Human-robot interactive collision prevention
Improved navigation in home environments

Laurie Bax (0710504)
Artificial Intelligence
Radboud University Nijmegen

August 30, 2012

Master's Thesis in Artificial Intelligence

Author:	Laurie Bax
Student number:	0710504
Email:	lauriebax@student.ru.nl

Supervisors:

dr. Louis Vuurpijl	Dietwig Lowet
Artificial Intelligence	Human Interaction & Experiences
Radboud University Nijmegen	Philips Research Eindhoven

Abstract

It is generally envisaged that in the near future, personal assistance robots will aid people in and around their homes. One of the requirements of these robots it that they can are able to navigate autonomously such that it is safe and comfortable for their users and at the same time efficient for the robot. Path planning and obstacle avoidance are crucial in these contexts. Current obstacle avoidance algorithms are not efficient when people cross the path of the robot. In this thesis four types of obstacle avoidance are compared by efficiency and user preference: static obstacle avoidance, two types of dynamic obstacle avoidance and *interactive collision prevention*(ICP). The efficiency is measured in terms of navigation time, amount of detour and the number of successful trials (without collisions). The results suggest that both dynamic obstacle avoidance algorithms are more efficient than static obstacle avoidance. Furthermore, first explorations in ICP indicate that users can safely and comfortably guide the robot and that ICP is preferred over the other algorithms.

Contents

List of Figures

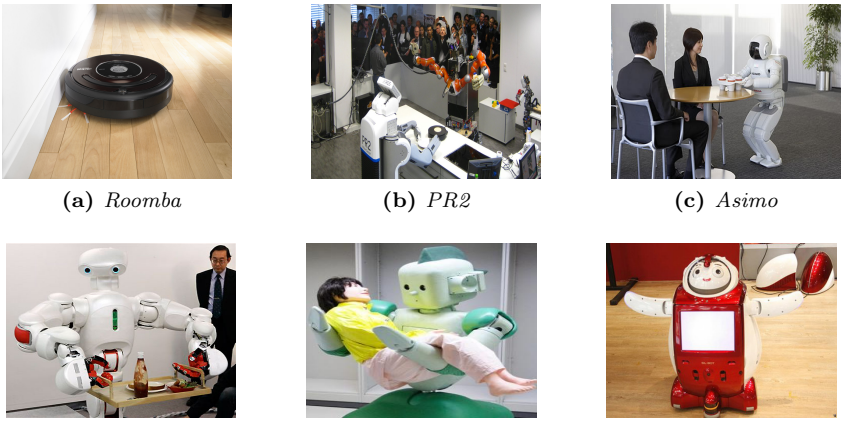
List of Tables

Chapter 1

Introduction

Robots come into our daily lives more and more, in particular personal service robots [30, 58]. Examples are shown in Figure 1.1. These robots are being developed to perform several household tasks, assist in care taking and to keep the residents company. For instance, the Roomba robot (Figure 1.1a) is able to vacuum a room [25]. The PR2 (Figure 1.1b), here baking pancakes [1], and Asimo (Figure 1.1c) [70] are designed to operate in human living spaces, using utilities optimized for humans. The distinctive characteristic is that the last two robots have arms so they can use the same tools as we do. This human-like functionality is a prerequisite for these robots to operate without requiring major modifications to the homes they operate in. These robots can be deployed to perform any number of different household tasks.

Twendy-One (Figure 1.1d) and RI-MAN (Figure 1.1e) are designed for assisting people in care taking. One of their functions is to help elderly up (Twendy-One) [35] or carry them (RI-MAN) [54]. An example of a companion robot is Sil-Bot (Figure 1.1f) [92]. Sil-bot is designed to interact and play with people.



(a) *Roomba* **(b)** *PR2* **(c)** *Asimo*

(d) *Twendy-One* **(e)** *RI-MAN* **(f)** *Sil-Bot*

Figure 1.1: *Examples of personal service robots.*

These personal service robots are often combined with the concept of Smart

Homes [11]. Smart Homes are houses equipped with numerous sensors to monitor the residents. Mobile robots can act as a mobile sensor and in return use the sensors of the house to enhance their performance. The robots can also be used as a telepresence device for family or medical staff [46].

In any case, the robots have to be able to move around in the homes without bumping into furniture or people. In static worlds, this problem is solved. However, home environments are not static. The main problem with autonomous navigation in home environments is the ability of the robot to avoid (moving) people in its path in an efficient way, which is also comfortable and intuitive for the people in the home. Most current day implementations make the robot stop when someone crosses its path. The robot will then wait until that person is out of sight before it resumes its path. Although this approach ensures collision free situations, it is not always the most efficient method for the robot. Humans have solved this problem though. Everyone wants to reach their goal as fast as possible, without colliding with other people. When two (or more) people do seem to be on a collision course, all parties will adjust their path to avoid it. This phenomenon can be observed at all crowded (and less crowded) areas, like train stations, shopping malls and hallways. Somehow, people *cooperatively* adjust their paths to avoid collisions, while the paths remain efficient for all parties.

There are many algorithms that try to navigate a robot from one point to another in a safe way. Although these methods (try to) assure a collision free path, the resulting paths may be unnecessary long or time consuming. Furthermore, some paths may not be comfortable for the people the robot tries to avoid, for instance by cutting them of. An optimal path P, from start position A to goal position B, therefore has to comply with three requirements:

- P is collision free
- P is optimal in distance and time
- P is optimal in user comfort

As will be shown in Chapter 2, only first explorations are being done with (cooperative) people avoidance in robotics. In this thesis we will research two dynamic obstacle avoidance algorithms (see Section 3.1.2) which try to make the robot to avoid people efficiently: namely *Asteroid Avoidance* (AA) and *Human Motion Model Avoidance* (HMMA). Furthermore, we will extend an existing (static) obstacle avoidance algorithm with human-robot interaction possibilities to increase the efficiency of the robot and comfort for the human (see Section 4.1). This new method of people avoidance will be called *Interactive Collision Prevention* (ICP). The main difference between the proposed methods is the knowledge the robot has about the future movements of the user. In this thesis will be investigated which method will provide the most efficient and user friendly obstacle avoidance mechanism. It will be shown that using knowledge about the most probable human movements, as exploited in the AA and HMMA methods, results in significant improvements over the baseline condition. Moreover, we will explore the usability of our new concept of ICP. This is done through a Wizard of Oz experiment in which users move along collision paths and indicate by gesturing to the robot the preferred direction the robot should take.

