*** is the part of the system where all controlling commands are generated and passed on to the robot through *Player*, thus covering the functionalities of the Behaviour Manager, the Velocity Manager, the Localizer, the Pilot, and the Navigator. As seen from Figure 2.22, the Localizer is placed in the scope of a regular PC. This is done in order to cope with the fairly large amount of computations carried out by the `amcl` driver as described in Section 2.3.1.

Human-Robot Interaction

As indicated on Figure 2.22, ***Robot interaction*** is to be implemented on a regular PC as a client to the *Player* server. Hence, the implementation of the, primarily HRI related, abilities; Person Detector and Evaluator, Communicator and Trainee can be implemented in a wide range of programming languages.

Controller

As described in Section 2.4, the ***Controller*** handles the system's internal communication as well as the overall system supervision.

Having analysed the various parts of the system, the overall software structure is presented in the following section.

CHAPTER 3

Design and Implementation

3.1 *Player* driver architecture

Along with the above described *Player* drivers developed during this project, a number of existing drivers are used. In combination, all drivers constitute a driver architecture. In Figure 3.1, an overview of this architecture is presented along with the interfaces combining the individual drivers.

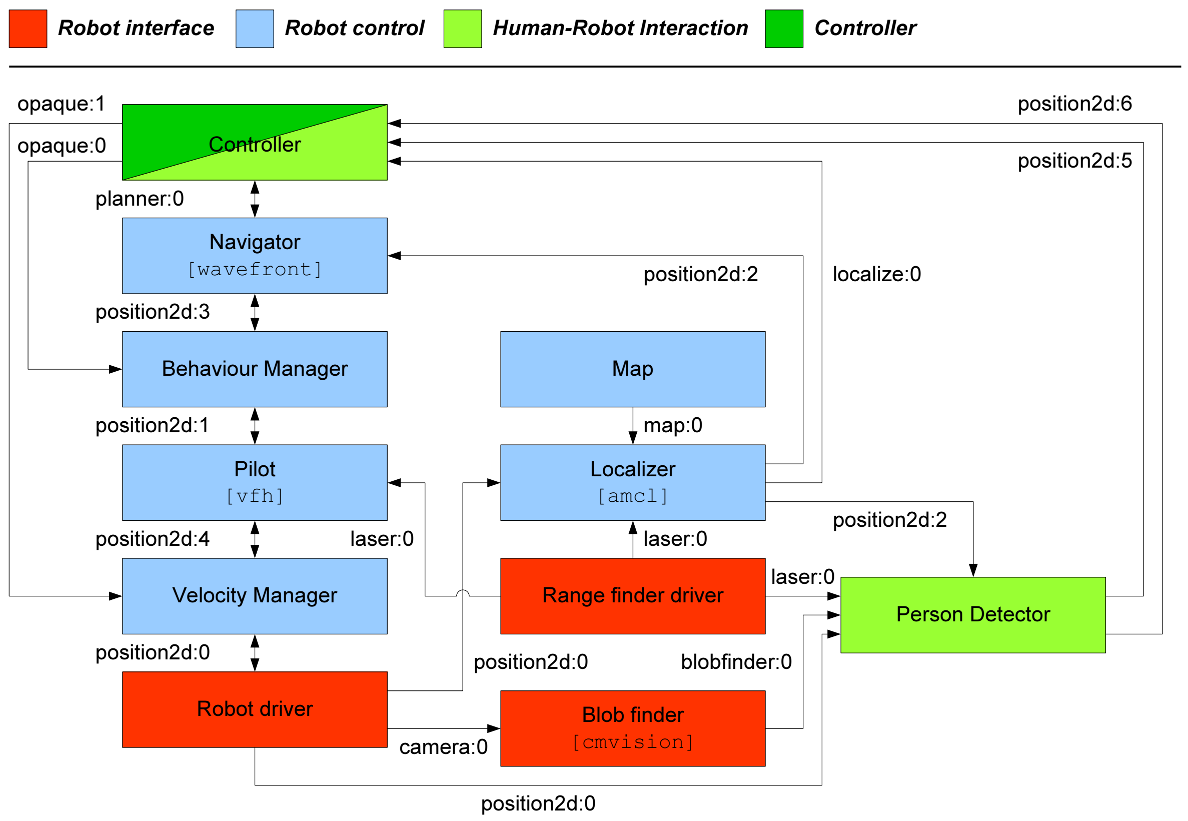


FIGURE 3.1: *The applied Player driver architecture. Each block contains the name functionality, while arrows represent interfaces and direction of data flow. Note, that the driver name is included where existing Player drivers are used.*

As seen in Figure 3.1, some interfaces are used more than once. To provide a comprehension of the use of the interfaces, the data exchanged through each interface is described in Appendix C. A detailed specification of each interface is found in [PlayerProject, 2006].

Referring to the legend of Figure 3.1, the following sections will treat each of these system parts individually.

3.2 Robot interface

3.2.1 *RTLinux* module

As described in Section 2.1.1, interfacing the Robotino® hardware is done through a *RTLinux* layer. Hence, a *RTLinux* module is needed to provide a connection between the user software executing in the Linux user space and the hardware through the *RTLinux* layer in kernel space. The module is written in standard C, primarily utilizing the functions found in `rt_com.h` and `rtl_fifo.h` and the overall principle of the module is seen in Figure 3.2

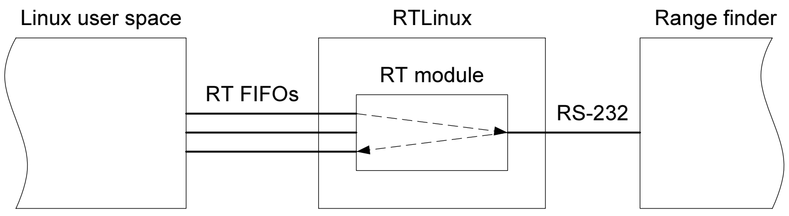


FIGURE 3.2: *Overall operational principle of the RTLinux module. Data is exchanged between Linux user space and RTLinux kernel space through RT FIFOs.*

The communication between Linux user space and *RTLinux* is done through FIFO buffers (RT FIFOs), whereas connecting to the serial port (used for communicating with the range finder device) is done directly from *RTLinux*. Three FIFOs are created: Two for moving serial data between user space and *RTLinux*, and one for moving messages from user space to *RTLinux*. Furthermore, three handlers are set up, of which two handle the actual data movement between the serial port and the data FIFOs. One is invoked by an interrupt from the serial connection, while the other is invoked when data is present in the FIFO. The third handler deals with messages sent from Linux user space, e.g. requests to change the baud rate used for the serial communication.

The following section treats the developed *Player* drivers, which as part of the project contributions, make the Robotino® platform available to the *Player* community.

3.2.2 Robot driver for *Player*

As mentioned earlier, the task of the robot driver is to interconnect the functions of the *RobotinoCom* API with the described framework of a *Player* plug-in driver. The following interfaces are provided by the Robotino® *Player* driver:

`position2d`

Used in controlling mobile robot bases in $\mathbb{R}^2$. Not all functionalities of the interface have been implemented, but only those enabling users to pass commands to the motors, and read data from the encoders. These functionalities are associated with certain messages and requests sent to and from the robot driver providing the `position2d` interface.

The developed driver accepts velocity commands which are applied to the Robotino® through the following *RobotinoCom* function:

```
void setVelocity( double vx, double vy, double omega )
```

`vx` is the velocity in x direction $\left[\frac{\mathrm{mm}}{\mathrm{s}}\right]$
`vy` is the velocity in y direction $\left[\frac{\mathrm{mm}}{\mathrm{s}}\right]$
`omega` is the angular velocity $\left[\frac{\mathrm{deg}}{\mathrm{s}}\right]$

Furthermore, the robot must continuously output its state in terms of velocity and position. The velocity information is retrieved from Robotino® by use of the following function:

```
float actualVelocity( unsigned int motor )
```

where `motor` is the given motor on the robot.

Note that this function `actualVelocity`, in spite of being declared as a **`float`**, returns an integer number specifying pulses pr. millisecond on the encoder.

The `actualVelocity` function returns the velocity of each individual motor. To make this conform with the `position2d` interface, it is translated into the robot's velocity in the x and y direction along with its angular velocity. This translation is done using the Robotino® kinematics, which are derived in the project Wiki. The position is calculated by using integrating the velocity data.

`ir`

The `ir` interface provides access to readings of all nine IR sensors on the Robotino®. The readings are obtained by the use of the following *RobotinoCom* function:

```
float distance( unsigned int n )
```

where `n` denotes the IR sensor, i.e. integers within the sequence $[1;10]$.

`bumper`

This interface provides access to the bumper on the Robotino®, using the following function:

```
bool bumper()
```

If the bumper is pressed the function returns 1, otherwise 0.

`camera`

The following *RobotinoCom* function is used to set up the connection to the camera:

```
void setCameraParameters( const CameraParameters& param )
```

`param` is a struct containing camera parameters.

Having set up the connection, the camera is directly accessed by *Player* through the `camera` interface.

`power`

Used to provide access to the Robotino® state of charge by using the following *RobotinoCom* function:

```
float voltageBatt1plus2() const
```

Source code of the Robotino® driver can be found in Annex 1.

3.2.3 URG driver for *Player*

A *Player* driver (`urglaser`) for the URG-04LX already exists and is distributed with the *Player* distribution. However, the driver needs some modification in order to be usable with the Robotino®. This need for the modification emanates from the fact that the sensor is interfaced through RS-232 compatible serial connection. However, as described in Section 2.1.1 accessing a serial port on the Robotino® can only be by using the *RTLinux* module described in Section 3.2.1.

Therefore, instead of accessing the serial port directly, the modified driver accesses the three serial FIFOs of the developed *RTLinux* module. For that reason, a number of options have been added to the configuration file of the driver. These options are listed in Code 3.1.

```
use_rt_fifo 1
fifo_write "/dev/rtf1"
fifo_read "/dev/rtf2"
```

```
    fifo_message "/dev/rtf3"
    baud 115200
```

CODE 3.1: *Additional options for the modification of the URG-04LX Player driver.*

The first option `use_rt_fifo` is used to specify whether the modification of the driver is to be used or not. If set to 0 the driver connects directly to a serial port. The options `fifo_write`, `fifo_read`, and `fifo_message` are used to specify the hardware addresses of the three FIFOs. Using the `baud` option, the baud rate can be specified. The possible baud rates are determined by the sensor's communication protocol to be 19200, 57600 and 115200 baud.

3.3 Robot control

3.3.1 Behaviour Manager

Based on the analysis regarding human perception of robot behaviour in Section 2.3.7, this section contains the implementation of the robot Behaviour Manager *Player* driver.

As seen in Figure 3.3, the Behaviour Manager is located between the Navigator and the Pilot, and must be invoked when the robot is inside a person's Social Zone. Hence, the Behaviour Manager must, in place of the Navigator and the Pilot, provide the robot with velocity commands. Therefore, the obstacle avoiding capabilities of the Pilot are disabled whenever the Behaviour Manager is invoked.

Figure 3.3 shows that the Behaviour Manager receives `floatPersonIndication` from an `opaque` interface which is used for transferring data from the Controller to the Behaviour Manager. The `floatPersonIndication` is used for indicating whether the Behaviour Manager should be invoked or not. Thus, a value different from -1 will result in the Behaviour Manager being invoked. Otherwise, the output from the Navigator is relayed directly to the Pilot. In the current implementation, only the `floatPersonIndication` is used by the Behaviour Manager. The specific use of the `floatPersonIndication` will be described later in this section.

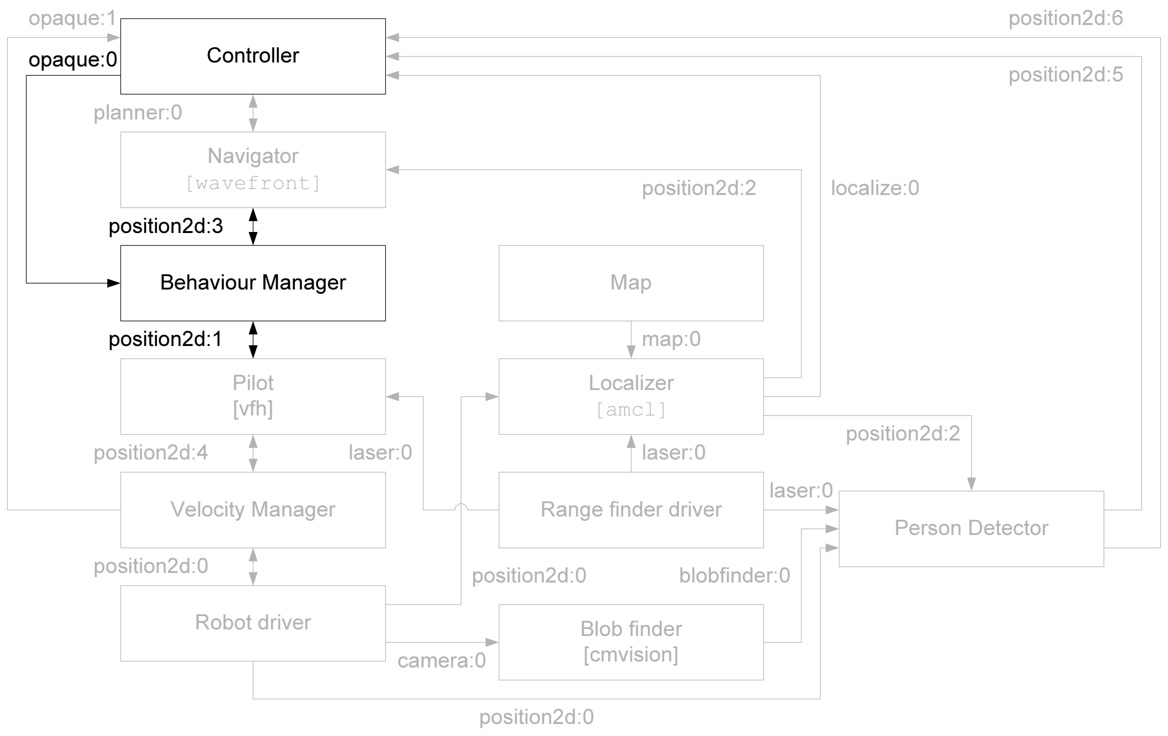


FIGURE 3.3: *Located between the Navigator and Pilot, the Behaviour Manager must be capable of relaying messages when not invoked itself. The data received from the Controller is used in the behaviour algorithm to calculate the velocity commands for the robot.*

Behaviour algorithm

In [Sisbot et al., 2006] and [Sisbot et al., 2005] a method for human-aware navigation is proposed, making use of cost functions to punish the robot when entering specific zones in relation to a human being. This is done by implementing a "safety grid" which is basically a human centred normal distribution in which the value of each grid cell represents a cost. Furthermore, a "visibility grid" is implemented. The visibility grid is "*constructed according to costs reflecting the effort required by the human to get the robot in his field of view*" [Sisbot et al., 2006]. However, a gap exists between the above described approach and the functionalities needed for the work of this project. These functionalities, concerning human-aware navigation are:

- The need for changing the shape and not only the circular size of the grid repelling the robot from the person.
- The desire for attracting the robot towards the person in specific approach angles.

The need for changing the grid shape emanates from the need for the robot to be able to actually get close to the person if so required by the person's behaviour. Furthermore, the single normal distribution causes no difference for the robot when approaching the person from the side or head-on compared to some desirable approach angle in between.

The approach for human-aware navigation proposed and implemented in this project is a *behaviour grid* based on a combination of multiple bi-variate normal distributions. The reason for this, is that normal distributions are computationally feasible and at the same time intuitively perceivable. A bi-variate distribution is given by:

$$f_X(x) = \frac{1}{(2\,\pi)\;|\mathbf{\Sigma}|^{1/2}}\;\exp\left(-\frac{1}{2}\;(x-\mu)^T\;\mathbf{\Sigma}^{-1}\;(x-\mu)\right) \tag{3.1}$$

Where $x \in \mathbb{R}^2$ and $\mu \in \mathbb{R}^2$ and the covariance matrix $\mathbf{\Sigma}$ is a positive semi-definite, and real 2×2 matrix.

The distributions are calculated in the person frame with mean $\mu = (0,0)$ as depicted in Figure 3.4. Furthermore, it is seen that negated distribution has a circular shape and thus is isotropic. The other three distributions have elliptic shapes. The shape of the individual distribution is defined by the covariance matrix $\boldsymbol{\Sigma}$. Defining the $\boldsymbol{\Sigma}$ matrices as seen in (3.2), the entries σ_x^2 and σ_y^2 can be used to adjust the minor and major axis (i.e. the width and length) of the distributions, whereas σ_{xy} can be used for rotating the distributions.

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix} \tag{3.2}$$

The rotation θ of the distribution is given by [Strang and Borre, 1997]:

$$\tan(2\theta) = \frac{2\,\sigma_{xy}}{\sigma_x^2 - \sigma_y^2} \tag{3.3}$$

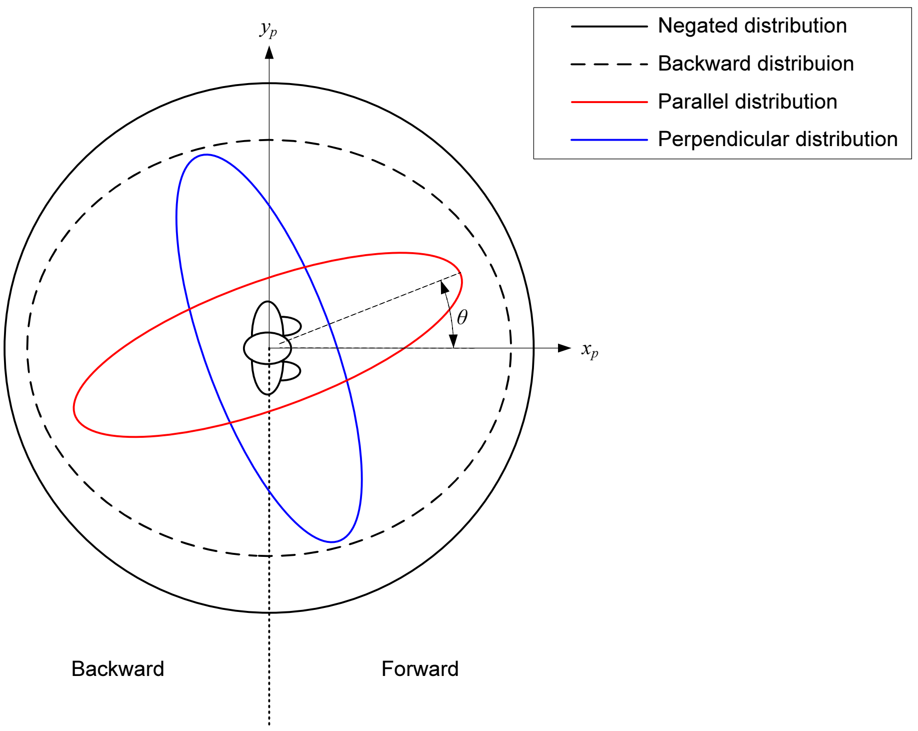


FIGURE 3.4: *The four bi-variate normal distributions used for grid calculation. Note, that when calculating the grid values, only the backward part of the backward distribution, and the forward parts of the parallel and perpendicular distributions are used.*

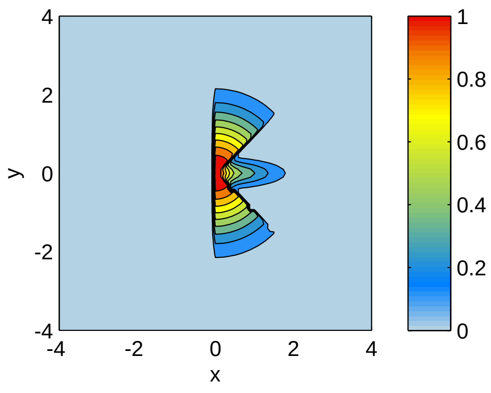
Negated distribution The purpose of the isotropic negated distribution is to attract the robot towards the person, and is used both when the robot is behind and in front of the person. The circular size of the negated distribution is fixed, using $\sigma_x^2 = \sigma_y^2 = 7.5$ and $\sigma_{xy} = 0$.