1.1 Interactive collision prevention

In this section, the concept of ICP is explained. ICP is inspired by the way people avoid each other when they are on a collision course. By using subtle body language, people signal to each other where they are going. When two people are on a collision course, they use this information to avoid a collision in an early stage. Robots could use this information too, to better predict the path of people in their vicinity. They can then adjust their path such that a collision is avoided while the amount of distance traveled and time lost is minimal.

Unfortunately, detecting this kind of body language is very hard, especially on a moving robot. Therefore, the users' intentions have to be signaled in another, more explicit way that the robot can more easily detect. There are many ways to interact with the robot, but the use of gestures is the most logical choice. Gestures are a form of body language, like the subtle cues. However, gestures are better detectable and easier to classify than the cues people give to each other. Another argument is that gestures contain deictic information that can be exploited in navigation. 'Go there' only has meaning when it is known where 'there' is. The spatial nature of a gesture contains precisely this information. Furthermore, people use gestures in their everyday lives too, to communicate their intentions in extreme cases. We all know the situation where we want to move out of the way of the oncoming person, but somehow end up mirroring the other. When you move to your left, the other moves to his right and vice versa. Sometimes this repeats several times before one party explicitly signals which way he is going and waits for the other to react. This concept of interactively solving the collision situation with the other party will be explored in this thesis as ICP.

1.2 Approach

The work presented in this thesis is part of a large European research project Florence [46]. Philips research is one of the prominent research partners in Florence and investigates novel assistive technologies and applications in the area of home automation. A significant amount of work has been put in setting up an appropriate laboratory environment and installing the technologies involved (Roomba, Kinect, ROS, Openni, path planning, Gazebo). It should be noted that these efforts are only briefly described in this thesis.

Using this new laboratory setting, first explorations are made to implement ICP. Furthermore, two additional people avoidance algorithms are implemented. These algorithms use knowledge about the user's behavior and preferences. The performances of the avoidance algorithms are compared to each other and to a standard reactive obstacle avoidance algorithm. The main questions that will be investigated are:

- Which avoidance algorithm is the most robust?
- Which avoidance algorithm is the most efficient, i.e., performs the best in terms of amount of detour taken and time needed to reach the goal?
- Which avoidance algorithm do users prefer?

- Does the new concept of gesture based ICP provide promising opportunities for interactive collision avoidance?

1.3 Outline

The outline of this thesis is as follows. First, state-of-the-art literature is presented in Chapter 2. The background literature contains information about the used concepts in the algorithms. Section 2.2 presents the various parts needed in navigation. Section 2.3 describes previous work in interactive navigation and different types of gesture recognition.

As mentioned before, four different obstacle avoidance algorithms are compared to each other in this thesis. First is the static obstacle avoidance algorithm called the *base algorithm* . As the name suggests, all other obstacle avoidance algorithms are based on this algorithm. Second and third are two dynamic obstacle avoidance algorithms. They each use a different prediction method for predicting the movement of the human. They are called *Asteroid Avoidance* (AA) (using a linear prediction function) and *Human Motion Model Avoidance* (HMMA) (using a priori path information). Finally, the *Interactive Collision Prevention*(ICP) algorithm is explained. ICP allows the human to control and guide the robot, instead of the robot making predictions of possible trajectories of the human.

To verify the performances of the developed algorithms, three types of experiments are designed and conducted. First are the simulation experiments, as described in Chapter 3. Then, the algorithms are tested in the real world. This is described in Chapter 4. Furthermore, the performance of two types of gesture recognition systems is determined in Chapter 5. This thesis is concluded in Chapter 6, with suggestions for future research in Chapter 7.

Chapter 2

State-of-the-art overview

Researchers have been attempting to solve the problem of robot navigation in the presence of people and people avoidance for a long time, with varying approaches and results. A short overview of these approaches can be found in Section 2.2. These methods of people avoidance make use of various sources of knowledge about the movements of people. However, none of these people avoidance methods have an interactive component.

However, there have been attempts to incorporate an interactive component in *navigation*. For instance, Bergh et al. [86] made a robot that would explore uncharted areas. When the robot had multiple areas to explore, people had to point which area should be next. These pointers are not really precise, they only indicate the rough direction where the robot should go, the robot had to reason about the exact position of the goal. The user could choose one out of maximal four directions at a time. Furthermore, the robot was not moving when it got instructions.

Similar research has been done by Park et al. [59]. Here, the researchers tried to estimate the position a user is pointing. This position can then be used to set a goal for the navigation system. Although the first results look promising there are still major constraints on their implementation. For one, the robot is again not moving while it has to detect the pointing position. The total navigation time is increased because the robot has to stop and is therefore not optimal. Furthermore, the authors found a serious trade-off between the accuracy of the position estimations and the time and training data needed for an acceptable performance. Moreover, this interaction method acts more like a controller for the robot than interactive navigation, where both the human and the robot have their own navigation goal.

ICP depends on a mechanism of shared control. The robot will navigate autonomously to its goal. If it gets a command from a user, it will find a temporary goal reflecting that command. The robot is not controlled directly through the interaction, like in the work of Uribe et al. [85]. The robot and the user each have their *own* goal, not a shared one as can be seen in (semi)autonomous wheelchairs [87]. The research done in this thesis in ICP is a first exploration. ICP will be compared to other people avoidance algorithms. All algorithms have the same navigation algorithm which is described in Section 2.1. Other research in navigation is described in Section 2.2. The final section describes the novel interactive navigation concepts and introduces our concept of using

gestures for guiding the robot in cases where it needs additional information.

2.1 Basic navigation

There are many ways to implement a navigation system. However, most systems share the same basic components. The system used in this thesis is the navigation system used by Marder-Eppstein et al. in [44]. This system is proven to work in indoor environments. Furthermore, it is implemented for and freely available in the ROS repository[1]. This navigation system is state-of-the-art and set as a benchmark system. Moreover, the configuration of the robot available for the experiments in this thesis (see Section 4.2.1) is supported by this system.

The basic navigation system has three main components, which cooperate to provide velocity commands to control the robot. These are:

- Localization
- Global path planner
- Local path planner

Each component is explained below. However, first some terminology is explained in Section 2.1.1. Finally, an overview is given of how the components interact with each other to form a navigation system. Algorithm 1 shows a pseudo code representation of the procedures used for navigation.