Backward distribution The anisotropic backward distribution is used to repel the robot when inside the backward part of the person's Social Zone as described in Section 2.3.7. The reason for the shape of this distribution not being completely circular is due computational issues, which will be introduced later in this section. Introducing such dissimilarity, is deemed insignificant to the purpose of the algorithm, due to the distribution still being largest one directly behind the person. The backward distribution is only used when the robot is behind the person and the length and width are fixed at $\sigma_x^2 = 2$, $\sigma_y^2 = 1$ and $\sigma_{xy} = 0$.

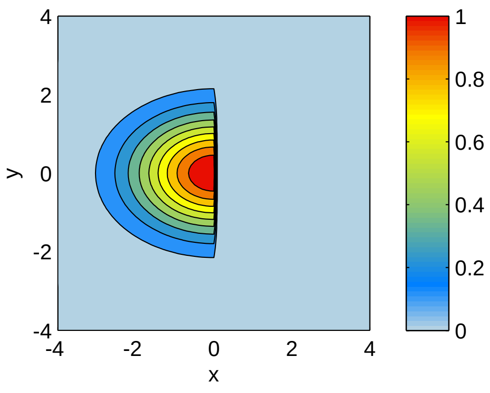
Parallel and perpendicular distributions From the analysis in Section 2.3.7, the robot is not allowed to approach the person directly from the front. Furthermore, approaching the person directly from the sides, might cause discomfort to the person. Hence, the approach angle must be inside $\pm 45°$ relative to x_p. The widths of the forward distributions are fixed at a width similar to that of the backward distribution (i.e. $\sigma_y^2 = 1$) at angles in the intervals $[-90°, -45°]$ and $[45°, 90°]$. Furthermore, the angle θ (as seen on Figure 3.4) is made variable, enabling the robot to approach the person when in front of him/her. This is a necessity when human-robot communication is desired. The parallel and perpendicular distributions are used only, when the robot is in front of the person.

Important to realize, is that changing the length and/or the width of a normal distribution, also affects its magnitude due to the fact that the integral of the distribution is 1. Hence, to equalize the contributions from the individual distributions, the magnitudes are normalized in the following way: The maximum of the backward distribution and the sum of the forward distributions are normalized to unity. Normalizing factors are determined by calculating the magnitude of each distribution at its mean value. The magnitude of the negated distribution along with the distribution widths and lengths are manually tuned based on the desired shape of the resulting grid.

An example of the contours of the resulting grid determined by the forward and backward distributions is presented in Figure 3.5(a). The values of the $\mathbf{\Sigma}$'s have been fitted to make the grid resemble the Hall zones mentioned above. Thus, for the parallel distribution $\sigma_x^2 = 1$ and $\sigma_y^2 = 0.15$, while for the perpendicular distribution $\sigma_x^2 = 0.15$ and $\sigma_y^2 = 1$. Furthermore, θ is fixed at $0°$.



(a) Forward part of the grid.



(b) Backward part of the grid.

FIGURE 3.5: *Contours of the behaviour grid. The grid consists of three parts; A forward part, a backward part, and a negated part, which are calculated independently. The negated part is merely a circular normal distribution and is thus not depicted. The grid co-ordinates are specified in meters, whereas the colour gradient represents the magnitude.*

It is seen, that this part of the grid will prevent the robot from coming too close to the front of the person i.e. co-ordinates in the person frame where x is larger than zero, and y is small. Furthermore, the robot is prevented from approaching the person directly from the sides.

The backward part of the grid (depicted in Figure 3.5(b)) ensures that the robot will not enter the Social Zone behind the person.

The final part of the grid is the isotropic negated distribution, attracting the robot towards the person and is not depicted due to its simplicity.

As described above, the proposed navigation approach makes it possible to use the person's behaviour to alter grid parameters. For this purpose, the input `floatPersonIndication` obtained from the Controller through an `opaque` interface is used for specifying the widths of the forward distributions, and for specifying the rotation angle θ. The `floatPersonIndication` takes on values in the range of -1 to 1 and results from the Person Evaluator's estimation of the person's interest in interacting with the robot. The person being "interested" will result in values above 0.5, whereas lower values between indicates a person being "not interested".

In Figure 3.6, the mapping from `floatPersonIndication` to forward distributions widths and rotation angle is depicted. Note, that the value of the widths can not be 0, since this results in the covariance matrix Σ becoming a zero matrix. Hence, the minimum value of the widths is set to 0.01.

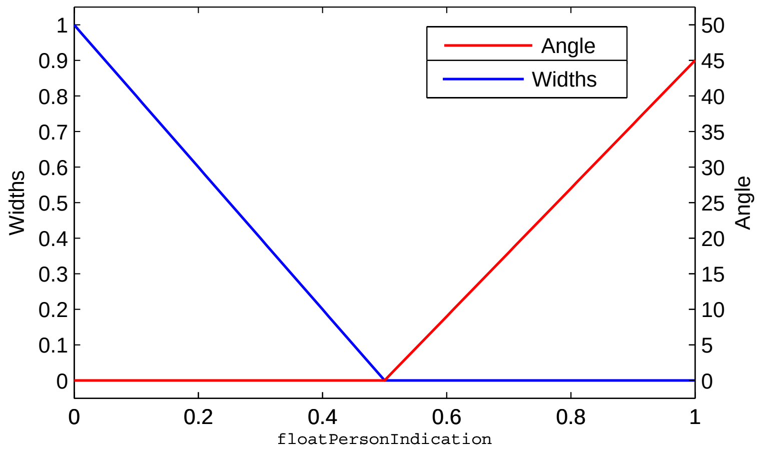
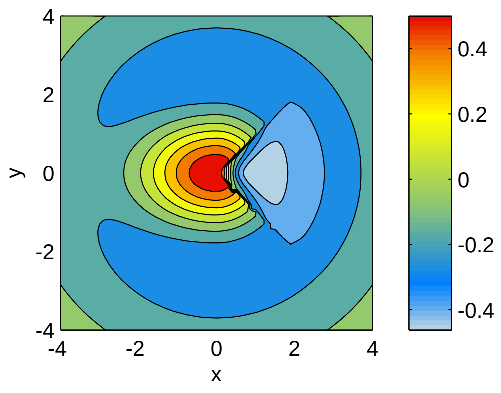


FIGURE 3.6: *The mapping from* `floatPersonIndication` *to the widths and rotation angle of the forward distributions. The minimum of the widths is set to* 0.01 *to prevent the covariance matrix from becoming a zero matrix.*

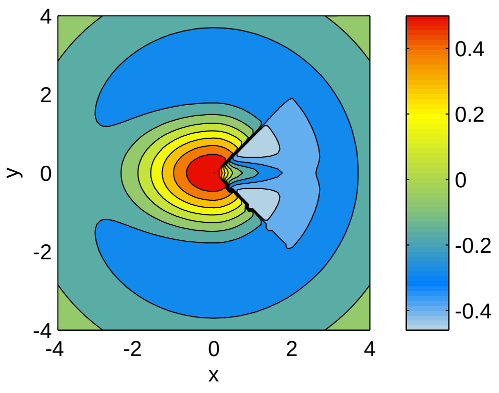
The figure shows, that starting from zero indication, the widths of the distributions decrease to a certain threshold when `floatPersonIndication` increases. Subsequently, the rotation angle is increased until $\theta = 45°$. Figure 3.7 illustrates the effect of varying the `floatPersonIndication` value.

In the case depicted in Figure 3.7(a), the `floatPersonIndication` is 1, representing an interested person. As seen in the figure, the robot will be attracted towards the minimum located directly in front of the person. Figure 3.7(b), illustrates the case in which the `floatPersonIndication` is 0.5. Both the widths of the forward distributions and the rotation angle are small, resulting in the robot approaching the person in an angle of approximately $45°$ before ending in one of the minima. Finally, in Figure 3.7(c), the `floatPersonIndication` is fixed at $0°$, representing a person who is not interested. As seen from the figure, the robot is not allowed to approach the person.

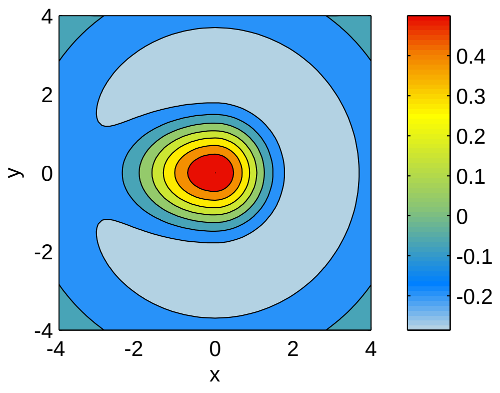
Figure 3.7 also illustrates the reason for the backward distribution not being completely circular. Had it been circular, the grid value would be constant at similar distances from the person. Thus, the robot would not be attracted towards the front of the person.



(a) `floatPersonIndication` $= 1$



(b) `floatPersonIndication` $= 0.5$



(c) `floatPersonIndication` $= 0$

FIGURE 3.7: *The resulting grid governed by the four distributions is depicted at different values of the person interest indication* `floatPersonIndication`*. The grid co-ordinates are specified in meters, whereas the colour gradient represents the magnitude.*

Human-aware navigation

In order to use the above described grid for navigation, the robot must choose a driving direction which will always lead it to a place on the grid with a lower value than the value of its current position, given that the robot is not located in one of the grid minima. This is done by utilizing an eight point connectivity scheme to define candidate points in the robot frame (see Figure 3.8).

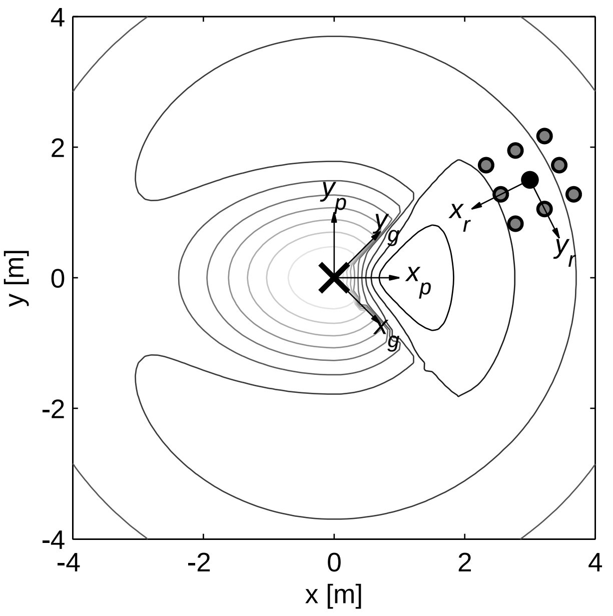
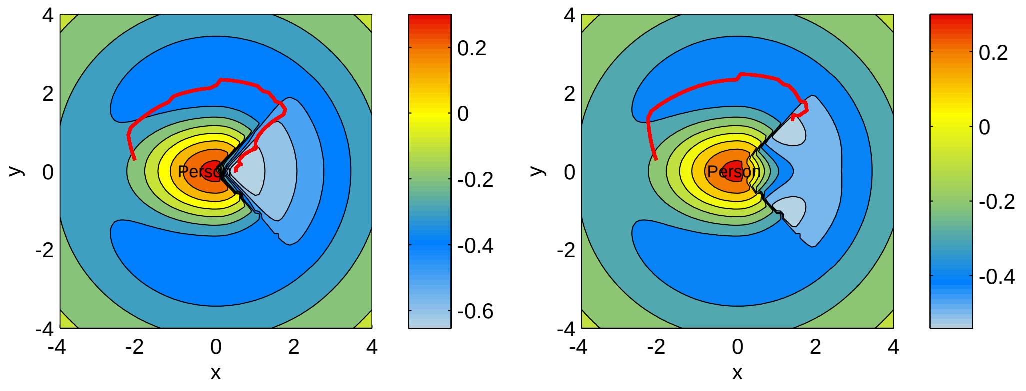


FIGURE 3.8: *When determining the robot's current velocity vector, the candidate points of the eight point connectivity scheme (illustrated with grey circles) adjacent to the current location of the robot in the robot frame are transformed into the person or grid frame, and the grid values are calculated. The vector to the point with the smallest grid value (given that it is smaller than the grid value at the robot's current location is used as velocity vector. The person is illustrated with an ×, whereas the robot is marked with a ●. Note, that the distances between candidate points and robot are exaggerated.*

These points correspond the robot's current position and the eight adjacent positions. The grid values of the candidate points are calculated and the robot is provided with a velocity vector corresponding to the point with the smallest value. Calculating the grid values is done by transforming the co-ordinates of the candidate points into the person frame (also denoted the grid frame) illustrated in Figure 3.8 by (x_p, y_p) and (x_g, y_g) respectively. The person frame is used when



(a) `floatPersonIndication` $= 1$ (b) `floatPersonIndication` $= 0.5$

FIGURE 3.9: *Simulation of the described navigational approach. The axes correspond to the person frame and the path of way-points chosen by the robot is marked by a red line. The initial position of the robot is behind the person and the final position in front of the person. The final position of the robot results from the person interest indication* `floatPersonIndication`*. The grid co-ordinates are specified in meters, whereas the colour gradient represents the magnitude.*

calculating the backward part of the grid, while the grid frame is used when calculating the forward part of the grid. This difference, results from the fact that the backward part is fixed to the person, whereas the forward part may be rotated relative to the person. In Figure 3.9, a simulation of the above described approach is presented, in which the robot is initially located behind the person.
The simulation shows that the robot initially is too close to the person and hence is repelled from the person. When an acceptable distance is reached, the robot follows the contours of the grid until reaching a minimum of the grid. This places the robot close to the person at an angle of approximately $45°$ in the “interested” case (Figure 3.9(b)) and $0°$ in the “not interested” case (Figure 3.9(a)).

When determining the velocity vector of the robot using the above described approach, no considerations are made to whether the velocity is suitable in relation to the distance between the robot and the person. This issue is handled in by the Velocity Manager described in the next section.

3.3.2 Velocity Manager

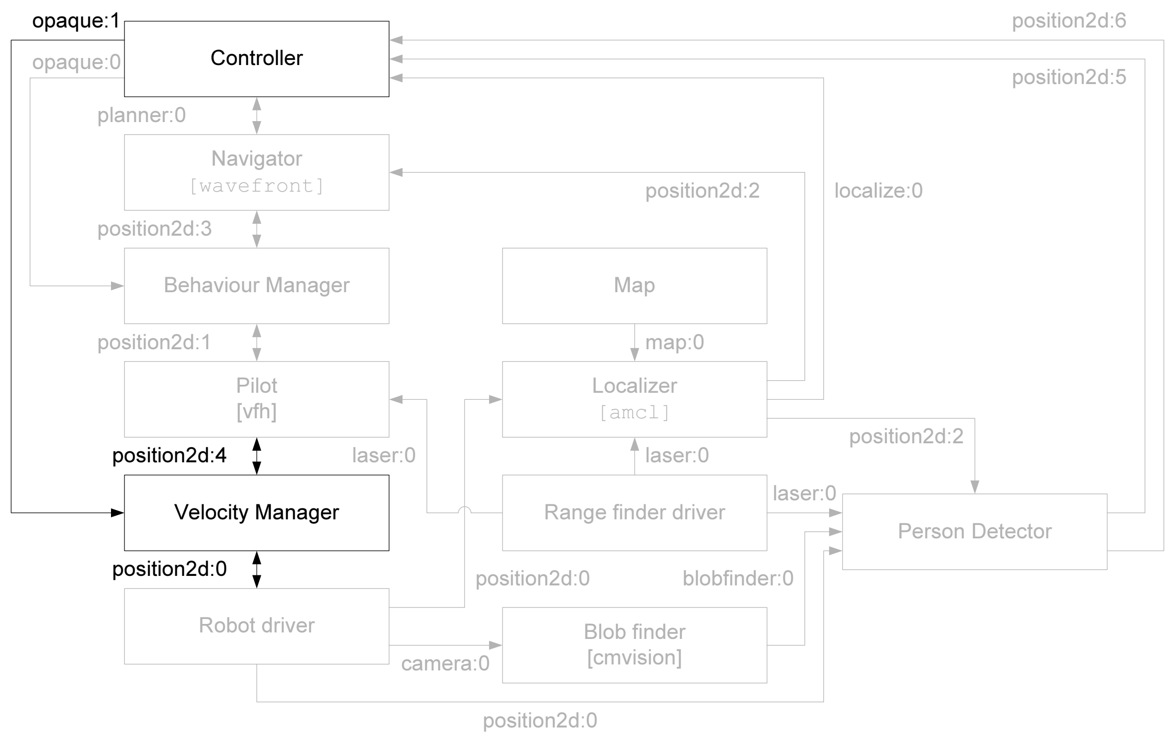


FIGURE 3.10: *The location of the Velocity Manager in the overall system structure.*

As seen in Figure 3.10 the Velocity Manager driver is located between the Pilot and the robot, limiting the velocity commands provided to robot when necessary.

Apart from standard `position2d` interfaces, the `opaque:1` interface is used for transferring data from the Controller to the Velocity Manager. This data is of data type **`float`**, resulting in each value being contained in four bytes to conform with the `opaque` interface of *Player*. In Figure 3.11 the order of the data is presented.

floatMaxVelocityX byte 0-3	floatMaxVelocityY byte 4-7	floatMaxVelocityA byte 8-11

FIGURE 3.11: *Protocol of the `opaque:1` interface.*

In addition, the bytes constituting the `floatMaxVelocityX` are used for indicating whether the velocity should be limited or not. If the value of `floatMaxVelocityX` is -1, the velocity is not limited.

One of the results presented in the robot behaviour analysis of Section 2.3.7, is that the robot velocity should be decreased when approaching a person. This is implemented by use of the Hall zones described in Section 2.3.7. The specific velocity of each zone has been chosen as outlined in Table 3.1.