Algorithm 1 The basic navigation module

```
function NAVIGATION
    goal ← main goal
    Plan global path
    while goal not reached do
        Read sensor data
        Adapt cost map
        Plan local path using DWA
        if no local path possible then
            Execute recovery behaviors
        else
            Generate velocity commands for local path
            Move robot
        end if
    end while
end function
```

2.1.1 Terminology and sensing

To be able to understand navigation in robots, first some important terms have to be explained. Readers that are already familiar with robotic navigation can skip this section and continue reading at Section 2.1.2.

[1] `http://ros.org/wiki/navigation/`

The robot is controlled by sending *velocity commands* to the driver component. The driver translates the velocity commands to motor commands which make the robot move. There are two types of motors, and thus two types of velocity commands to control the robot. First there is a holonomic drive. A drive is said to be holonomic if the number of controllable degrees of freedom is equal to the total number of degrees of freedom. In practice this means that a holonomic robot which can drive on a two-dimensional plane can move in any direction without having to turn first, like a shopping cart. A non-holonomic drive has less controllable degrees of freedom than the total number of degrees of freedom. The motors of a non-holonomic drive can only move forward or backward. The robot can in that case only turn by using a steering axis, like a car, or using one side of wheels at a time (or reversing the opposite wheel), like a wheelchair. The robot that is used in this thesis has a non-holonomic drive. The velocity commands that are given to the robot therefore consist out of a translational (forward/backward speed) and a rotational (turning speed) component instead of a speed and direction.

In order for the robot to avoid obstacles, the robot should be able to sense its environment. There are many types of sensors that are able to sense objects. Below is a list of the most frequently used object detection sensors:

Laser range finder A laser range finder measures the distance to an object by using the *time of flight* principle. It shoots out a laser beam and measures the time it tasks the beam to be reflected back to the sensor. From there the distance of the obstacle in the path of the laser beam can be calculated. By aligning a lot of laser range finders a laser scanner is created which can create a fairly detailed scan of the environment. The scan frequency of such a scanner is usually high (about 100 ms per scan). Furthermore, because a laser range finder uses a laser beam, the scans are not influenced by lighting conditions.

Infrared An infrared sensor emits a beam of infrared (IR) light (not a laser beam) and uses triangulation to determine the distance to an obstacle. In the sensor the angle of the returning light is measured. With that information, the distance to the reflecting surface is calculated. The main problems with IR sensing are that sunlight (or other IR sources) can influence the readings significantly. Second is that performance of the IR sensor depends largely on how the light is reflected. Therefore and IR sensor can return different values depending on surface texture and even color of the object, even if the range is the same.

Sonar A SONAR (standing for SOund Navigation And Ranging) works the similar to a laser range finder, but instead of using a laser beam, sound waves are used. The time of flight is also used to determine the distance to obstacles. Where the infrared beams from IR sensors can be influenced by bright sunlight, the main issue with sonar sensors is the so-called 'ghost echo'. A ghost echo occurs when the sound waves that are emitted bounced of walls back to the sensor. Figure 2.1 demonstrates this effect. Two sonars cannot be used close together for the same reason: the sound waves will interfere with each other. Nevertheless, sonar sensors are usually as sensitive as laser range finders.

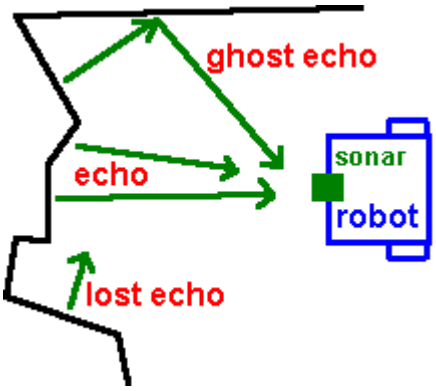


Figure 2.1: *The 'ghost echo' effect. The sound waves bounce of a wall and return to the sensor, providing a false distance reading of an obstacle. This image is taken from* `http://www.societyofrobots.com/member_tutorials/node/71`

Camera Using visual processing techniques, obstacles and their distances can be extrapolated from camera images. There are a lot of different techniques, some of which are discussed in Section 2.3.3. Some of the most used techniques in robotic obstacle avoidance are optic flow [14] and flow field divergence [45]. These methods are used mostly in flying robots, because they are cheap and lightweight in relation to advanced sensors like laser scanners.

Tactile sensors Although bumpers are usually saved as a last resort (since when they are activated a collision has already occurred) they are obstacle detection sensors, albeit with a very short range. Another form of tactile sensors, with a slightly larger range, are antennas or whiskers. The antennas can feel where obstacles are. Although the range and precision is very limited they are cheap sensors to find obstacles.

Besides sensor data, a representation of the environment is vital for efficient navigation. This representation is usually in the form of a map. On the map, stationary obstacles (like walls, pillars and other objects that do not change position) are marked. These obstacles can act as landmarks for the localization module and provide information to the path planners where the robot can definitely not go. Each component of the navigation system uses the sensor data and the map in its own way. For each component the workings are explained below.

2.1.2 Localization

The *localization* component uses the adaptive (or KLD-sampling) Monte Carlo Localization (AMCL) approach, as described in [83]. It is a probabilistic localization approach which combines laser scan and odometry data and uses a particle filter to track the pose of the robot against a known map. Odometry data is information from the wheels about how much they have rotated. From this the displacement of the robot can be calculated. However, since the wheels

slip, this information is not 100% accurate. The laser scan data is therefore combined with the odometry to provide a more accurate location.

2.1.3 Global path planner

The *global path planner* calculates a path from the robot's position to the goal, by using a *cost map*. The cost map is built up from the map of the environment (if provided) and sensory data. It is a grid representing the world around the robot. Each cell in the grid can represent a free, occupied or unknown space. Furthermore, each cell has a cost value assigned. The value is low (close to zero) when sensors detect no obstacles at the position of the cell, and high if an obstacle is detected. The costs are propagated to neighboring cells, decreasing the cost value with increasing distance to the obstacle. This creates a buffer zone, keeping the robot away from obstacles, but which is still traversable if necessary. Once the cost map is created the global path is planned using Dijkstra's algorithm [16]. Dijkstra's algorithm uses breadth-first search to find the optimal path through the cost map, starting at the robot's origin and radiating outward until the goal is reached. The global path is re-planned regularly, though not as frequent as the local path.

2.1.4 Local path planner

Once a global path has been found, it is sent to the *local path planner*. The local path planner calculates the velocity commands to send to the robot using the Dynamic Window Approach (DWA) [26]. The DWA first creates a search space consisting out of velocities the robot can reach within a short time. Those commands are safe such that the robot can stop before it reaches the closest obstacle in that trajectory. For each velocity command, a forward simulation is performed to predict the location of the robot in a short amount of time. Then, the resulting trajectory is evaluated, taking the proximity to obstacles, proximity to the goal, proximity to the global path and speed in to account. Trajectories that result in a collision are automatically disregarded. The highest-scoring trajectory is picked to generate velocity commands from to control the robot. This is done continuously, enabling the robot to follow the global path and locally avoid obstacles appearing in the sensory data.

2.1.5 Error and recovery

Sometimes the local and global path planners are not able to calculate a feasible path. When a path planner perceives itself as stuck, a series of recovery behaviors[2] can be executed to clear out the cost map and enable the planner to find a feasible path. First a *conservative reset* is performed by clearing the obstacles[3] outside a specified region. Next, if possible, the robot performs a *clearing rotation*. The robot rotates in-place and uses sensory data to clear obstacles from the cost map. If this fails too, an *aggressive reset* is performed. This removes

[2] The recovery behaviors are as implemented in the standard move_base stack in ROS at `http://www.ros.org/wiki/move_base#Expected_Robot_Behavior`.

[3] Obstacles are cleared from the cost map when sensory scans return a distance above a certain threshold. If a scan does not return a value, i.e, there are no obstacles within the range of the scanner, the cost map is not cleared.

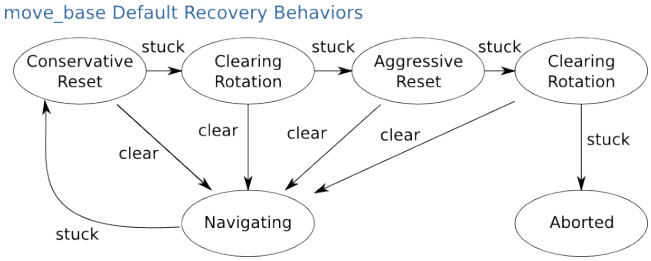


Figure 2.2: *Overview of the recovery behaviors. This image is taken from* `http://www.ros.org/wiki/move_base`.

all the obstacles from the cost map outside of the rectangular area in which the robot can rotate in-place. This is followed by another *clearing rotation*. If the path planner still is not able to plan a path, the goal is *aborted*. Figure 2.2 shows an overviews of the recovery behaviors described above.

Another way to resolve navigation problems is by using interaction as a recovery mechanism. When the robot perceives itself as stuck, it could call for help. A human could than approach the robot and give a command for what it should do. However, since interaction in this thesis is used as a means to predict the user's future position and not as an error recovery mechanism, this is not further explored.

2.1.6 Global structure

The global structure of the complete navigation algorithm is depicted in Figure 2.3. The localization algorithm takes the sensory and odometry data from the robot and a map of the environment and provides an estimated position of the robot in the world. The global planner uses this position together with the map and sensor data to calculate a global path. The local path planner, which also uses the estimated position of the robot and sensor data, takes in the global path and calculates velocity commands to steer the robot towards the goal, while avoiding obstacles locally.

2.2 Related work in people avoidance

Previous work in path planning is summarized in [6]. Most algorithms mentioned in [6] focus on navigation without the presence of moving objects. The biggest challenge for navigation is dealing with moving obstacles, like people. Moving obstacles are also referred to as *dynamic obstacles* whereas obstacles that remain stationary are called *static obstacles*.

There are three main approaches for handling dynamic obstacles [2]:

- Reduce the dynamic obstacle to a static obstacle. The planner does not take any dynamic aspects of the obstacle, like speed and direction, into account. It merely updates its representation of the surrounding space every time step and changes its path accordingly. This approach has been taken by [23, 44]. Since this is the most simple approach the algorithm implementing this is called the *base algorithm* .

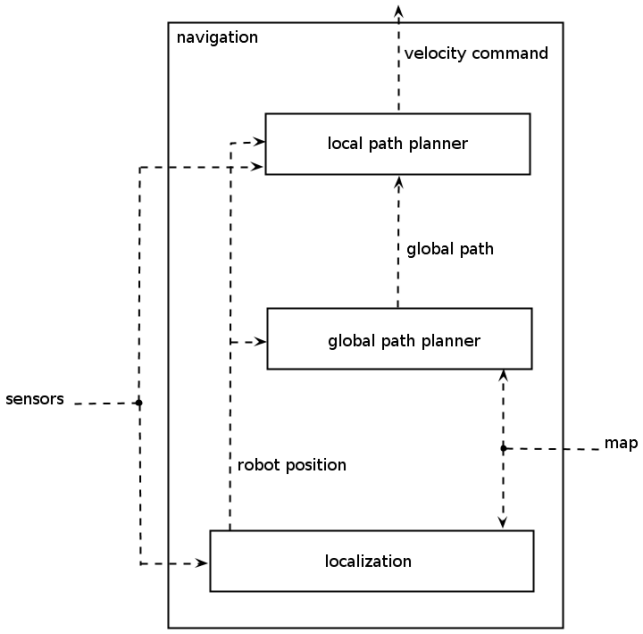


Figure 2.3: *Schematic overview of the components of the navigation system. The localization compares the sensor data to the map and provides a position estimate on the map. The global planner creates a globally optimal path using the map and the robot's estimated position. The local planner takes the sensor data as input and tries to avoid obstacles and maintain close to the global path. The local planner generates the commands to control the robot.*

- Extrapolate the estimated velocity of the obstacle and use this to estimate the future positions of the obstacle. This approach is called *Asteroid Avoidance* (AA) [27]. It is assumed that the movement of the obstacle can be predicted using a linear function. Once the future position is predicted, the path planner can calculate which actions to undertake to prevent a collision while maintaining an efficient path. In [2, 24] this approach was implemented.

- The last approach is named *reciprocal avoidance* [2] or *reflective navigation* [39]. This approach assumes that when an obstacle is moving, it is also able to react to the world and the robot. It can make decisions and therefore it is hard to predict the future position using any linear function. A mental model has to be built to predict the future actions of the other robot. Hennes et al. implemented reciprocal avoidance for multiple robots [33].