Hall zone	**Max. velocity $[\frac{m}{s}]$**
Public	1
Social	0.4
Personal	0.2
Intimate	0.1

TABLE 3.1: *Maximum velocities to be maintained by the Velocity Manager. The velocities depend upon the Hall zone in which the robot is currently located.*

Having presented the Behaviour Manager and Velocity Manager, constituting the ***Robot Control*** part of the overall system, the next section addresses the HRI capabilities of the developed system.

3.4 Human-Robot Interaction

3.4.1 Person Detector

From Section 2.3.4 the objective of the Person Detector is to provide an indication of whether or not a person is detected, and in the latter case to also provide motional data of the person. These data consist of a robot frame oriented pose of the detected person along with the person's translational velocities. Furthermore, a world frame oriented pose is provided. The reason for not providing the angular velocity is, that direct measuring of a person's heading is not possible in the current set-up. Thus, the heading can only be calculated based on measured velocities. Consequently, detection of a person's heading while he/she is turning on the spot is currently not supported. Instead, the angular velocity is used for signalling whether a person has been detected or not.

The Person Detector utilizes marker detection by use of the `cmvision` driver along with range finder readings as seen from Figure 3.12.

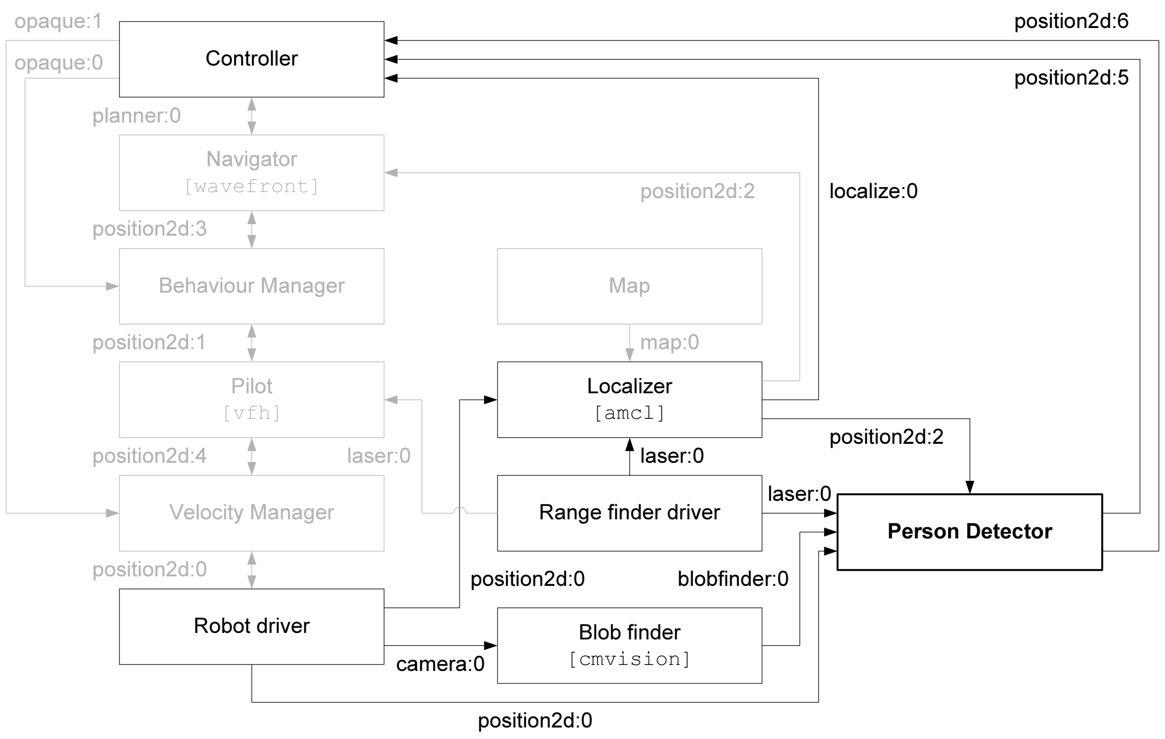


FIGURE 3.12: *Overall system structure where the Person Detector and all connected functionalities are highlighted.*

Estimating position and heading

The approach of acquiring a detected person's position is illustrated in Figure 3.13. To estimate the position, the proper placement of the person in the range finder Field Of View (FOV) needs to be determined. Therefore, the angle θ_{rf} must be identified, by first determining in which part of the image received from the camera, the centre of the largest detected blob is situated. Hereafter, from the fact that the camera FOV has been measured to be $25°$, θ_{rf} can be calculated from knowledge of the range finder FOV, resolution and sample count.

Having identified the correct laser sample to read and calculated the position of the person relative to the robot, the position must be rotated by θ and displaced by $\mathbf{p}_r$, in order to also be represented in world co-ordinates.

When the position of the detected person is known in both the robot frame (x_r,y_r) and the world frame (x_w,y_w), it still remains to estimate heading and translational velocities. As stated above these parameters are closely related, since the heading of the detected person is calculated from monitoring his/her movement.

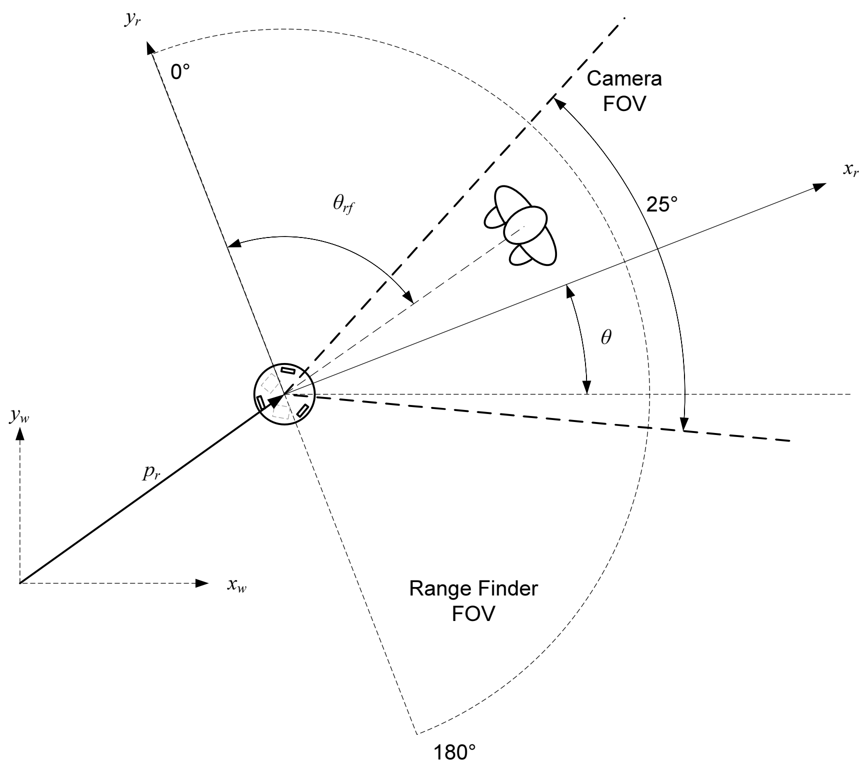


FIGURE 3.13: *The basic approach for the Person Detector algorithm. Note, that the camera FOV has been exaggerated for illustrative purposes.*

Thus, from calculating the translational velocities by differentiation of estimated positions, the heading of the person is estimated. Such estimation of the heading clearly requires the person to be moving, and furthermore requires decisioning on whether movement is actually occurring or simply caused by sensor noise. The latter has been accounted for by introducing movement thresholds. Furthermore, when calculating the person's velocity and heading in the robot frame, it must be taken into account, that the robot's motion also contributes to changes of the person's position in the robot frame (See Figure 3.14). This is handled, by calculating the change in robot position from received odometry information between the samples of the person position. The contribution from the robot velocity to the person velocity is then eliminated to yield the resulting person velocity relative to the current robot frame. The person's velocity relative to the latest robot frame is given by transforming the person's position in the prior robot frame into a position in the current robot frame and subtracting this from the current position. Using the notations of Figure 3.14 yields:

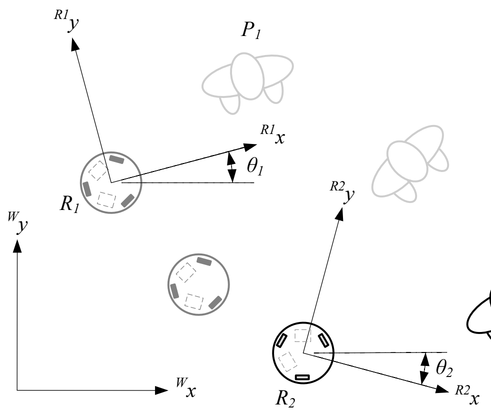


FIGURE 3.14: *When both the robot and the person moves, calculating the person's velocity in the robot frame, necessitates elimination of the contribution from the robot's velocity.*

$$^{R_2}v_p = ^{R_2}P_2 - ^{R_2}P_1 \tag{3.4}$$

$^{R_2}v_p$ is the person velocity relative to the robot frame R_2.
$^{R_2}P_2$ is the person position P_2 relative to robot frame R_2.
$^{R_2}P_1$ is the person position P_1 transformed into co-ordinates relative to robot frame R_2 using the following transformation:

$$^{R_2}P_1 = \text{Rot}(\theta_{\Delta R})\ (^{R_1}P_1 -^{R_1} R_2) \tag{3.5}$$

Rot$(\cdot)$ is the rotation given by $\begin{bmatrix} \cos(\cdot) & \sin(\cdot) \\ -\sin(\cdot) & \cos(\cdot) \end{bmatrix}$

$\theta_{\Delta R}$ is the angular difference between robot frames R_1 and R_2 in world co-ordinates ($\theta_{\Delta R} = \theta_2 - \theta_1$ on Figure 3.14).

$^{R_1}R_2$ is the co-ordinate of the origin of robot frame R_2 relative to robot frame R_1 and is calculated by using the following expression:

$$^{R_1}R_2 = \text{Rot}(^W\theta_{R1})\ [\ x_{\Delta R} \quad y_{\Delta R}\ ]^T \tag{3.6}$$

$^W\theta_{R1}$ is the angular displacement of the robot frame R_1 in world co-ordinates.
$x_{\Delta R}$ and $y_{\Delta R}$ are the differences in x and y co-ordinates between the robot frames R_1 and R_2 relative to the world frame. This difference is calculated based on odometry information received from the robot through its `position2d` interface.

Having calculated the person's velocity, the heading of the person relative to the current robot frame can now be determined. If the person has moved, the heading is calculated based on the velocity vector by using the arctan function. If the person has not moved, the heading is calculated by adding the angular difference $\theta_{\Delta R}$ between the prior and the current robot frame to the person's heading in the prior robot frame.

Detection limitations

Person Detection based on blobs may be challenging, due to changes in the light of a real-world test environment, blobs can indeed appear when no person is present. No measures have been incorporated to counter this phenomenon. A solution to the problem could be to compare range finder readings with the map of the given environment. Changes in light can also appear when a person is in sight of the robot. This has been countered by always making the Person Detector perceive the largest detected blob as a person. It should be emphasized, that this is merely a simple rectification, based on the fact that blobs caused by changes in light often appear as small fragments, unless the change is widespread. Again, more thorough remedial measures involve either map comparison or possibly dynamically changing detection thresholds based on the amount of light in the camera images received.

3.4.2 Person Evaluator

Provides
Parameters for the final case description
Parameters for the Behaviour Manager
Requires
Motion data of detected person

TABLE 3.2: *Interfaces of the Person Evaluator*

The Person Evaluator represents the robot's ability to judge the detected person's reaction to the robot's own motion pattern. Thus, the Person Evaluator enables the robot to actively participate in the observation of a detected person, and thereby strengthen the judgement of whether the person is interested in an encounter or not. The Person Evaluator is only intended to be active when the robot is in front of the person.

Based on studies of human behaviour [García-Rojas et al., 2006], the Person Evaluator algorithm is based on the fact that if a person A is interested in a close encounter with another person B, he would undoubtedly approach this person in a straightforward manner. On the contrary, if in no interest of closer contact, person A would carefully avoid the path of person B. Although the robot is not perceived as a human being when encountering people, it is assumed that the human behavioural reactions are the same, or at least very alike.

Since the net direction of the robot is presumed to always be targeted on the detected person while evaluating, the developed algorithm solely focuses on the motional changes of the detected person. The basic idea of the algorithm is illustrated in Figure 3.15.

As seen from the figure, the features chosen to characterize the behaviour of a person is the projected vector $\mathbf{v}_{pers,proj}$, and the area of the triangle spanned by the person's velocity vector $\mathbf{v}_{pers}$ and the vector between the robot's and the person's

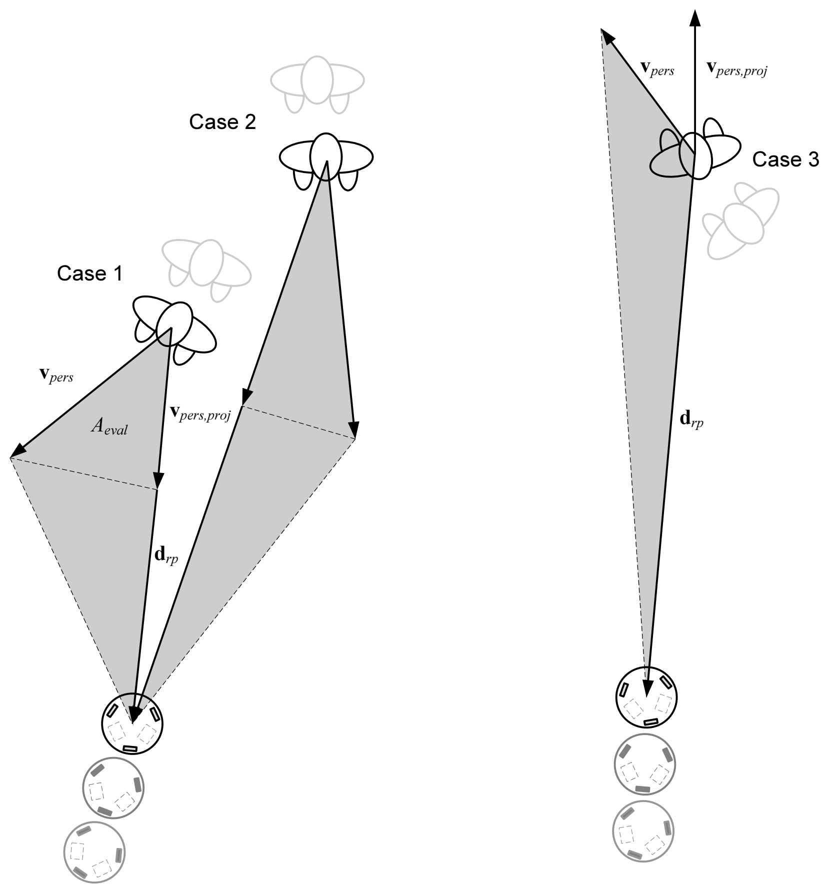


FIGURE 3.15: *The basic element of the Person Evaluator algorithm, being the calculation of the spanned area A_{eval} and the projected vector $\mathbf{v}_{pers,proj}$.*

current position $\mathbf{d}_{rp}$. Starting from the projected vector $\mathbf{v}_{pers,proj}$, this feature can tell whether the person is actually moving towards the robot or not, i.e. a negative projection relative to the person's heading would imply a velocity vector of the person $\mathbf{v}_{pers}$ directed away from the robot (see Case 3 on Figure 3.15). A person moving away from the robot, will undoubtedly suggest that the person is not interested in encountering the robot. The reason for calculating $\mathbf{v}_{pers,proj}$ instead of simply using the direction of $\mathbf{v}_{pers}$ is that further information on the person's movement is extracted. Thus, comparing $\mathbf{v}_{pers,proj}$ to the distance $|\mathbf{d}_{rp}|$ between the robot and the person, indicates the net velocity of the person towards the robot (see Case 1 and 3).

If a person is indeed moving towards the robot, the calculation of the area A_{eval} as represented in all three cases on Figure 3.15 becomes relevant. The area provides a good indication of how interested the person is in encountering the robot. The smaller the area, the more interested the person is. However, as illustrated by the three cases on Figure 3.15 the information provided by the size of the area spanned, must be interpreted by taking the distance $|\mathbf{d}_{dist}|$ into consideration. The more apart the robot and person is, the less value the area provides. Studying Case 2 in the figure, the person's reaction is less certain to be caused by the movement of the robot, and thus the large area spanned should somehow be weighted according to the distance between robot and person. To overcome this problem, measures have been taken in the CBR solution of the Trainee described in the following section.

3.4.3 Trainee

It has been chosen to implement the CBR solution using a *MySQL* database. Therefore, due to the Controller being developed in C++, the CBR implementation has been carried out using *MySQL*++, a C++ wrapper for *MySQL*'s C API. This wrapper is built upon Standard Library Template (STL) principles, meaning that the task of handling the database, is basically like dealing with STL containers as e.g. vectors and lists [Tangentsoft, 2007]. The following treats the choice of the actual case contents as well as the method by which encountered cases will be handled.

Defining a case

As stated in Section 2.3.5, the task of specifying a case is a question of determining a distinct and representative set of features connected to the event of a robot-human encounter. The more relevant features extracted, the more specific the robot experiencing can be made towards the various person encounters. Hence, the outcome of the Person Evaluator, being the spanned area between the velocity vector of the person and the vector between the person and the robot, is a natural choice for inclusion in the case description along with the distance at which the area is recorded. The chosen features are listed below:

Distance

Recording the distance to the person, allows the robot to relatively evaluate the related spanned area. As mentioned in Section 3.4.2, a large spanned area at a great distance, does not contain as much person behavioural information as one recorded close to the person.

Values: Distances with a precision of two decimals governing the Personal and Social Zones as designated by Hall.

Spanned area

As mentioned above, the spanned area is the calculated outcome of the Person Evaluator, containing information on how directly the person and robot are approaching each other.

Values: Calculated area spanned by person's velocity vector and the vector between person and robot.

Position

Having detected a person, the robot must estimate his/her position in the environment. This information is recorded in order for the robot to learn if people exhibiting the same kind of behaviour, are most likely to be encountered in certain areas.

Values: The position of the person is represented by an x and y position in meters, with a precision of one decimal.

Time of day

By recording the time of day upon detecting a person, the robot can gradually become aware of possible similarities between the solution outcome i.e. whether assistance is needed or not and the associated time of day.

Values: The time of day is represented by an unsigned integer value. A total of six periods have been selected as representative for a normal work day:

- 0: Morning rush hour (06:00-09:00)
- 1: Late morning (09:00-11:00)
- 2: Lunch (11:00-13:00)
- 3: Afternoon (13:00-15:00)
- 4: Work day end (15:00-17:00)
- 5: Evening (17:00-23:00)

Type

In an ideal setting, the robot should be able recognize every person it has previously tracked in a given environment and identify certain characteristics about them. Such knowledge would enable the robot to e.g. see if a person it has previously judged as exposing an uninterested behaviour and therefore chosen to ignore, is still around. Furthermore, being able to e.g. distinguish between children and adults, would provide the robot the ability to e.g. always encounter children no matter their exhibited behaviour.

Values: Although this feature has been included in the software design, it has not been implemented in the final solution tested in Chapter 4.

From the above description of the features to include in the case representation, the following section treats the actual implementation of the Trainee sequence of events constituting the CBR solution.

Looking up a case

Due to the active participation of the robot in evaluating a detected person, the method for looking up cases should somehow be able to govern the dynamics of the Person Evaluator outcome over a period of time. Thus, limiting the case lookup to a single case during encountering would be an inadequate solution. The method developed to overcome this problem is illustrated in Figure 3.16.

As seen from the figure, the robot begins to perform temporary case lookups when it has reached a distance to the person of $3.6\,\mathrm{m}$, equivalent to the intersection between the Personal and Social Hall zones. Hereafter, case lookups will be

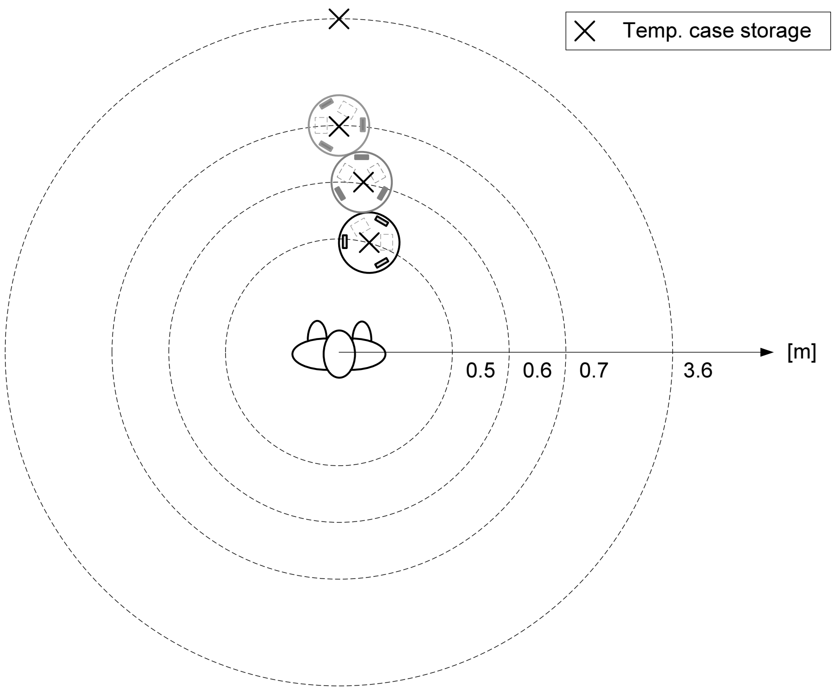


FIGURE 3.16: *The developed method for looking up cases during an encounter. All temporary cases will be stored, until an outcome of the encounter is known. At this point, the cases will be altered accordingly, stored in the main case database, and finally the temporary case database is emptied.*

performed according to a quantization of the distance into steps of $10\,\mathrm{cm}$. This limit in precision has been introduced to set a boundary for the inevitable contradiction of whether the robot should create new cases, or make choices from past experiences. Moreover, it is deemed reasonable not to judge the behaviour of a person too frequently, since behavioural reactions would be more difficult to detect.

Since multiple lookups are required during person evaluation, two distinct databases are used. One serving as the main case library, the other functioning as storage when performing temporary case lookups. The two databases are described more thoroughly in the following:

Main database
Stores all experiences of the robot and thus functions as the main lookup database. The database table contains the columns as described in Figure 3.17.

case_id	dist	area	pos_x	pos_y	time_of_day	type	indication

FIGURE 3.17: *The columns contained in the main database table.*

Compared to the case features outlined in Section 3.4.2, the fields `case_id` and `indication` are new. The `case_id` is merely a result of the *MySQL* implementation, being the primary and auto-incremental key of the main database. The `indication` field is introduced in order to store the probability or indication of the detected person's interest to interact.

Temporary database
Used during person evaluation. Cases recorded during encountering will be temporarily stored in this database, for later to be evaluated and transferred to the main database. These actions call for new fields to be introduced in the temporary database table as illustrated in Figure 3.18.

case_id	dist	area	pos_x	pos_y	time_of_day	type	indication	edit	stored_id

FIGURE 3.18: *The columns contained in the temporary database table. The highlighted fields are those different from the main database.*

These new fields are used when a case match is found in the main database. In these situations, the case will be copied to the temporary database and marked with the value 1 in the `edit` field indicating, that it should be updated and not created during case revision. The `stored_id` field is used to store the `case_id` value of the matching case in the main database in order to be able to locate the case again, when the outcome is known.

Thus, a typical course of events is, that the robot detects a person, and (while approaching and evaluating the person), stores a case for every ten centimetres difference in travelled distance between robot and person. Whenever an outcome of the encounter is known, the temporary cases must be evaluated and afterwards erased. Thus, the derived method can be divided into two being retrieval/reuse and revision/creation to be described in the following.

Case retrieval and reuse When looking up a case in the main database, two possible scenarios can occur:

- No match: The currently faced case is stored directly into the temporary case database. The value of field `indication` is set to the default value of 0.5, indicating that the robot should, as default, perceive the person as being interested in an encounter.
- Match: The existing case is copied to the temporary case database, for later alteration of its indication when an outcome is known, i.e. during case revision. The temporary case database fields of `edit` and `stored_id` are used when a matching case has been found.

When searching for cases in the main database, rules must be set up in order for the robot to be able to identify certain parameters from others. Comparing cases by simply taking the average of all parameters would undoubtedly introduce errors in the case retrieval, since some cases would have more influence than others due to e.g. differing units or person behavioural judgement value.

The case retrieval has been implemented by exploiting the query features of *MySQL*. Studying the code excerpt of Code 3.2, line 2-4 indicates a database lookup from the main case library `stored_cases`. Note, that the **`SELECT`** command selects the `case_id` along with a customized field `area_diff` calculated as the difference between the `area` of the given stored case and that of the temporary case currently revised. This selection of data is followed by the *MySQL* syntax of **`WHERE`** , indicating that certain restraints governs the data lookup. As seen from lines 5-9, a case match will only be made if an existing case holds the same data as the temporary case in the fields: `distance`,`pos_x`, `pos_y`,`time_of_day` and finally `type`. If one of these fields contains different data, the temporary case will instead be treated as a new case and stored in the main database accordingly. Turning to the final *MySQL* instruction of line 10, all cases found to match the current temporary case will be returned in ascending order, meaning that the case with the largest area difference i.e. `area_diff` will be first. Hence, if the difference exceeds a specified threshold value, the temporary case will again be stored as a new case, while a smaller difference will result in a case revision of the stored case.

```
query <<
   "SELECT case_id,"<<
   "ABS((area-'"<<ptrCase->get_floatSpannedArea()<<"')) AS area_diff"<<
   "FROM stored_cases"<<
   "WHERE distance='"<<ptrCase->get_floatDistance()<<"'"<<
   "AND pos_x='"<<ptrCase->ptrPositionData.get_floatPoseX()<<"'"<<
   "AND pos_y='"<<ptrCase->ptrPositionData.get_floatPoseY()<<"'"<<
   "AND time_of_day='"<<ptrCase->get_intTimeOfDay()<<"'"<<
   "AND type='"<<ptrCase->get_intType()<<"'"<<
   "ORDER BY area_diff ASC"<< endl;
res = query.store();
```

CODE 3.2: *MySQL++ code excerpt outlining the method used for retrieving cases from the main database during person evaluation.*

Case revision and creation Imagine that the robot has completed a person evaluation, and that the temporary case database, as a result of the performed case lookups, holds a given amount of cases. Whether the person evaluation has ended because of the person being evaluated as not interested or as a result of conducted communication, the robot should now revise all of the temporary stored cases. Thus, some cases should be created in `stored_cases`, while others should be used in updating existing cases. Either way, the temporary case database field of `indication` is updated during revision according to the experienced outcome. As indicated in Section 3.4.2, this alteration should be considered in relation to the distance to the person associated with the case. The further away, the less the indication value due to the person's reaction towards the robot naturally being strengthened the closer he/she is to robot. Such weighted alteration has been implemented utilizing the behavioural zones as designated by Hall. Two weight functions have been derived and illustrated in Figure 3.19.

As seen from the figure, the weight on the person indication will increase as the distance between robot and person decreases. Furthermore, having entered the Personal Zone of the detected person, the weight function shifts resulting in a radical increase in weight according to distance.

When updating the indication value during revision, a variable `intLearningRate` has been implemented to allow for adjustment of the rate

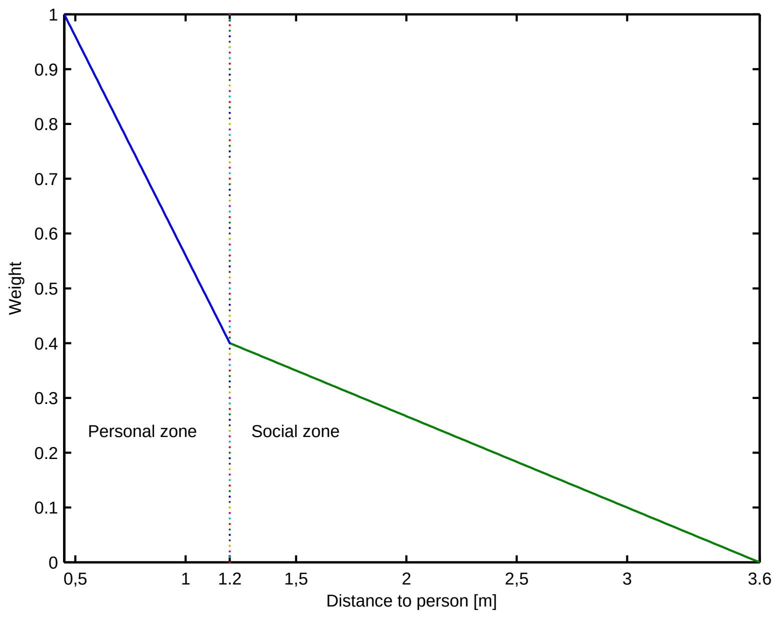


FIGURE 3.19: *The weight function governing the Personal and Social Zones respectively, introduced, in order for the robot to pay more attention to the reactions of the detected person, the closer he/she is to the robot. Note that the limits on the distance-axis reflects the social distances according to Hall. Thus the minimum distance signifies the transition from the Personal Zone to the Intimate Zone, never to be entered by the robot.*

by which the robot learns from the experienced outcomes. Thus, the lower the learning rate, the less effect the weighing will have.

Having described the implementation of the Trainee, the following section treats the development of the Controller client, tying all developed modules together.

3.5 Controller

3.5.1 Controller development

As described in Section 2.4, the objectives of the Controller are to:

- Embody a *Player* client,
- Act as a state supervisor, e.g. the Person Evaluator should only be active at certain times in the operation cycle of the robot.
- Incorporate the functionalities of the Person Evaluator, Trainee and Communicator.

Furthermore, it has been decided to include a GUI in the Controller design. Targeted on the phase of implementation and test, the Graphical User Interface (GUI) allows for parameter and robot state surveillance. Since the introduction of the GUI in the Controller implementation is caused by a desire of monitoring certain variables of the system during debugging and testing, no further details will be included on this part of the Controller solution.

In the following, some parts of the Controller design will be treated more briefly than others, e.g. not all class procedures and attributes will be described in the text. If more information is needed than those included, refer to Annex 2 where the full documentation of the Controller is available in HTML format.

3.5.2 *Player* client

The `libplayerc++` client library functions as a so-called "service proxy" model [PlayerProject, 2006]. Hence, the client is implemented by including objects functioning as proxies for remote services, i.e. the services provided by the initiated *Player* server(s). Thus, developing a client for *Player* is merely a question of deciding which proxies to use and to develop associated callback functions.

The proxies needed maintained by the Controller, in order to provide and receive the necessary data are listed in Table 3.3. Furthermore, the functionality subscribed for, and the nature of the conducted communication is included alongside the proxies.

Proxy	**Connected functionality**	**Communication**
`LocalizeProxy`	Localizer	Receive
`PlannerProxy`	Navigator	Provide/Receive
`Position2dProxy`	Person Detector (robot frame)	Receive
`Position2dProxy`	Person Detector (world frame)	Receive
`OpaqueProxy`	Behaviour Manager	Provide
`OpaqueProxy`	Velocity Manager	Provide
`IrProxy`	Communicator	Receive

TABLE 3.3: *Proxies maintained by the class `RobotClient` in order to communicate with active Player servers.*

By maintaining these proxies the Controller is able to carry out necessary routines using the callback procedures as listed in the class `RobotClient` in Figure 3.20.

The above listed procedures are briefly described below.

callbackLocalizer

Receives information on the current `amcl` driver status, i.e. the estimated position and the likelihood of its certainty through the `LocalizerProxy`.

callbackNavigator

Designates navigational targets through the `PlannerProxy` if active, and receives feedback on whether an appointed target is valid, or if the robot has reached it.

callbackPersonDetector/callbackPersonDetectorWorld

Receives information through the two `Position2dProxy` interfaces, on whether any person is detected, and if so receives the position of the detected person both in the robot frame and world frame.

callbackCommunicatorIR

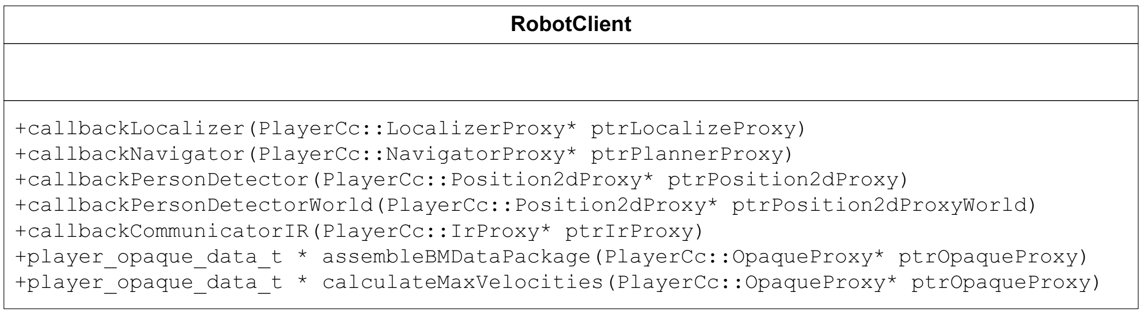
Receives input from the Robotino® IR sensors during person interaction, in order to determine the answer provided by the person. Note, that since this feature is essentially the task of the Communicator, this particular functionality will not be treated further.

A `libplayerc++` client is started by creating its own individual thread. Thus, when a message is received from the server, the client fires the associated callback procedure handling the received message. Consequently, communication with underlying *Player* server(s) will constantly be maintained, regardless of the main Controller thread's current state of execution.

Having described the class `RobotClient`, the following section threats the state supervisory role of the Controller.

3.5.3 State supervision

In Section 2.4.1, the states and conditions for state transitions were determined. Since all of the conditions depend on both information received from the robot, and on the implemented functionalities in the Controller, it has been decided to include the state supervisory control in the main class `Controller`, while stor-



```
RobotClient

+callbackLocalizer(PlayerCc::LocalizerProxy* ptrLocalizeProxy)
+callbackNavigator(PlayerCc::NavigatorProxy* ptrPlannerProxy)
+callbackPersonDetector(PlayerCc::Position2dProxy* ptrPosition2dProxy)
+callbackPersonDetectorWorld(PlayerCc::Position2dProxy* ptrPosition2dProxyWorld)
+callbackCommunicatorIR(PlayerCc::IrProxy* ptrIrProxy)
+player_opaque_data_t * assembleBMDataPackage(PlayerCc::OpaqueProxy* ptrOpaqueProxy)
+player_opaque_data_t * calculateMaxVelocities(PlayerCc::OpaqueProxy* ptrOpaqueProxy)
```

FIGURE 3.20: *The `RobotClient` class of the Controller. Note that constructor/destructor procedures have been omitted.*

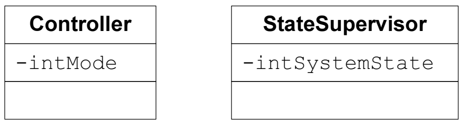


FIGURE 3.21: *The main class `Controller` covering the supervisory control and the class `StateSupervisor` holding the current state in `intSystemState`. The `intMode` attribute provided to the `Controller` represents user input, on whether the Controller should be executed in real-world test mode or in simulation mode. Note that standard procedures, constructor/destructor and attribute get/set procedures, have been omitted.*

ing the current state in the class `StateSupervisor` attribute `intSystemState` (see Figure 3.21).

As seen from the figure, the class `Controller` only includes one attribute being the `intMode`. This attribute represents user input from the GUI, determining whether the Controller should be executed in real-world test mode or in simulation mode. Such selection of execution mode have been included in order to allow for more easy and rapid simulation, by generating person answers instead of requiring the simulation supervisor to somehow affect the robots IR sensors as required in a real-world scenario (see Section 2.3.6). Therefore, three simulation modes have been established, covering the person behaviours of being interested, not interested and randomly interested.

Since the real-world Communicator functionality is actually handled through the `IrProxy` of the class `RobotClient`, the Communicator functionality during simulation has been included in class `Communicator` through the `interactWithPerson()` procedure. The `Communicator` class is depicted in Figure 3.22.

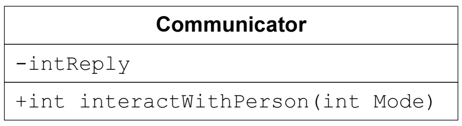


FIGURE 3.22: *The class* `Communicator` *which is only applicable, when the Controller is executed in simulation mode due to the real-world communication being handled in the class* `RobotClient`*. Thus, generated replies are stored in the* `intReply` *attribute. Notice, that standard constructor/destructor procedures have been omitted in the class description.*

Having described the main class `Controller` with focus on its primary functionalities, the following section outlines the entire Controller structure.

3.5.4 Controller structure

All the above described parts of the Controller, along with those not treated in detail are gathered in the overview class diagram of Figure 3.23.

Starting from an overall system point of view, the Controller has been divided into two separate threads, implemented by utilizing the *Boost* C++ libraries [Boost, 2007]. These two threads cover the main Controller application and the GUI (class `GraphicalUserInterface`) respectively. Consequently, the `GUIData` class has been included to act as a data container allowing data exchange between the two threads.