The reciprocal avoidance can be extended to be used to avoid humans. According to Sisbot et al. [76] the following points should be taken into consideration for the motion planning of mobile robots:

- Safe motion, i.e., that does not harm the human;

- Reliable and effective motion, i.e, that achieves the task adequately considering the motion capabilities of the robot;

- Socially acceptable motion, i.e, that takes into account a motion model of the human as well as his preferences and needs. This means that the robot should not cut off the human or move through areas the human cannot perceive. An example is given in Figure 2.4, where the cost are displayed for moving in the proximity of a standing and moving human.

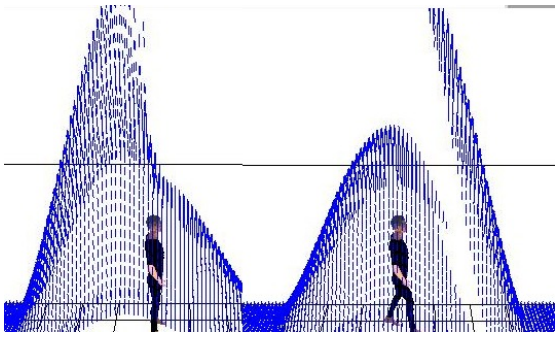


Figure 2.4: *An example of an implementation of socially acceptable motion. The blue lines represent the cost of traveling in the proximity of the human. This figure is taken from [41]. If the human is standing still (left) the area directly behind him has a high cost, because that area is not perceivable. When the human is moving (right) the area directly in front of him should be avoided.*

Socially acceptable motion only provides a very local prediction of the movements of the human. By adding a prediction of a more global path of the human a model of where the robot should and should not travel can be made. This type of dynamic obstacle avoidance will be called *Human Motion Model Avoidance* (HMMA). Such a model can be made by combining the field of proxemics [31,57] with path detection and/or prediction [34, 72]. Furthermore, a decent people detection and tracking algorithm is needed to estimate where the users are and where they are going. There exist many people detection and tracking algorithms already. Many are used on a stationary camera [7] or for use in mobile robots [84]. For interested readers, Dollar et al. [19] provide a good state-of-the-art overview of people detection algorithms.

An example of HMMA is given in Figure 2.5. This approach, as described in [82], assigns high cost directly in front of the human (using proxemics) and slightly lower cost to areas where the human might travel to (using path prediction).

Several groups have tried to implement HMMA in various circumstances. In the work of Sisbot et al. [76] the humans stood still and the robot needed to find a suitable path around them. Svenstrup et al. [79] implemented an algorithm to determine whether or not to approach people for interaction (and do so in a socially accepted manner). Also, Kruse et al. [41] and Lam et al. [42] implemented HMMA successfully in a confined space like a hallway.

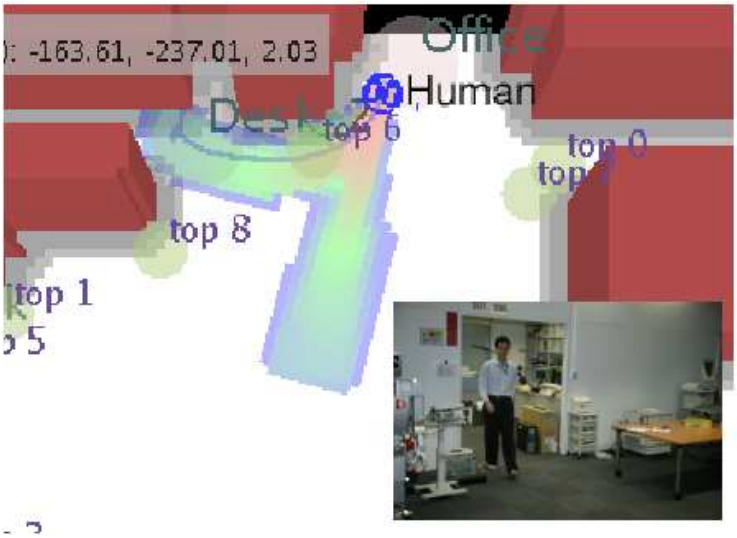


Figure 2.5: *An example of a HMMA implementation. Cost assignment for robot navigation is done by combining proxemics with path prediction. This image is taken from [82].*

2.3 Interactive navigation using gesture recognition

The main problem of AA and HMMA is predicting where the human is going. The earlier this prediction is (accurately) made, the bigger the improvement of the global path is in terms of amount of detour and amount of time needed. Besides using different path detection algorithms that can learn general motion patterns, as described above, cues from the human can be used to predict the path he is going to take. For instance, Prévost et al. [64] showed that humans turn their head before they take a turn in their path. This 'go where we look strategy' can be exploited to solve pre-collision situations. However, these head turning cues are not easy to pick up on using only sensors on a mobile robot. Therefore, by making these interactions more explicit, it can help the robot make decisions on where to pass the person and help the person trust the robot. The interactions can take place using various media, including a touch interface, speech, or gestures. Since performing gestures are closest to body language, arm gestures are chosen as the interaction channel. To classify these arm gestures a gesture recognition algorithm is needed.

One of the first prominent examples of a system which recognizes human arm and hand gestures for controlling an application was Bolt's put-that-there concept [5]. Since then, many more research labs have investigated various aspects of gesture recognition. Gesture recognition has been explored from a wide range of backgrounds, such as computer vision, computer graphics, pen computing, human motion understanding, human computer interaction and human robot interaction. For some excellent surveys on gesture recognition, the reader is referred to, e.g., [32, 48, 49, 93].

The next sections will introduce our approach to gesture recognition. After a brief overview of the state of the art in gesture recognition, we will describe the most prominent elements of a gesture recognition system and focus on the approach that we have taken in our research.