The fact that the Controller is acting as an interconnecting unit for local and underlying drivers, requires storage of information concerning the robot and the detected person. Studying the class diagram of Figure 3.23, two classes, namely the `Person` and `Robot` have been introduced, acting as primary data containers for all such information. Starting from the class attributes, the classes are treated in the following:

Person

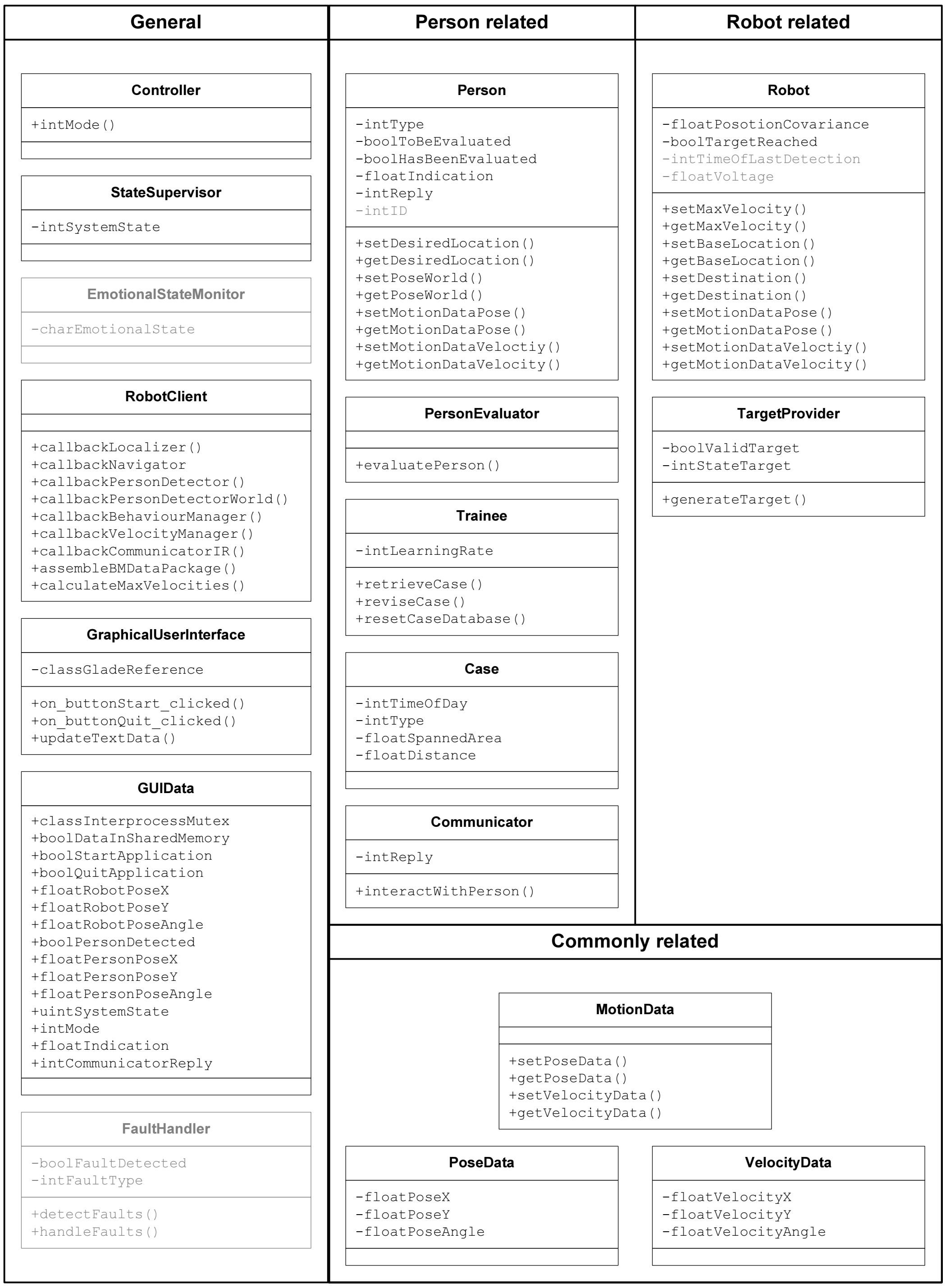


FIGURE 3.23: *Complete class diagram of the developed Controller. The General column contain system-wide related classes, while the columns of the Robot and Person contains similar related classes. The Commonly related section contains blueprint classes, used in all depicted get/set procedures. Greyed out elements have been part of the design process, but not included in the current implementation.*

`boolToBeEvaluated`

Having detected a person, the Behaviour Manager is invoked at once, while the Person Evaluator should only be invoked when certain, that the person is heading towards the robot. When this has been verified from the Person Detector, `boolToBeEvaluated` is set to true allowing the Person Evaluator to be invoked.

`boolHasBeenEvaluated`

The `boolHasBeenEvaluated` is set to true, the moment the Person Evaluator is initiated, and set to false whenever the person is out of sight. Combining the logical values of `boolToBeEvaluated` and `boolHasBeenEvaluated`, the Behaviour Manager should only be disabled whenever the following expression is true:

```
!boolToBeEvaluated && boolHasBeenEvaluated
```

This way, the Behaviour Manager will be disabled if the Person Evaluator has judged the person to be uninterested, although he/she is still detectable by the camera on the robot.

`floatIndication`

Holds the value of person interest indication obtained through the continuous case database lookups during person evaluation. As described in Section 3.3.1, the `floatIndication` is used in the Behaviour Manager to adjust the motion of the robot.

`intReply`

Upon initiating communication with a detected person, the reply, whether generated in simulation or received through the `RobotClient ::callbackCommunicatorIR`, is stored in the `intReply` attribute.

`intID`

Included in the design progress to function as an identification label, in case the robot is to detect multiple persons in an environment. This attribute has not been implemented.

`intType`

Intended to make the robot capable of distinguishing between certain types of person. This would e.g. enable the robot to distinguish the cleaning lady from a visitor, allowing for more seamless robot integration in a given environment. This attribute has not been implemented.

Robot

`floatPositionCovariance`

Whenever the `RobotClient::callbackLocalizer()` is invoked, the received data is processed in order to isolate the best pose estimate, as well as its associated confidence level. The latter is stored in the `floatPositionCovariance`.

`boolTargetReached`

If a target has been set for the robot to reach, the `boolTargetReached` attribute is set to true the moment the robot reaches its destination.

`intTimeOfLastDetection`

Having considered the parameters affecting the mood and behaviour of the robot in the design process, the `intTimeOfLastDetection` is included as an indicator of the robot's level of frustration. By example, the longer the time passed and not meeting any people, the more frustrated the robot should get. This attribute has not been implemented.

`floatVoltage`

This attribute was included in the design process to enable the Controller to determine when the robot should return to base. The *RobotinoCom* API provides functions for receiving the current battery status. This attribute has not been implemented.

Having described the main storage classes of the Controller, the following treats the general functioning of the Controller as seen from the classes `RobotClient`, `Controller` and `Person Evaluator`. These three classes represent the main functionalities of the Controller being communication, overall system control, and human-robot interaction. The description aims at explaining the most important attributes and procedures included in the class overview diagram of Figure 3.23.

`RobotClient`

As mentioned earlier, the `RobotClient` handles all communication with active *Player* server(s). Much of the information received is stored in the storage classes `Robot` and `Person` described above. Apart from these, the `RobotClient` updates the position and velocity of the person and robot through the procedures `Robot::setMotionDataPosition()` and `Robot::setMotionDataVelocity()`. Furthermore, the person's position in the world frame is set by `Person::setPositionWorld()`, while the maximum allowed robot velocity allowed is updated by use of the `Robot::setMaxVelocity()`.

`Controller`

When invoking the Controller application, the robot starts to roam the environment looking for persons to evaluate. The roaming behaviour is obtained through the class `TargetProvider`, offering the `generateTarget()` procedure which calculates targets for the robot to follow. All robot targets are stored using the `Robot::setDestination()`. The class `TargetProvider::generateTarget()` is also invoked when the robot should guide the person or return to base. The person's goal is updated by `Person::setDesiredLocation()`, while the base location is altered using the `Robot::setBaseLocation()` procedure.

Mentioned in Section 3.5.3, the `Controller` functions as the nodal centre of the Controller. It controls the state of the system, and stores the current state in `StateSupervisor::intSystemState()`. Furthermore, though not implemented, the `StateSupervisor` is to provide the current state to the `EmtionalStateMonitor`, designed to control the mood/behaviour of the robot.

`PersonEvaluator`

Upon detection of a person heading in the direction of the robot, the `PersonEvaluator::evaluatePerson()` procedure is invoked. The `PersonEvaluator` is furthermore associated with the following classes:

- `CaseDatabase` - containing the CBR implementation
- `Case` - the blueprint for newly generated cases
- `Communicator` - generating replies when the Controller is executed in simulation mode.

During person evaluation, a case object of class `Case` is constantly updated with the most recent values, and used in the continuous case database lookups performed using the `CaseDatabase::retrieveCases()` procedure. Whenever the Person Evaluator reasons that the person is not interested, e.g. if moving out of sight or by communicating, `PersonEvaluator::reviseCases()` is fired. As described in Section 2.3.5, this procedure performs a clean-up of the temporary case database, updating the `indication` fields according to the experienced outcome. Furthermore, useful cases are stored in the main case database.

The above description of the Controller concludes the Design and Implementation chapter. The following chapter treats the conducted system experiments and comments on the obtained results.

CHAPTER 4

Experiments and Results

Having developed a system solution providing the Robotino® with the functionalities described in Section 1.2, this chapter treats the experiments conducted in order to validate the overall functionality. Starting from the overall objective of the developed system, the robot should be able to:

- Localize itself an a given environment.
- Roam the environment while keeping track of its own location.
- Approach and interact with a detected person taking into account the effect of robot behaviour on the person.
- Guide the person to a desired location.

Some of the listed test objectives are a necessity for others to function, and will thus not be tested separately. When applicable, comments will be made on which objectives are implicitly tested. Furthermore, it seems natural to divide the test into the above three categories, i.e. before, during and after encountering a person. However, before treating the individual experiments, the data acquisition, which is common for all tests, will be described.

4.1 Data acquisition

Generally, the tests will be conducted in both a real-world and in a simulation environment.

When a test is conducted by simulation, the resulting data is easily acquired through software by writing relevant data to log files. In contrast, testing in a real-world environment requires certain precautions to be taken, in order to be able to monitor the planned test. However, some internal robot data which can not be observed from the outside is still saved in log files. The proposed solution for acquiring the external data is to survey the test environment using a camera. Figure 4.1 presents a still picture captured from the installed surveillance camera. Applying easily distinguishable markings to test subjects and objects, provides for position data to be extracted from camera recordings. The procedure for such

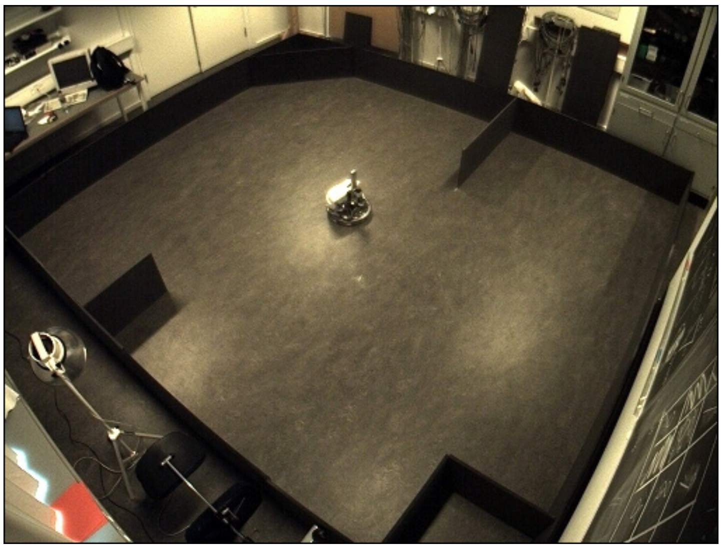


FIGURE 4.1: *Still picture captured from the camera surveiling the real-world test environment. Using a wide-angle* 4 mm *lens ensures full test space coverage.*

extraction, has been to initially calibrate the camera using the MATLAB© targeted Camera Calibration Toolbox[1], which outputs both intrinsic and extrinsic camera parameters. These parameters are subsequently used to correct internal camera characteristics (intrinsic) and to transform (extrinsic) the coordinates of the markings in the camera frame to world frame coordinates. Data acquired from calibration along with sample images are included in Annex 5.

4.1.1 Person test subject

Performing the "during encounter" tests, introduces the need for placing a person in the test environment. In order to conduct test in a simulated environment, a virtual test subject has been created for *Stage*. The test subject is basically implemented as a robot, where certain characteristics have been changed in order to simulate the nature of a human being. Furthermore, the virtual person test subject can be provided both path planning and obstacle avoiding capabilities by using applying the `wavefront` and `vfh` *Player* drivers, respectively. This becomes useful when simulating certain human behaviour.

Having described how data is acquired, the following sections concern the

[1]`http://www.vision.caltech.edu/bouguetj/calib_doc/`

individual tests.

4.2 Before encounter

When the system is initialized, the robot must first and foremost be able to localize itself in the given environment. Thus, this test is targeted on the Localizer functionality of the robot, and is carried out to verify the essential localization, and furthermore to identify possible performance issues. Furthermore, the conducted experiment indirectly verifies the functioning of the developed Robotino® and range finder *Player* drivers.

In order to test the Localizer functionality a test scenario have been set up, in which the robot's Localizer is provided with an initial pose of $(1.5, 1.5, 0)$, while the robot actually starts in $(0, 0, 0)$. When set to roam the environment, the robot must discover that it is currently not situated at the right location, and afterwards find its true location.

The following performance related parameters are registered during the test:

- Precision of the performed localization.
- Time spent when performing the localization.
- The robot's perception of the precision of the localization.

During simulation, the precision of the performed localization is monitored by logging the true pose of the robot along with the estimated pose obtained from the Localizer. Furthermore, the sample times are logged in order to keep track of the overall simulation time. The `amcl` implementation of the Localizer provides means for monitoring the diagonal elements of the covariance matrix associated with the most likely pose of the robot. Thus, during testing, the mean of the pooled position variances will be logged in order to evaluate the robot's perception of its pose estimate, while the actual position of the robot is acquired from camera recordings in real-world experiments.

4.2.1 Results from simulation

The actual initial pose of the robot is $(1.5, 1.5, 0)$, while the robot is told that it is initially placed in $(0, 0, 0)$.

The actual path of the robot along with the estimated positions of the Localizer are plotted in Figure 4.2. Thus, it is seen, that the difference between the actual path and the estimated poses decreases as the robot moves and settles after about one minute of driving.

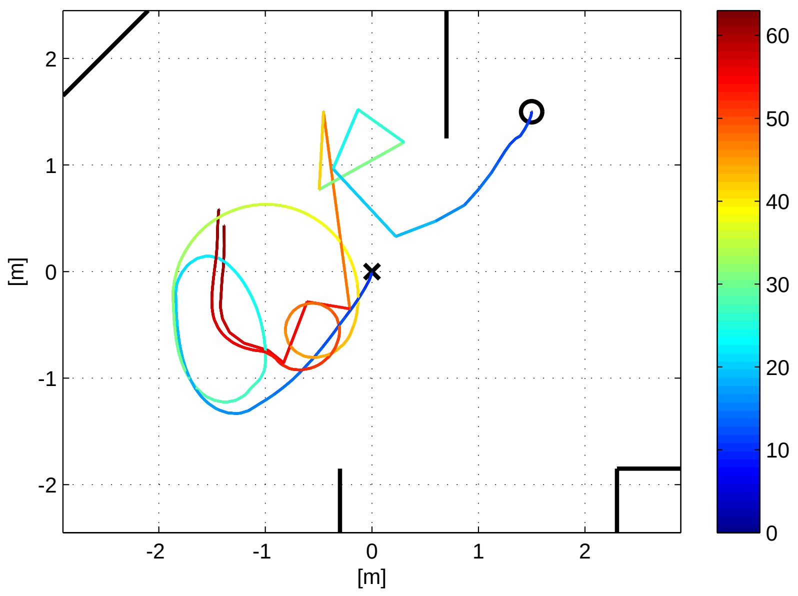


FIGURE 4.2: *Plot of the actual robot path (starting in* $(0, 0, 0)$ *marked with* × *) along with the estimated position of the Localizer (initial pose of* $(1.5, 1.5, 0)$ *marked with* ○ *). The gradient colour denotes the amount of seconds passed during robot operation.*

To verify, that the Localizer is indeed sure of its final position, Figure 4.3 illustrates the evolution of the mean position variance during simulation. Overall, it is seen, that the variance decreases at the end indicating a high degree of Localizer certainty on its final position estimate.

Comparing the mean variance plot with Figure 4.2 above, it is seen that during the first 20 seconds (the blue colour range) the Localizer is certain of its pose estimate. Comparing the actual and estimated robot paths, such belief seems reasonable since only a limited amount of environmental features tells the Localizer otherwise. As described in Section 2.3.1, the `amcl` driver implementing the Localizer, only uses a limited set of the entire range of laser readings available. In this experiment, the default value of 6 readings was used. Therefore, the degree of environmental features must be relatively high, in order for the Localizer to

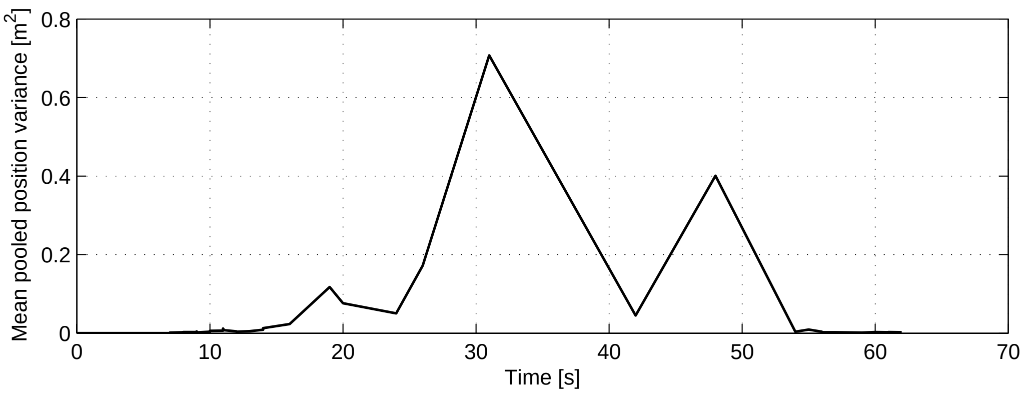


FIGURE 4.3: *The mean of the (x, y) position variance of the most likely Localizer pose estimate. Thus, a small variance indicates a high degree of precision, while higher values indicates Localizer uncertainty. Compare with Figure 4.2, to identify the environment locations causing trouble for the Localizer, i.e. at $t \approx 20$.*

detect them. This fact explains the sudden increase in the mean Localizer variance values after about 20 seconds. At this point all laser readings are affected by the bottom left corner, but the Localizer's pose estimate should not be obstructed by anything. This results in an enlargement of the Localizer's particle cloud to embody more possible poses, in order to find a better pose estimate than the one currently provided to the robot. The exact same explanation, applies for the mean variance decrease after about 42 seconds (corresponding to the light orange colour in Figure 4.2). However, in this case the difference between the actual and estimated poses is not as clear as in the case described above.

In conclusion, the fact that the difference between the actual position and the estimated one decreases with time, and moreover that the degree of certainty increases, indicates that the Localizer functions as intended. In regards to the performance issues of precision and time spent, the above simulation, with a localization time of about 60 s identifies no signs of unacceptable behaviour. However, since the `amcl` driver offers a variety of tuning parameters (odometry precision, range finder sensor model etc.), it seems unreasonable to judge these performance issues solely on the basis of simulation.

The following section treats the real-world experiments of the robot's localizing functionality.

4.2.2 Results from real-world test

The resulting plots of the robot's position during the real-world experiment, along with the estimated position of the Localizer is presented in Figure 4.4.

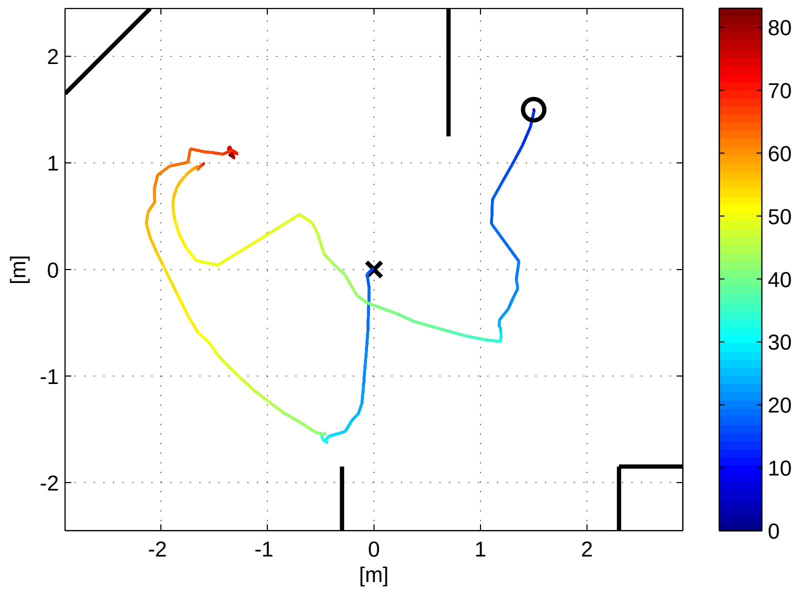


FIGURE 4.4: *Plot of the actual robot path (starting in* (0.0, 0.0, 0.0) *marked with* × *) along with the estimated position of the Localizer (initial pose of* (1.5, 1.5, 0.0) *marked with* ○ *). The gradient colour denotes the amount of seconds passed during robot operation.*

Comparing these plots with the pooled variances of Figure 4.5, the exact same explanations applies as for the simulated case.

In conclusion, the Localizer is seen to enable the robot to perform self-localization whenever the robot's perception of its surroundings disagrees with its stored map of the environment. Regarding the performance of the Localizer, the real-world test shows no deteriorating signs, and thus no need for further optimization is deemed necessary.

4.3 During encounter

During encounter, the robot must be able to judge the reactions from a person according to the robot's own movement, in order to determine whether the person is interested in a close encounter or not.

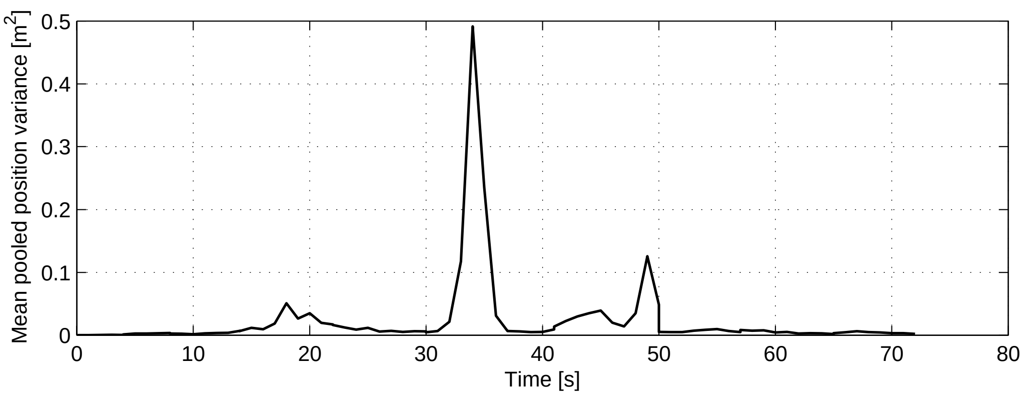


FIGURE 4.5: *Plot of the mean pooled position variance during real-world testing. The lower the variance, the more certain is the Localizer of its current pose estimate.*

The test scenario is divided into two test issues, being **human-aware navigation** and **learning**, which will be treated separately in the following.

4.3.1 Human-aware navigation

The human-aware navigation of the robot is tested in both a simulated and in a real-world setting. Common for both environments, the following scenarios must be tested:

- Encountered person is interested
  - Approach from the front
  - Approach from behind
- Encountered person is not interested
  - Approach from the front
  - Approach from behind

In order to isolate the verification of the robot navigation from influences of learning capabilities, the person's indication of interest is held at fixed values during testing. Thus, in the "interested" case, the indication value is fixed at 1 corresponding to a distribution width of 0.01 and a rotation of $45°$, whereas the indication value of the “not interested” case is fixed at 0.4 corresponding to a distribution width of 0.25 and a rotation of 0° as described in Section 3.3.1.

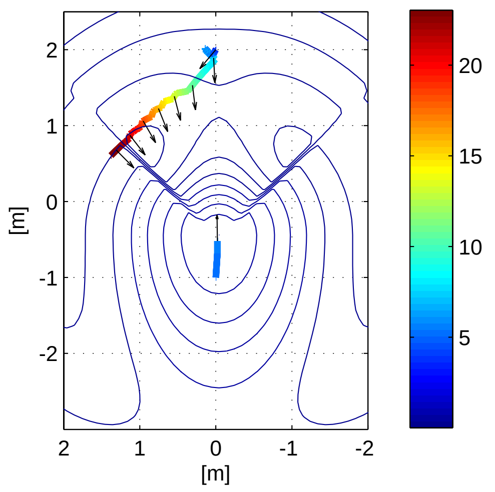
In all of the above scenarios, the robot must be able to locate the given test person by itself, thus testing the functionality of the Person Detector. Furthermore, the person test subject must exhibit a motional behaviour matching the actual case tested for, in order for the Person Evaluator to determine the heading of the person as described in Section 3.4.1. During testing, the motion of the robot and the person is recorded in order to verify if the behaviour of the robot is as intended.

Results from simulation

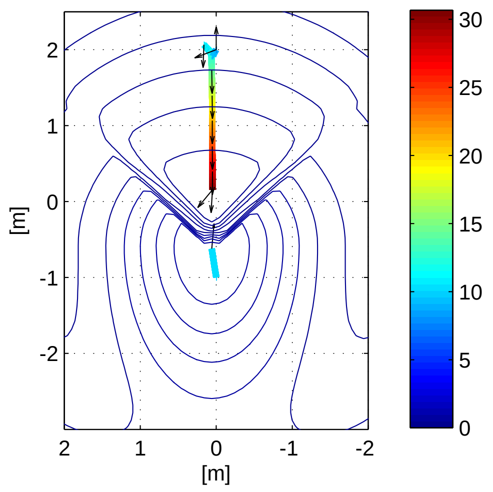
Results from simulation are presented in Figure 4.6 along with contour plots of the resulting Behaviour Manager distributions as they are designed to appear at the end of each test.

In Figures 4.6(a) and 4.6(b), the person initially moves towards the robot, making the robot aware that the person faces the robot, resulting in an approach from the front. In Figure 4.6(a), the person is “not interested” which results in the robot's final position being 45° relative to the person. As seen, by the underlying contour plot of the distributions, this final position is as expected, due to the forward distributions (see Section 3.3.1) being rotated 45°. Furthermore, the robot keeps a distance of approximately $1.5\,\mathrm{m}$ to the person, governed by the local minima of the resulting distribution. In contrast, the case depicted in 4.6(b), in which the person is “interested”, the robot approaches the person directly, resulting in a final position in front of the person at a distance of approximately $0.7\,\mathrm{m}$. Again, the path and final position of the robot are consistent with the rotation and widths of the forward distributions.

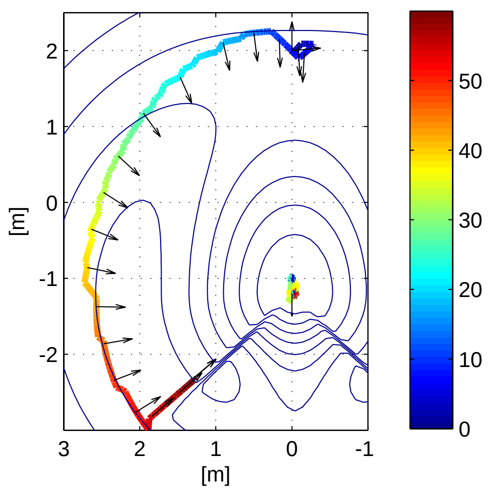
Figures 4.6(c) and 4.6(d), represent the cases in which the person does not move. Thus, the robot assumes that detected persons are heading the opposite way of itself (see Section 2.3.4), resulting in an approach from the back. Hence, in both the “not interested” and “interested” cases, the robot keeps a large distance to the person, when it is both behind and on the side of the person. Upon reaching a direction of approximately 45° relative to the person, the robot approaches the person. In the “not interested” case of Figure 4.6(c), the robot stops approx-



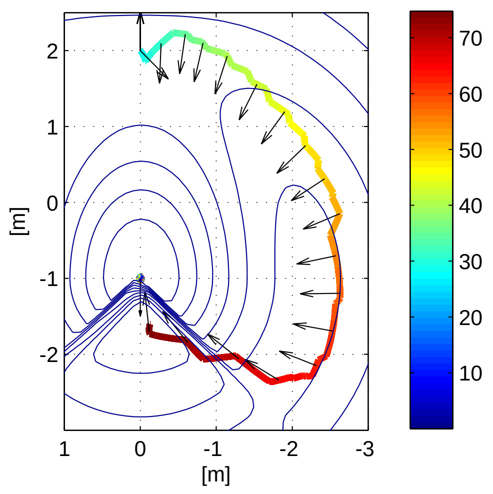
(a) Front (person not interested).



(b) Front (person interested).



(c) Back (person not interested).



(d) Back (person interested).

FIGURE 4.6: *Results from the simulated human-aware navigation test. Initial positions of the person and robot is* $(-1, 0)$ *and* $(2, 0)$*, respectively. The blue lines represent the contours of the theoretically calculated behaviour grid. The path of the robot is depicted by the multi-coloured line of which the colour gradient represents the elapsed time in relation with the colour bar to the right of each plot. Arrows indicate the current heading of the robot and the person.*

imately $1.5\,\mathrm{m}$ from the person, whereas in the "interested" case of Figure 4.6(d), the robot approaches the person further, ending approximately $0.7\,\mathrm{m}$ in front of the person. In both cases, the path and final position of the robot agree with the rotation and widths of the distributions.

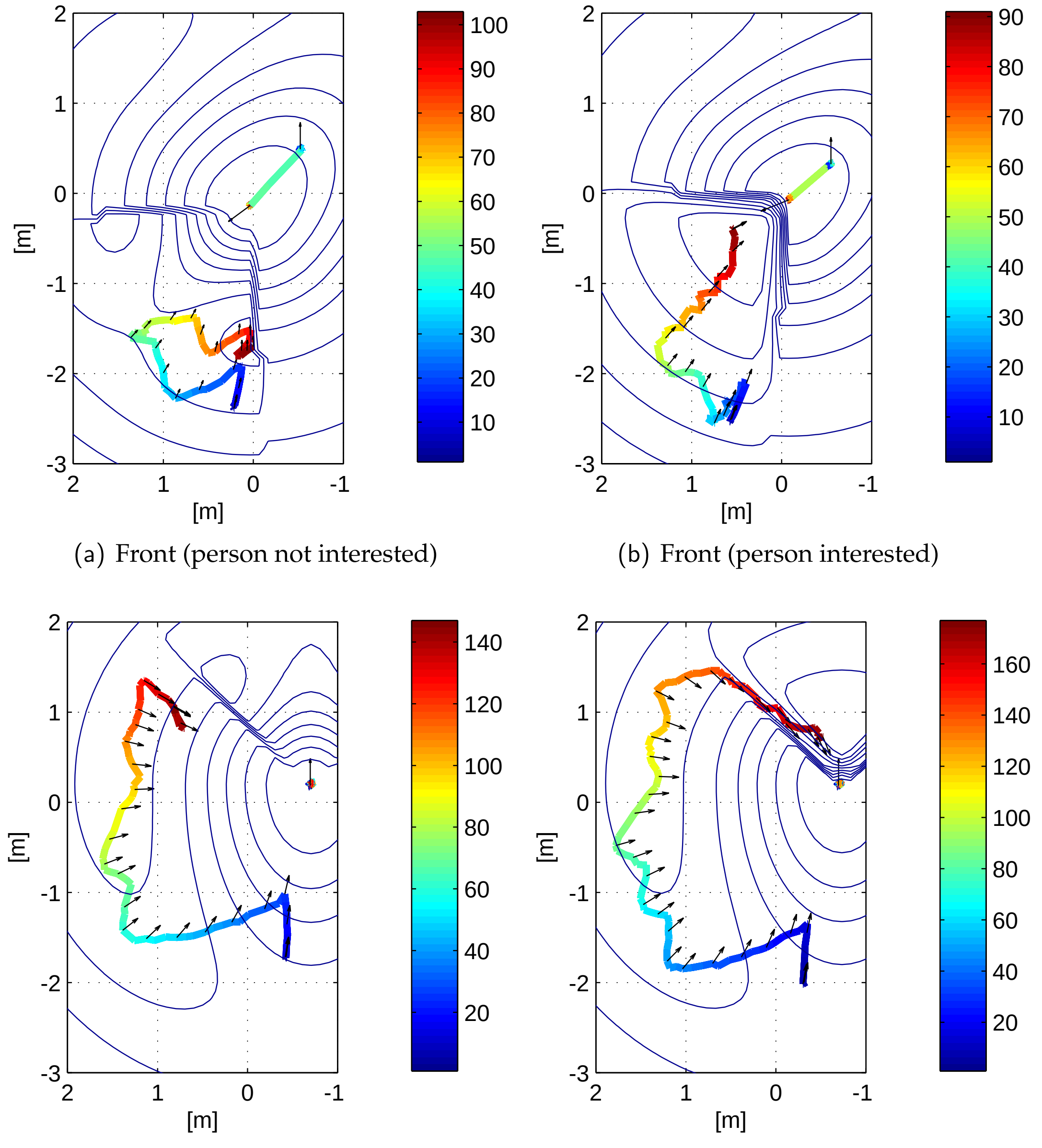
Results from real-world test

The results of the real-world human-aware navigation experiments are presented in Figure 4.7.

Generally, the results from the real-world test are very similar to the simulation. However, in the frontal approach cases of Figures 4.7(a) and 4.7(b), the robot is allowed to move for approximately $50\,\mathrm{s}$ before the person moves towards it. This, more significantly shows the robot altering the approach strategy from back to frontal.

By comparing Figures 4.7(c) and 4.6(c), a difference between simulation and real-world is noticed. In the real-world test, the robot does not reach the desired approach angle of $45°$ relative to the person, before continuing its approach. Instead, the robot approaches the person in an angle of approximately $60°$. This discrepancy is believed to be primarily caused by the non-ideal conditions of the real-world scenario, combined with the fact that the person is not moving. Hence, the initial determination of the person's heading performed by the Person Detector, seems to deviate from the correct value. Furthermore, the fact that the robot, while interacting with the person, does not use the Localizer functionality results in the robot keeping track of its position relative to the person by odometry only. This will inevitably cause some error. However, due to the fact that an encountered person is most likely in motion when being approached, the deviation of $15°$ from the desired approach angle seems of minor importance to the functioning of the system, since this motion will constantly provide the robot with the person's heading.

A general difference between the simulation and the real-world test, is the robot exhibiting a more rocking motion in the real-world test. This might be caused by a larger response time of the real-world robot due to e.g. image processing and the amount of network communication needed due to the distributed nature of the system.



(a) Front (person not interested)

(b) Front (person interested)

(c) Back (person not interested)

(d) Back (person interested)

FIGURE 4.7: *Results from the real-world human-aware navigation test. The blue lines represent the contours of the theoretically calculated behaviour grid. The paths of the robot and the person are depicted by the multi-coloured lines of which the colour gradient represents the elapsed time in relation to the colour bar to the right of each plot. Arrows indicate the current heading of the robot and person.*

Generally, the results of both the simulated and real-world tests are very promising as they show that the developed behaviour-based navigation algorithm functions as intended. Furthermore the tests show very little difference between the simulations and the real-world scenario. The results show that the algorithm can indeed be implemented on a robot functioning in a real-world scenario, with its non-ideal conditions introducing e.g. sensor noise.

4.3.2 Learning

Since the robot must be able to learn from its various person encounters, the CBR feature of the Trainee functionality must be tested separately, to verify its functioning. Testing a CBR solution properly involves a great amount of controlled repetitions, and thus, the test is carried out by simulation only. In addition to testing the Trainee functionality, the test will also treat the Person Evaluator functionality, providing some of the data for the Trainee to experience from.

Again, the two test cases of the person being in "interested" and "not interested" are considered. As described in Section 4.1.1, a virtual test person subject have been created to, like in the real-world, act differently according to the given case. This is illustrated in Figure 4.8.

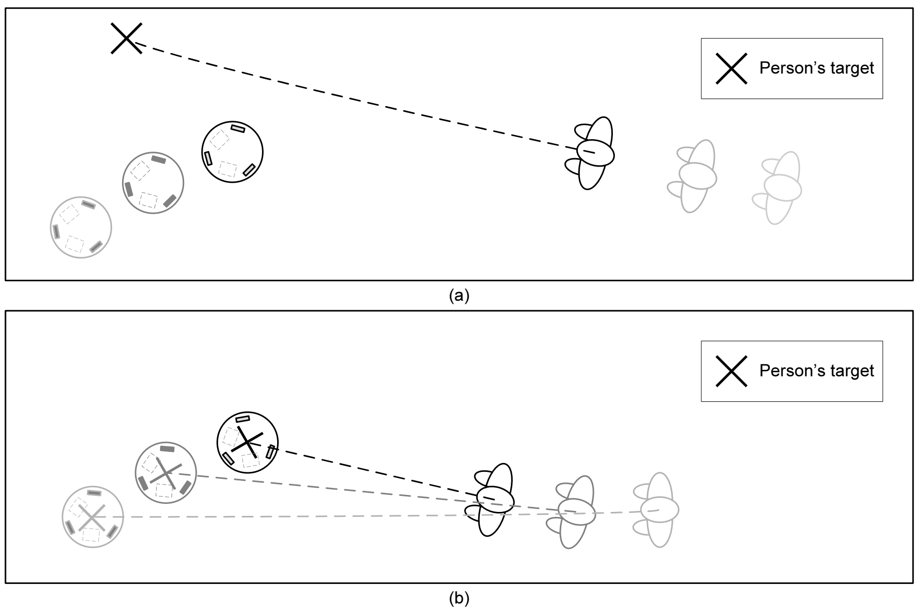


FIGURE 4.8: *Set-up for testing the Trainee functionality of the robot. In (a) the person avoids the robot and in (b) the person approaches the robot illustrating the presumed behaviour of a person when he is "not interested" and "interested" respectively.*

In Figure 4.8(a) the target of the person is fixed, meaning that the person will avoid the robot and any other obstacles in order to reach the target. This results, in the person exhibiting a "not interested" behaviour towards the robot. In the other case, depicted in Figure 4.8(b), the target of the person will at all times be the position of the robot. This causes the person to approach the robot and thereby exhibit an "interested" behaviour.