2.3.1 The chalearn gesture recognition challenges

Although many inspiring results have been published, these have mainly been developed and tested in controlled lab situations. Only very recently, demonstrations of gesture recognition technology have become available for the larger public. In particular, the introduction of the Microsoft Kinect and corresponding software development kits has boosted the world-wide interest of companies and universities in gesture recognition [88]. The state-of-the-art performance in gesture recognition technology is currently challenged under the auspices of `chalearn.org`, an organization which frequently hosts challenging benchmarks for the machine learning and pattern recognition communities. Four gesture recognition challenges will be held, the results from the first round have recently been presented at CVPR2012 and are available via `http://gesture.chalearn.org/`. In total 54 teams participated to the first round challenge, which contained more than 30 small data sets comprising between 8-10 gesture classes. The best result of this challenge achieved a 10% error rate, which is still significantly below human performance. The most efficient techniques used sequences of features processed by variants of Dynamic Time Warping and graphical models, in particular Hidden Markov Models and Conditional Random Fields. As we will outline in Section 4.1, one of our gesture recognition techniques uses Dynamic Time Warping.

Each of the data sets was collected by a fixed Kinect camera, which recorded a single user performing upper-body gestures. Data was easily segmentable, as subjects initiated and finished each gesture from a specified resting position.

2.3.2 Our challenge: robust gesture recognition

The development of a robust and usable interactive system employing gesture recognition requires (i) the design of a gesture repertoire which is easy to learn and easy to produce by human subjects and (ii) the design of gesture recognition technology which is capable of distinguishing the different gesture classes with sufficiently high accuracy [89]. For our envisaged context, i.e., gesture recognition for human robot interaction in home environments, the 10% error rate as achieved in the chalearn gesture recognition challenge is not sufficient. To explore the possibility of using gesture-based interaction for collision avoidance, we therefore decided to reduce the number of gesture classes and design a gesture repertoire which is easily distinguishable by AI techniques. As will be outlined in Section 4.1, our studies will explore the feasibility of our concept of interactive collission prevention, by designing three gestures representing, respectively, a *go left*, *go right*, and *stop* command.

2.3.3 Components of gesture recognition systems

A typical gesture recognition system consists out of three main components [91]:

- Sensor
- Feature-extraction (also called front-end)
- Analysis (also called back-end)

Each component is discussed separately.

Many different sensors can be used to track a person's movement. Each type of sensor has its own advantages and disadvantages. Below are some of the most commonly used sensors:

Single camera A single camera is widely used in gesture recognition because it is relatively cheap. No special equipment has to be bought. However, extracting feature information from 2D images is relatively hard and computationally expensive. A lot of research has been done using a single camera for detecting hand gestures [20, 74], pointing gestures [10, 68] and whole body gestures [75]. Some algorithms make use of visual markers to simplify the feature extraction [9]. However, the markers have the disadvantage that the interaction is less spontaneous, because they first need to be applied before the interaction can take place.

Controller By using controllers as an interface, several types of data can be gathered. Examples of controllers are a mouse or pen [89] (acquiring 2D trajectory data) and a Wii Remote[4] [73] (acquiring 3D positions in space, as well as accelerometer data). Wired gloves are also used to record hand movements [37]. Controllers usually record only trajectories (2D or 3D) of one or more points. Because of this, there is no need for intensive preprocessing to acquire trajectories, as is with visual data. However, just like using visual markers, interacting with a controller is less natural and intuitive as interacting without them.

Stereo cameras By combining the images of multiple cameras, and using a known distance relationship between the cameras, a rough depth image can be constructed. A depth image, also known as depth map or 3D image, is an image representing the distance of surfaces to the camera. They act much like how our own visual system uses stereoscopic information to infer the distance to objects. The resulting depth images can be used to simplify feature extraction with relation to using only a single camera [77]. A downside is unfortunately that the calibration of the cameras has to be precise, otherwise the depth images will be distorted. Furthermore, the extra preprocessing step of combining the multiple images into one depth image increases the total processing time significantly.

Depth aware cameras Instead of combining two 2D images, there are other techniques to generate depth data. For instance, time-of-light cameras or structured light cameras (like the Microsoft Kinect sensor). These cameras are able to generate a depth map directly. For instance, the Kinect projects

[4] `http://www.nintendo.com/wii`

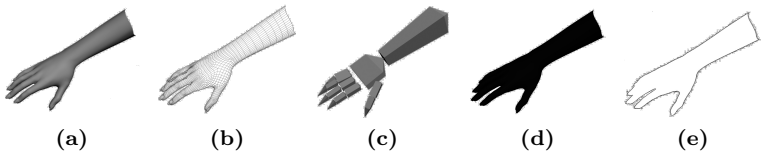


Figure 2.6: *Representation of a human hand (a) by using a volumetric model(b), skeletal model(c), binary silhouette (appearance based)(d) and contour model (appearance-based)(e). The images are taken from* `http://en.wikipedia.org/wiki/Gesture_recognition`.

a grid of infrared dots and by measuring the distortions in the grid, the distance to objects distorting the grid can be calculated. Especially the Kinect is used very often for gesture recognition [4, 67, 94], since it is a relatively cheap sensor with a lot of support software, like OpenNi[5] and Microsoft SDK for Kinect[6].

In this thesis the Kinect will be used for gesture recognition. The Kinect is already widely supported on ROS and is a cheap sensor. Furthermore, as said before, it is used very often for gesture recognition.

2.3.4 Gesture representations and recognition

Depending on the sensor (and the type of data that is gathered) the gestures can be represented using different types of models. Pavlovic et al. [60] classified the models in two classes: *3D models* and *appearance-based models*. However, since *skeletal models* are so widely used nowadays, instead of listing it under 3D models it gets a separate mention.

Figure 2.6 shows for each type of model a representation of a human hand. A good survey that describes approaches to gain the above models is one written by Moeslund et al. [49]. The different types of models are:

3D models A 3D model is a volumetric representation of the user's body. This type of model gives the most detailed representation to use, however, it is computational very expensive. Therefore, 3D models have not yet been used in real-time gesture recognition systems.

Skeletal models Although Pavlovic et al. classify this as a 3D model, skeletal models are mentioned separately. A skeletal model consists not out of a (large) collection of vertices an lines as in volumetric models, but out of a collection of the positions of joints in space. This results in less variables than a volumetric models and thus in faster processing times.

Appearance-based models Instead of using spatial representations, an appearance-based model uses the appearance of the body in the images to extract features. Appearance-based models generally have real time performances because of the use of 2D image features. There are several

[5] `http://openni.org/`
[6] `http://www.microsoft.com/en-us/kinectforwindows/`

approaches like skin color detection, the use of eigenspace or using local invariant features, like AdaBoost learning or Haar like features. However, most appearance-based models are only feasible under specific lighting circumstances and for small gesture sets [28].