The robot is set to initiate communication with the person when it reaches a predefined distance of $0.6\,\mathrm{m}$ from the person. Answers from the person are provided by the class `Communicator` of the Controller, being executed in simulation mode, which generates the proper reply according to the given case of interest tested for.

In total, three tests are performed each initiated with an empty main case database, `stored_cases` (see Section 3.4.3). Furthermore, a high learning rate is used to make sure that the robot learns relatively fast from its encounters. Details of the three test scenarios are specified below:

Untrained robot - person "interested" The main case database is cleared. A training run is performed in which the person is "interested". This test will verify if the main case database evolves as intended.

Untrained robot - person "not interested" The main case database is cleared. A training run is performed in which the person is not interested. This test will verify whether the main case database evolves as intended.

Trained robot The main case database is cleared. 10 training runs are performed in which the person is "interested". 10 training runs are performed in which the person is "not interested". One test run is performed in which the person is "interested". One test run is performed in which the person is "not interested". This test will verify, whether the robot makes the right decisions regarding the person interest based on prior encounters.

During the three tests, the motions of the robot and the person are logged along with the evolution of the main case database (`stored_cases`).

Results

The two training runs with the untrained robot result in the main case databases in Table 4.1 and 4.2, respectively. Both of the above tables show that the evolution of the main case database is as intended. Thus, in the "interested" case of Table 4.1 the `indication` values increase as the `distance` decreases. Contrary to this, in the "not interested" case of Table 4.2, the `indication` values decrease along with the `distance`. This shows, that the robot is able to learn from an encounter, and

id	distance	area	pos_x	pos_y	indication
1	3.600	0.400	-2.000	0.000	0.500
2	3.400	0.500	-2.000	0.000	0.511
3	3.300	0.500	-2.000	0.000	0.516
4	3.200	0.000	-2.000	0.000	0.521
5	3.100	0.500	-2.000	0.000	0.527
6	3.000	0.000	-1.000	0.000	0.532
7	2.900	0.200	-1.000	0.000	0.537
8	2.800	0.200	-1.000	0.000	0.543
9	2.700	0.000	-1.000	0.000	0.548
10	2.600	0.000	-1.000	0.000	0.553
11	2.500	0.000	-1.000	0.000	0.559
12	2.400	0.000	-1.000	0.000	0.564
13	2.300	0.000	-1.000	0.000	0.569
14	2.200	0.000	-1.000	0.000	0.575
15	2.000	0.000	-1.000	0.000	0.585
16	1.900	0.000	-1.000	0.000	0.591
17	1.800	0.000	-1.000	0.000	0.596
18	1.700	0.200	-1.000	0.000	0.601
19	1.600	0.000	-1.000	0.000	0.607
20	1.500	0.000	0.000	0.000	0.612
21	1.400	0.000	0.000	0.000	0.617
22	1.300	0.000	0.000	0.000	0.623
23	1.200	0.000	0.000	0.000	0.628
24	1.100	0.100	0.000	0.000	0.654
25	1.000	0.000	0.000	0.000	0.679
26	0.900	0.000	0.000	0.000	0.705
27	0.800	0.000	0.000	0.000	0.730
28	0.700	0.000	0.000	0.000	0.756
29	0.600	0.000	0.000	0.000	0.782

TABLE 4.1: *The main case database (`stored_cases`) after simulating one encounter in which both the robot and person were moving. Furthermore, the person was "interested" as can be seen from the fields of `indication` values ranging from the default of* 0.5 *to* 1*. Note, that in accordance with the described weight functions in Section 2.3.5, the values are altered more extensively, the closer the robot gets to the person.*

furthermore, that the robot regards close encounters more credible than encounters at larger distances. The motion of the person and the robot during the above test runs is depicted in Figures 4.9(a) and 4.9(c), respectively.

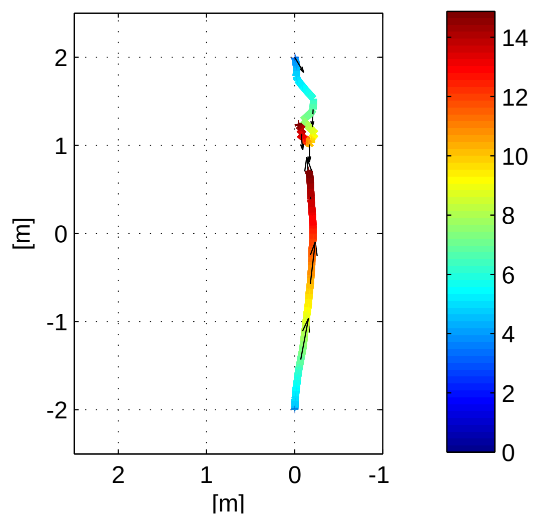
Having verified that the encounters between robot and a person are reflected in the main case database, the next experiment shows the effect of learning on the motion of the robot. A total of 20 training runs are performed to train the robot; 10 in which the person is "interested" and 10 in which the person is "not

id	distance	area	pos_x	pos_y	indication
1	3.300	0.200	-2.000	0.000	0.484
2	3.200	0.100	-2.000	0.000	0.479
3	3.100	0.000	-1.000	0.000	0.473
4	3.000	0.200	-1.000	0.000	0.468
5	2.900	0.100	-1.000	0.000	0.463
6	2.800	0.100	-1.000	0.000	0.457
7	2.700	0.000	-1.000	0.000	0.452
8	2.600	0.300	-1.000	0.000	0.447
9	2.500	0.300	-1.000	0.000	0.441
10	2.400	0.200	-1.000	0.000	0.436
11	2.300	0.000	-1.000	0.000	0.431
12	2.200	0.200	-1.000	0.000	0.425
13	2.100	0.100	-1.000	0.000	0.420
14	1.800	0.200	-1.000	0.000	0.404
15	1.700	0.100	0.000	0.000	0.399
16	1.600	0.200	0.000	1.000	0.393
17	1.500	0.100	0.000	1.000	0.388
18	1.400	0.100	0.000	1.000	0.383
19	1.300	0.100	0.000	1.000	0.377
20	1.200	0.100	0.000	1.000	0.372
21	1.100	0.100	0.000	1.000	0.346
22	1.000	0.100	0.000	1.000	0.321
23	0.900	0.000	0.000	1.000	0.295

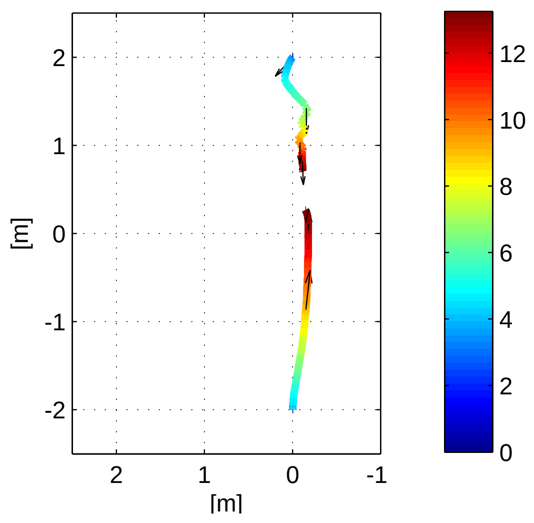
TABLE 4.2: *The main case database (`stored_cases`) after simulating one encounter in which both the robot and person were moving. Furthermore, the person was "not interested" as can be seen from the fields of `indication` values ranging from the default 0.5 to 0. Note that, in accordance with the described weight functions in Section 2.3.5, the values are altered more extensively, the closer the robot gets to the person.*

interested". This results, in a main case database with approximately 200 entries. Using this database as the robot's experiences of prior encounters, two test runs, one "interested" and one "not interested", are performed. The results from these test runs are depicted in Figures 4.9(b) and 4.9(d), respectively.

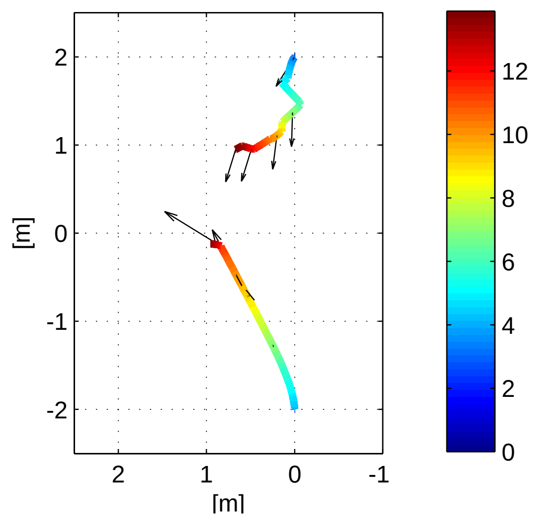
Comparing Figures 4.9(a) and 4.9(b), it is seen, that the behaviour of the untrained robot and the trained robot in the "interested" case does not deviate significantly at large distances. This results from the fact that the robot is not able to differentiate between "interested" and "not interested" behaviour at large distances due to the lower weighting of the experiences. Hence, in the trained case the person interest indication will be close to the default value of 0.5. In contrast, when the distance reaches approximately $2\,\mathrm{m}$ at time $t \approx 9\,\mathrm{s}$ (the yellow coloured



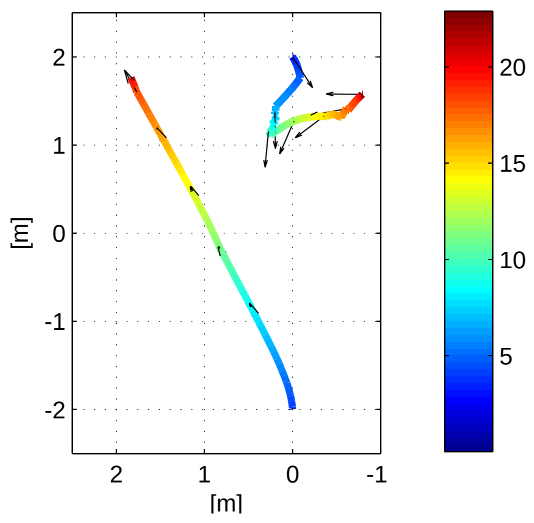
(a) Untrained robot (person "interested")



(b) Trained robot (person "interested")



(c) Untrained robot (person "not interested")



(d) Trained robot (person "not interested")

FIGURE 4.9: *Results from testing the robot's learning functionality. The robot's initial position is* $(2, 0)$*, whereas the person is initially located at* $(-2, 0)$*. In the "not interested" case, the desired goal of the person is located at* $(2, 2)$*. The paths of the robot and the person are depicted by the multi-coloured lines of which the colour gradient represents the elapsed time in relation to the colour bar to the right of each plot. Arrows indicate the current heading of the robot and the person.*

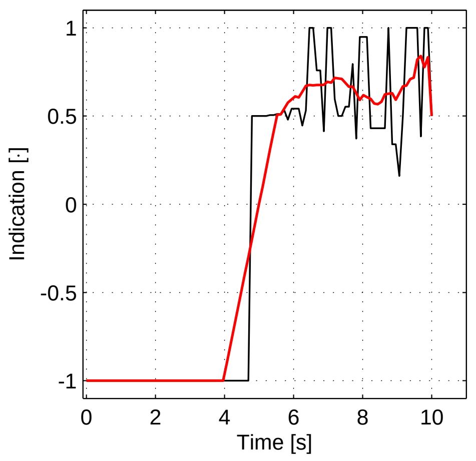
area), the paths of the trained robot and the untrained robot start to deviate. In the untrained case, the robot starts to reverse while deviating to the right. This is caused by the forward part of the behaviour grid not being rotated due to the person interest indication not yet having changed from the default value of 0.5. In the trained case however, the robot correctly evaluates the person as "interested" based on the past experiences. Thus, the forward part of the behaviour grid is rotated, allowing the robot to approach the person frontally.

The case in which the person is "not interested" depicted in Figures 4.9(c) and 4.9(d) shows significant changes in robot behaviour after the training has been performed. The untrained robot of Figure 4.9(c) assumes that the person is interested and approaches him until actually preventing the person from continuing his path towards the goal. However, the trained robot, based on the experiences stored in database, correctly evaluates the behaviour of the person as "not interested". Thus, the robot keeps a large distance to the person allowing him to reach his goal without robot interference.

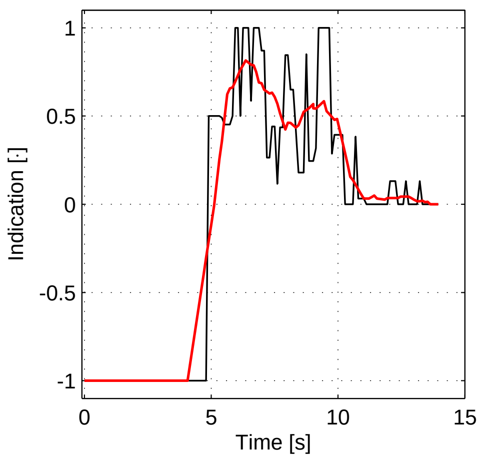
A more detailed look at the person interest estimated by the robot is presented in Figure 4.10.

The figure shows the evolution of the person interest indication of the two test runs. At first, the indication is fixed at -1 due to the Behaviour Manager not being invoked. When invoked, it is seen that the interest indication initially is quite similar in the two tests. This is explained by two person behaviours looking similar to the robot at large distances, along with fact that the credibility of the robot's perception of person behaviour being weighted lower at large distances. As time elapses, the indications evolve differently. In the "interested" case of Figure 4.10(a), the trend of the indication is increasing while it in the "not interested" case of Figure 4.10(b) is decreasing. Finally, the indications end close to the values of 1 and 0, respectively, consistent with the fact that 1 indicates "interested", while 0 indicates "not interested".

Generally, the conducted experiments on the robot's Trainee functionality show that the method of CBR implemented in this project, can advantageously be applied to a robot, which needs to evaluate the behaviour of a person.



(a) Person interest indication (person interested)



(b) Person interest indication (person not interested)

FIGURE 4.10: *The person interest indication estimated by the robot during the two test runs of a person being "interested" and "not interested", respectively. The black line shows the actual indications, while the red line is smoothed by using a sliding window of* 15 *samples to illustrate the trend of the indication evolution. Notice that the indications end close to the values of* 1 *and* 0*, respectively, consistent with the fact that* 1 *indicates "interested", while* 0 *indicates "not interested".*

4.4 After encounter

In order to comply with the robotic context description in Section 1.1, the robot must be able to guide a person to his/her desired destination, avoiding any unknown obstacles on its way.

The test is carried out by placing the robot in the test environment along with an obstacle not included on the map. Providing the robot with a target destination, and placing the obstacle such that the robot must avoid both known and unknown obstacles, it remains to record all data on the motion of the robot. During simulation, motion data are logged to a file, while robot positions are extracted from the camera recordings in the real-world setting.

The following sections presents the experimental results obtained from the conducted simulation and real-world testing of the robot's guiding functionality. In both environment settings, the robot was provided an initial pose of $(1.5, 1, 0)$ and a target destination of $(-1.5, 1, 90)$. The obstacle, measuring $0.28\,\mathrm{m}$ in diameter was placed in $(0.5, 0)$.

4.4.1 Results from simulation

The results of the simulated test scenario is illustrated in Figure 4.11, showing the travelled path of the robot.

As described in Section 2.3.3, the Navigator plans routes for the robot in direct lines. Thus, to avoid the obstacle to the left of the robot's initial starting point, the Navigator directs the robot towards the centre of the environment by an intermediate way-point, whereafter a direct line can be drawn to the target destination. However, as seen from the figure, the robot's path is blocked by the unknown obstacle in $(0.5, 0)$, and thus the Pilot functionality takes over the control to avoid the unexpected hindrance.

Having steered clear of the unknown obstacle, the robot resumes to follow the way-point(s) designated by the Navigator.

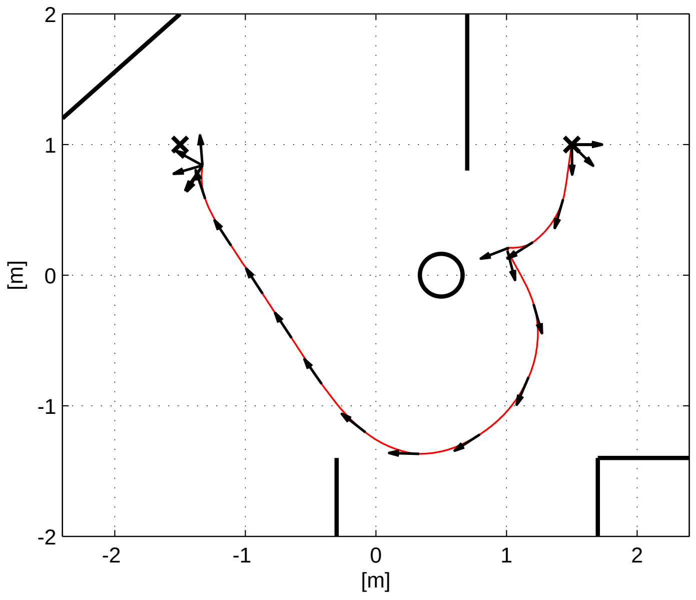


FIGURE 4.11: *Results from the simulated case of the "after encounter" scenario, testing the robot's ability to move to a designated target. The robot's initial position and target destination are marked with an* ×*. The arrows along the travelled path of the robot indicates its current heading. The obstacle is marked with a* ○*.*

4.4.2 Results from real-world test

Turning to the real-world case, Figure 4.12 presents the results of the test conducted using the Robotino® robot in a real-world setting.

Comparing with the simulated case, the real-world test shows the exact same pattern of the Navigator and Pilot exchanging robot control. In the real-world setting, it is moreover seen that the Pilot first seeks to guide the robot in a counter-clockwise path around the unknown obstacle, but concludes that the opening is too narrow, and chooses the clockwise way around instead. Considering the results presented above, the robot's capability of guiding a person to a desired location is functioning as intended.

Having presented the results of the conducted simulations and real-world experiments, the following chapter elaborates on these results, and furthermore on the project findings in general.