Once the relevant features are selected the back-end component is responsible for recognizing the poses (single frame[7]) and gestures (sequence of frames). The most useful features for gesture recognition are usually the positions, velocities and angles of the various relevant body parts. Other features could be the smoothness of the movement (usually it goes the slower the movement, the less smooth it is). In order to recognize a gesture, two things are required:

1. The classification results for the pose on a frame-by-frame basis.
2. The temporal structure of the frames.

The poses can be distinguished by comparing the input frame to the template frame using one of many standard pattern recognition techniques, like cost functions [63], template matching [17], geometric feature classification [3], neural networks [55] or support vector machines [51]. For incorporating the temporal structure of the gestures in the analysis, special techniques have been developed and are also used in handwriting recognition and speech recognition. Examples are dynamic time warping [13], hidden markov models [22] and Bayesian networks [78]. For a detailed survey about gesture recognition see the work of Mitra & Acharya [48].

One of the big challenges in gesture recognition is determining when a gesture is recognized. First of all, there is usually ac continuous input stream of poses. There is no clear indication when a gesture starts and when it ends. The sliding window approach is used to solve this problem. A sliding window is a set of frames with a fixed width around the last received frame. For instance, every new frame, the last x frames are also used in the gesture recognizer. Finally, there are two methods which can be used with a sliding window to determine whether a gesture is recognized.

First is *incremental recognition*. With this technique the frames are fed one by one to the recognizer. After a certain amount of frames the recognizer is forced to classify the gesture. This allows for the recognizer to classify a gesture before it is completed. For this reason it is important that (especially the initial phases of) the gestures are not too similar. An example of incremental recognition is given by [80].

The second method is using *confidence based recognition*. For each stored gesture, a confidence value is kept representing how much the current incoming gesture resembles the templated gesture. Once the confidence rises above a threshold value, that particular gesture is recognized.

In this thesis, a gesture recognition system is incorporated in in the standard navigation system. This will allow users to give commands to the robot to ensure an efficient, collision free path for both parties. The user can indicate with a gesture on which side the robot can pass the user. This mechanism is called ICP. ICP will be compared to the other (dynamic) obstacle avoidance methods in Chapter 4.

[7] By *frame* is meant a single image

Chapter 3

Non-interactive dynamic obstacle avoidance

In this chapter the performance of the non-interactive obstacle avoidance algorithms (the base algorithm, AA and HMMA) is tested. Each algorithm is first described in Section 3.1. The performance is tested in a simulator. Section 3.2 describes the performed experiments. Then, in Section 3.3 the results of those experiments are presented. Finally, Section 3.4 provides a discussion about the results.

3.1 Dynamic obstacle avoidance algorithms

In this section the different avoidance algorithms that are used in this thesis are described. First is the basic navigation, which is taken directly from the ROS open source website[1]. Sections 3.1.3 and 3.1.4 describe two extensions of the basic navigation algorithms for use in environments where people should be avoided.

3.1.1 The base algorithm

The robot in this thesis is designed to navigate in an indoor home-like environment, where it can encounter people. The robot should adapt its navigation plans to disturb those people as minimal as possible. For simplicity, it is assumed that the robot only encounters a maximum of one person at a time. The algorithms can be adapted to deal with multiple people however.

The basic navigation system as explained in Section 2.1 works well in a static world. However in a world with moving objects (for instance people) this approach has its drawbacks. Let's take an example of a person crossing the path of the robot, moving in from the right. If the robot does not take action a collision will occur. This example is illustrated in Figure 3.1. At a certain time t (before the collision) the person comes in the visual field of the robot. The local path planner will mark the person as an obstacle on its cost map. The local planner will re-plan its path a little to the left, because this is the path

[1] `http://www.ros.org/wiki/navigation`

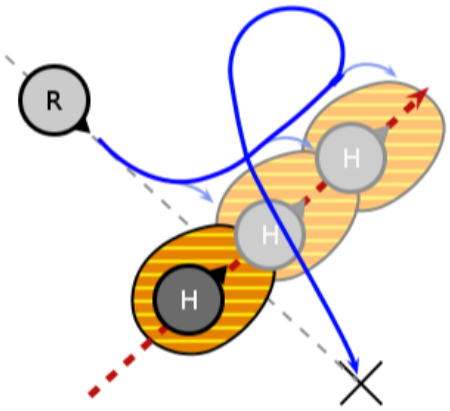


Figure 3.1: *The problem of using the base algorithm with dynamic obstacles. When an obstacle is coming in from the right, the local path planner will re-plan the local path to the left. This way it will keep encountering the person in a* collision loop *until it is shorter to pass the obstacle at the back. However, the robot has already made a substantial detour.*

with the lowest costs. At time $t + 1$ the robot starts to go left, but then the person moves too. This results in another obstacle (i.e, at another location) in the view of the robot. The path is re-planned as before. This will repeat several times, causing the robot to get stuck in a *collision loop* and detour from an efficient path. Eventually, the robot can break out of it, but then a substantial detour has been made. Furthermore, because the person is continuously in the visual field of the robot (like a long object, or wall), the localization module gets confused as to where the robot could be. The localization module may not recover from this, even though the person is already gone. In those cases the goal will not be reached by the robot.

The navigation of robots in a dynamic world can be improved by using a prediction of the movement of objects the robot can encounter. How this can be done is explained in Section 3.1.2.

3.1.2 People aware navigation

As mentioned in the Introduction, two types of non-interactive people aware navigation are investigated. One, called *Asteroid Avoidance*(AA) [66], assumes a linear motion of the person. The other method, *Human Motion Model Avoidance*(HMMA), assumes that people are following (predefined and known) paths. How both algorithms work is explained in Section 3.1.3 and Section 3.1.4 respectively.

Both methods will use the same path planners described above. The difference between the methods is in the data that is used to calculate the cost map for both planners. Remember the algorithm described in Section 2.1 is called the *base algorithm*. In the base algorithm the cost map is built directly from the sensor data. The new algorithms, which implement AA and HMMA, modify these sensor data in such a way, that the people are not displayed at the position they are currently at, but in the position where they are predicted to

be over an δ_t amount of time[2]. The people aware navigation algorithm is shown in Algorithm 2. The cursive lines are the ones that differ from the original algorithm (see Algorithm 1).