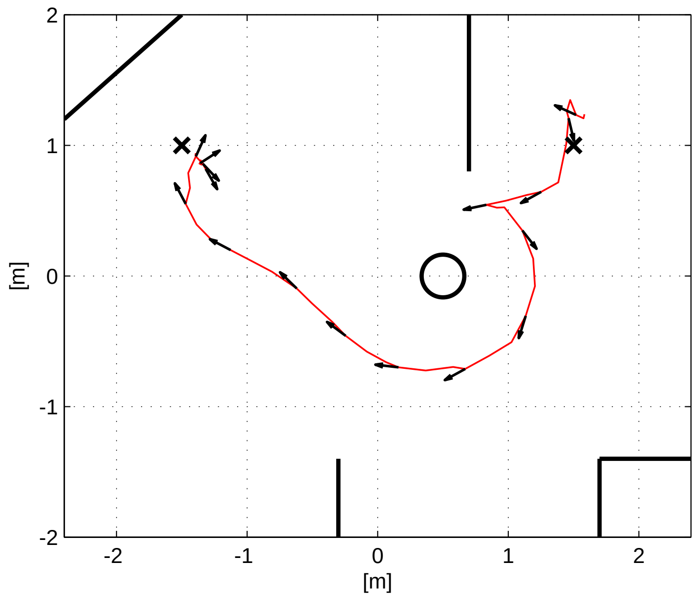


FIGURE 4.12: *Results from the real-world environment, when testing the robot's ability to move to a designated target. The initial position and target of the robot are marked with an* ×*, while the robot heading is indicated with arrows along the travelled path. The obstacle is marked with a* ○*.*

CHAPTER 5
Closure

5.1 Conclusion

As described in Section 1.2, the overall objective of this project has been to make a Robotino® robot capable of navigating in a human environment, and furthermore to target the problem of successful HRI initiation, concerning the initial, and very important of successfully establishing communication between man and machine.

A behavioural controlling algorithm has been proposed and validated through a number of experiments conducted in both a simulated and a small scale real-world scenario. From the results obtained, the robot shows promising human-friendly spatial behaviours, complying to the so-called Hall zones known from prior studies of human-human interaction [Hall et al., 1968]. Furthermore, the distribution-based nature of the algorithm provides for uncomplicated alterations of the spatial behaviour, if such is desired by future users and developers.

To make the behaviour of the robot intelligent with respect to human behaviour, a CBR solution has been developed. This solution, in conjunction with the behaviour algorithm, enables the robot to distinguish between different human behaviours, and base its own behaviour on the behaviour of the human. The developed learning functionality has been verified by simulation experiments showing that the robot, subject to only a limited training, is indeed able to distinguish between two human behaviours and base its own behaviour on these. Furthermore, having based the implementation of CBR on *MySQL*, offers an easy accessible API in regards to future development.

Besides from the main contributions, the project work has spawned various sub-contributions benefiting both the HRI research field as well as future projects at SAC. Thus, prior to the commencement of this project, no *Player* driver existed for the Robotino® platform to make it conform with the *Player* framework. Hence, drivers have been developed for the Robotino® as well as for the additional range finder sensors. The function of these drivers has been verified implicitly through the promising results of overall system experiments. Consequently, the drivers have been submitted for possible inclusion in upcoming releases of the *Player/Stage* distribution, and furthermore published on the Wiki accompanying the workings of this project.

In conclusion, this project has proposed a new behavioural controlling algorithm targeting the preliminary steps of human-robot encounters, entailing a number of significant contributions bringing the Human Robot Interaction research one step further towards a fully autonomous and human-aware robot.

5.2 Future Work

The real-world experiments conducted to verify the functioning of the system, have all been performed in an enclosed small-scale test environment, where disturbances from e.g. dynamics and lighting are easily controlled. For the system to robustly function in a real-world scenario, requires some additional work.

Starting from the developed behaviour algorithm, means could advantageously be taken to support avoidance of obstacles while interacting with a person. One possible way to incorporate such capabilities, is to include detected obstacles in the behaviour grid, thus, applying the same distributional approach used for the behaviour algorithm.

Moreover, attention could be directed on the capabilities of the Person Detector, enabling it to facilitate more advanced human recognition. The Person Detector could be extended by e.g. incorporating more advanced sensor fusion capabilities to unite the camera and range finder inputs, obtaining an increased degree of person detector certainty and precision. Also, completely different person detection methodologies could be considered, involving e.g. leg detection using the range finder, or facial recognition using the camera.

The documented effect of the developed CBR solution, favours that learning capabilities should indeed be part of future development on the system. Thus, further improvements could reside in incorporating currently omitted features, in order to provide the robot with additional possibilities for differentiating between all experienced encounters.

As for the Robotino® platform, this project has focused on distributing the system solution, such that debugging and system monitoring have been performed more easily on a regular PC. However, such a distributed system requires communicating entities to be connected, introducing a severe restriction on robot mobility and automaticity. Thus, future development could possibly be concerned on manipulating and redesigning the current divided structure, such that it applies for sole implementation on the robot.

Finally, the communicative skills of the robot could be considered for improvement, by e.g. incorporating a display on the robot. Work is already being done inside SAC, to develop a robot display mimicking the likes of a human head, but other improvements such as introduction of speech capabilities, would involve vast possibilities for future experiments and modes of application.

Acronyms

API	Application Programming Interface
AMCL	Adaptive Monte Carlo Localization
CBR	Case-Based Reasoning
EKF	Extended Kalman Filter
FIFO	First In First Out
FOV	Field Of View
GUI	Graphical User Interface
HRI	Human Robot Interaction
POD	Polar Obstacle Density
SAC	Section of Automation and Control
SLAM	Simultaneous Localization And Mapping
STL	Standard Library Template
UML	Unified Modelling Language
VFH+	Vector Field Histogram+

Bibliography

A. Aamodt and E. Plaza.
Case-based reasoning - foundational issues, methodological variations, and system approaches, 1994.
`http://www.iiia.csic.es/People/enric/AICom.html`.

Boost.
Boost C++ libraries, 2007.
URL `http://www.boost.org`.
12.05.2007.

J. Borenstein and Y. Koren.
Vector field histogram - fast obstacle avoidance for mobile robots.
IEEE Journal of Robotics and Automation Vol 7, No 3, June 1991, pp. 278-288, 1991.

C. L. Breazeal.
Designing Sociable Robots.
MIT Press, 2002.
ISBN 0-262-52431-7.

J. Bruce.
Cmvision, 2006.
`http://www.cs.cmu.edu/~jbruce/cmvision/`.

H. Bruyninckx.
Open robot control software, 2006.
`http://www.orocos.org/`.

W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun.
Experiences with an interactive museum tour-guide robot.
Artificial Intelligence, 114(1-2):3-55, 1999.

J. T. Butler and A. Agah.
Psychological effects of behavior patterns of a mobile personal robot.
Autonomuos Robots 10, 185-202, 2001, 2001.

CARMEN.
Carmen robot navigation toolkit, 2006.
http://carmen.sourceforge.net/.

T. H. J. Collet, B. A. MacDonald, and B. Gerkey.
Player 2.0: Toward a practical robot programming framework, 2005.
http://www.araa.asn.au/acra/acra2005/papers/collet.pdf.

K. Dautenhahn.
Methodology & themes of human-robot interaction: A growing research field.
International Journal of Advanced Robotic Systems, Vol. 4, No. 1, 2007.

S. J. Delany.
Using case-based reasoning for spam filtering, 2006.
http://www.comp.dit.ie/sjdelany/publications/thesis-final.pdf.

Frauenhofer.
Care-o-bot ii, 2004.
http://www.care-o-bot.de/.

A. García-Rojas, F. Vexo, D. Thalmann, A. Raouzaiou, K. Karpouzis, and S. Kollias.
Emotional body expression parameters in virtual human ontology.
1st Int. Workshop on Shapes and Semantics, 2006.

A. Garulli, A. Giannitrapani, A. Rossi, and A. Vicino.
Mobile robot slam for line-based environment representation.
Proceedings of the 44th IEEE Conference on Decision and Control, 2005.

E. T. Hall, R. L. Birdwhistell, B. Bock, P. Bohannan, A. R. D. Jr., M. Durbin, M. S. Edmonson, J. L. Fischer, D. Hymes, S. T. Kimball, W. L. Barre, S. J. F. Lynch, J. E. McClellan, D. S. Marshall, G. B. Milner, H. B. Sarles, G. L. Trager, and A. P. Vayda.
Proxemics.
Current Anthropology, Vol. 9, No. 2/3. (Apr. - Jun., 1968), pp. 83-108., 1968.

HONDA.
Asimo - the honda humanoid robot asimo, 2006.

http://world.honda.com/ASIMO/.

H. Huettenrauch, K. S. Eklundh, A. Green, and E. A. Topp.
Investigating spatial relationships in human-robot interaction.
Cogniron, 2006.

Z. Huiliang and H. S. Ying.
Dynamic map for obstacle avoidance.
Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE, Vol 2, pp. 1152-1157, 2003.

E. Isaacs and A. Walendowski.
Designing from both sides of the screen.
SAMS - New Riders, 2001.
ISBN 0-672-32151-3.

M. Kleinehagenbrock, J. Fritsch, and G. Sagerer.
Supporting advanced interaction capabilities on a mobile robot with a flexible control system.
Intelligent Robots and Systems, 2004.
http://ieeexplore.ieee.org/iel5/9577/30278/01389982.pdf.

K. L. Koay, K. Dautenhahn, S. N. Woods, and M. L. Walters.
Empirical results from using a comfort level device in human-robot interaction studies.
HRI'06, March 2-4, 2006, 2006.

J. Kolodner.
Case-Based Reasoning.
Morgan Kaufmann, 1993.
ISBN 1-55860-237-2.

MARIE.
Mobile and autonomous robotics integration environment, 2006.
http://marie.sourceforge.net/.

J. S. Mathiasen, M. . Halse, and R. A. L. Sørensen.
A framework for robot navigation with case based reasoning, 2006.

Orca.
Orca: Components for robotics, 2006.
`http://orca-robotics.sourceforge.net/.`

PlayerProject.
Player - user / reference manual, 2006.
`http://playerstage.sourceforge.net/doc/Player-2.0.0/player/.`

PYRO.
Python robotics, 2006.
`http://emergent.brynmawr.edu/~dblank/pyro/.`

E. A. Sisbot, R. Alami, T. Simeon, K. Dautenhahn, M. Walters, and S. Woods.
Navigation in the presence of humans.
IEEE-RAS International Conference on Humanoid Robots Humanoids2005 December 5-7, 2005 Tsukuba Japan, 2005.

E. A. Sisbot, A. Clodic, L. F. Marin, M. Fontmarty, L. Bréthes, and R. Alami.
Implementing a human-aware robot system.
IEEE International Symposium on Robot and Human Interactive Communication 2006 (RO-MAN 06), Hatfield, U.K., 2006.

G. Strang and K. Borre.
Linear Algebra, Geodesy, and GPS.
Wellesley-Cambridge Press, 1997.
ISBN 0-9614088-6-3.

Tangentsoft.
MySQL++, 2007.
`http://tangentsoft.net/mysql++/.`

T. The Danish Ministry of Science and Innovation.
Technology foresight on cognition and robotics, 2006.
`http://teknologiskfremsyn.dk/download/195.pdf.`

S. Thrun, W. Burgard, and D. Fox.
Probabilistic Robotics.
MIT Press, 2005.

ISBN 0-262-20162-3.

I. Ulrich and J. Borenstein.
Vfh+: Reliable obstacle avoidance for fast mobile robots.
Proceedings of the 1998 IEEE International Conference on Robotics and Automation, 1998.

M. L. Walters, K. Dautenhahn, K. L. Koay, C. Kaouri, R. t. Beokhorst, C. Nehaniv, I. Werry, and D. Lee.
Close encounters: Spatial distances between people and a robot of mechanistic appearance.
Cogniron, 2005.

APPENDIX A

Framework comparison

SAC has a desire of making the software developed through this project reusable for other robotics researchers using different types of robots. Furthermore, this also has the advantage that software developed for other projects can be reused during this project. For this reusabilty to be possible, it is necessary to base the developed software on some platform which is available within the research community. A number of such platforms exist each having different features. The requirements for the software platform can now be specified as listed below:

- Open Source software.
- Facilitate the development of an interface to the Robotino® sensors and actuators.
- Used within the robotics research community.
- Allow for the low level software such as sensor drivers to be written in C++ as this is the language used for the software delivered with the Robotino®.
- Provide possibility for the low level software and control software to run on separate machines and communicate via e.g. WLAN. This is due to a desire of monitoring the progress of the control software at runtime. Robotino® itself does not provide means for this.
- Must contain a simulator for testing the control software without using the robot. Using the robot for intermediate testing might result in hardware damage due to bugs or shortcomings in the control software.
- Must be relatively simple to get familiarized with since this software platform is not the primary focus of the project.
- Must be able to run on a Linux platform as this is installed on the Robotino®.

The most widely used platforms have been reviewed and the primary features are listed in Table A.1.
At the time of writing, none of the mentioned platforms support the Robotino®. Furthermore, only some of them conform to the requirements listed above. *Carmen* [CARMEN, 2006], *Orca* [Orca, 2006], and *Pyro* [PYRO, 2006] are developed for specific robots, and apparently the addition of another hardware platform is not provided for. Turning to *Orocos*, the prime focus of this software is industrial

Platform	**Programming language**	**Supports distribution**	**Hardware specific**
Carmen	C	No	Yes
Orca	C++	No	Yes
Orocos	C++	No	No
Pyro	Python	No	Yes
Marie	C++	Yes	No
Player	C/C++/Java	Yes	No

TABLE A.1: *The primary features of the reviewed software platforms.*

robots, hence no software is at present available for mobile robots. Moreover, according to *Orocos* developers: *"The code and documentation are divided over many libraries and directories. This can slow down new users in getting a grip of the whole project, or finding the solutions they are looking for"* [Bruyninckx, 2006]. *Marie* can be *"used for building robotics software systems by integrating previously-existing and new software components"* [MARIE, 2006]. This implies, that using *Marie* requires the use of at least one other software platform such as *Carmen* or *Player*. Hence, *Marie* is not a reasonable choice for this project. In contrast, *Player* complies with the requirements listed above and is thus chosen for software platform for this project.

APPENDIX B
RobotinoCom **API**

The Robotino® comes bundled with a C++ API for accessing the robot's hardware. These access functions are listed below along with a short description of what the function does.

```
void setVelocity( unsigned int motor, float rpm )
```
Sets the velocity of motor to `rpm`. Motors are counted starting from 0. This overrides any previous `set/addVelocity` calls for the given motor.

```
void setVelocity( double vx, double vy, double omega )
```
Sets the robot's velocity. `vx` is the velocity in x direction [$\frac{\text{mm}}{\text{s}}$], `vy` is the velocity in y direction [$\frac{\text{mm}}{\text{s}}$], `omega` is the angular velocity [$\frac{\circ}{\text{s}}$].

```
float actualVelocity( unsigned int motor )
```
Returns the `actualVelocity` of motor received by the last `StatusMessage`. If no `StatusMessage` has been received, or if there is no operational connection to the Robotino®, it returns 0.

```
void setKp( unsigned int motor, float value )
```
Sets the PID controllers proportional constant of `motor` to `value`. Motors are counted starting from 0.

```
void setKd( unsigned int motor, float value )
```
Sets the PID controllers differential constant of `motor` to `value`. Motors are counted starting from 0.

```
void setKi( unsigned int motor, float value )
```
Sets the PID controllers integral constant of `motor` to `value`. Motors are counted starting from 0.

```
bool bumper()const
```
Returns **`true`** if the bumper of the Robotino® is pressed otherwise **`false`**.

```
float distance( unsigned int n )const
```
Returns the reading of distance sensor `n`. Counting of the sensors starts from 1.

`void setDigitalOutput( std::string name, bool value )`

Sets the digital output referenced by `name` to low, if value is **`false`**. If value is **`true`** the output goes high.

`void setDigitalOutput( unsigned int n, bool value )`

Sets the digital output `n` to low, if value is **`false`**. If value is **`true`** the output goes high. The output channels are counted starting from 0.

`bool digitalInput( std::string name )const`

Returns **`true`** if the digital input referenced by `name` is high, and **`false`** if low. In cases where the `name` is not found, `digitalInput()` returns **`false`**.

`float voltageBatt1plus2()const`

Returns the voltage of the two batteries.

APPENDIX C
Interface specifications

Interface	Between	Description
`blobfinder:0`	Blob finder and Person Detector	Used for transferring extracted blob information of a certain colour to the Person Detector which utilizes the blobs when detecting a person.
`camera:0`	Robotino® and Blob finder	Used for transferring images from the Robotino® webcam to the Blob finder.
`laser:0`	Range finder and Person Detector	Used for transferring laser readings which are utilized by the Person Detector for estimating the distance to a person.
`laser:0`	Range finder and Pilot	Used for transferring laser readings to the Pilot, utilizing the readings for detecting and avoiding obstacles not present on the map.
`laser:0`	Range finder and Localizer	Used for transferring laser readings which are utilized by the Localizer to estimate the pose of the robot.
`localize:0`	Localizer and Controller	Used for transferring pose estimates and likelihoods to the Controller.
`map:0`	Map and Localizer	Used by the Localizer to estimate the correct pose of the robot, by matching detected environment features with the provided map.
`opaque:0`	Controller and Behaviour Manager	Customized interface, further specified in Section 3.3.1.
`opaque:1`	Controller and Velocity Manager	Customized interface, further specified in Section 3.3.2.
`planner:0`	Controller and Navigator	Transfers targets from Controller to Navigator.
`pos2d:0`	Robotino® and Velocity Manager	Transfers velocity commands from Velocity Manager to the Robotino®. Furthermore, position data intended for above drivers is transferred from the Robotino® to the Velocity Manager.
`pos2d:0`	Robotino® and Localizer	Odometry data is transferred from the Robotino® to the Localizer, where it is used for pose estimation.
`pos2d:1`	Behaviour Manager and Pilot	Velocity commands are passed to the Robot through the Pilot. Furthermore, pose information is transferred from the Pilot to the Behaviour Manager.
`pos2d:2`	Localizer and Controller	The most likely pose of the robot is transferred to the Controller.
`pos2d:2`	Localizer and Person Detector	The most likely pose of the robot is transferred to the Person Detector, where it is used for transforming a detected person's pose from robot coordinates to world coordinates.
`pos2d:3`	Behaviour Manager and Navigator	A way-point is transferred from the Navigator to the Behaviour Manager. Furthermore, pose information is transferred from the Behaviour Manager to the Navigator.
`pos2d:4`	Pilot and Velocity Manager	Velocity commands are transferred from the Pilot to the Velocity Manager where it, if applicable, is limited to the maximal allowed velocity.
`pos2d:5`	Person Detector and Controller	The position of a detected person is transferred from the Person Detector to the Controller in robot frame coordinates.
`pos2d:6`	Person Detector and Controller	The position of a detected person is transferred from the Person Detector to the Controller in robot frame coordinates.

TABLE C.1: *Description of the interfaces used within the overall system solution as depicted in Figure 3.1 on page 45.*