Algorithm 2 The people aware navigation module

```
function NAVIGATION
    goal ← main goal
    Plan global path
    while goal not reached do
        Modify sensor data                          ▷ See Algorithm 3
        Read sensor data
        Adapt cost map
        Plan local path using DWA
        if no local path possible then
            Execute recovery behaviors
        else
            Generate velocity commands for local path
            Move robot
        end if
    end while
end function
```

The sensor data is modified in two steps. A pseudo code representation of this sensor data modification can be found in Algorithm 3:

1. First the user is removed from the sensor data. This is done by setting the ranges in the laser scan, which is used for building the cost map, to maximal at the current position of the person (plus and minus a predefined width for the users personal space), unless the scan is reflected from something closer than the person. Also, if the user is really close, the scan data remain as they are. This acts as a failsafe for when the robot comes too close to the user.

2. Then, at the predicted position, the calculated ranges are inserted, again only if the user is further away than the closest object at that position. Note that the predictions include a buffer, as shown in Figure 3.1 by the hatched areas, to account for uncertainty in the predictions. This buffer is also inserted in the sensor data, which in the current implementation is represented by a rectangle with a predefined width and height.

Notice that the proposed algorithms only predict the next position of a person for time t, instead of calculating a trajectory. The DWA which plans the local trajectory, which was explained in Section 2.1 takes only the next time step into account so it only needs the predicted position for the next time step. Therefore it does not need (or can handle for that matter) any trajectory information. This makes the algorithms a fast and computationally inexpensive improvement on the base algorithm.

[2] The amount of time depends on the time it takes for the local path planner to update its cycle, allowing for an accurate plan per time step. However, finding this value is not easy.

Algorithm 3 Modification of the sensor data for people aware navigation

```
function USER AVOIDANCE
    for each time step do
        if user detected then
            user position ← get user position
            REMOVE USER FROM SCAN(user position)
            future position ← predict future position
            INSERT FUTURE POSITION IN SENSOR DATA(future position)
        end if
        plan new local path
    end for
end function

function REMOVE USER FROM SCAN(user_position)
    index ← find index of position in scan (user_position)
    for i = −user_width → user_width do
        if scan[index + i] > close range and
            scan[index + i] ≥ user distance then
            scan[index + i] ← MAX_RANGE
        end if
    end for
end function

function INSERT FUTURE POSITION IN SENSOR DATA(futureposition)
    init new_sensor_data
    for x = −buffer_width → buffer_width do
        new_point.x ← future_position.x + x
        for x = −buffer_width → buffer_width do
            new_point.y ← future_position.y + y
            new_sensor_data.Insert(new_point)
        end for
    end for
    Publish new_sensor_data
end function
```

The relations of the algorithms proposed in this thesis can be caught in the subsumption architecture of Brooks [8]. If there is a moving obstacle, the base algorithm is overridden by the human aware algorithms. If a gesture is received, the ICP will subsume the human aware algorithms. Figure 3.2 shows the proposed subsumption architecture. The implementations of AA and HMMA are further described in Section 3.1.3 and Section 3.1.4. ICP is explained in Section 4.1.

Human motion

In this thesis three categories of human movement are investigated. These categories represent the vast majority of possible human motion patterns in home environments. The categories are:

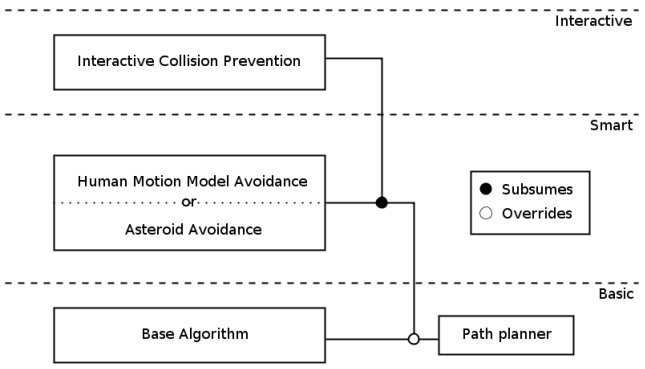


Figure 3.2: *The subsumption architecture of all algorithms. AA and HMMA override the base algorithm (in case of a moving obstacle). ICP in turn subsumes everything but the most basic algorithm. The base algorithm is always active as a failsafe.*

- Users move in a linear fashion.
- Users follow a certain, predefined path.
- Users move about (seemingly) randomly.

Each of these assumptions is discussed below.

By assuming linear motion in users, it means that their movement can be described by a linear function. In this thesis the function describes a straight line (see Section 3.1.3)[3] Although the movement of people is not always a straight line, on short intervals it generally is. People prefer shorter paths over longer ones, and straight paths are shorter than curvy ones. If there is a curve in the path the predictions for that interval will be off. However, as said before, the majority of the path is a straight line. The predictions will therefore be right most of the time.

When there are a lot of obstacles in the navigation space people are forced to follow a curvy path to avoid the obstacles. These movements are hard to predict using a linear function. It can however be predicted using a human motion model. This model has to perform two tasks:

- find the paths the user can follow
- classify which path the user is following currently.

The paths can be found by observing people using cameras placed in the environment [43, 72] or by using a mobile robot [34]. It is also possible to determine the paths by hand, using common sense. An example is given in Figure 3.3. Here a typical living room is displayed with a dining table and a television corner with a couch and coffee table. There are two doorways connecting it to other rooms. By using common sense, it can be assumed that users travel from one door to the other using an as straight as possible path. Furthermore, it is likely the users will go to the television corner and dining table.

[3]The function can in fact be any linear function, however not all functions are reasonable candidates to describe human movements.

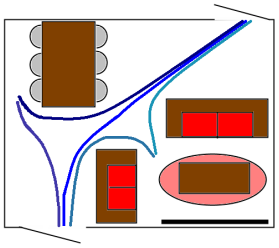


Figure 3.3: *An example of frequently traveled paths in a living room. The dining table in the top left and the TV corner at the bottom right are goals users travel to frequently. The doors are important goals as well.*

Besides detecting possible paths, the human motion model should contain a classifier too to determine which path the user is taking at a certain time. This classifier can be implemented using all kinds of techniques, ranging from fairly simple (using only trajectory information [43]) to complex classifiers. An example of a complex classifier is one that, besides using trajectory information, also uses information from other devices. For instance the user was just now in the kitchen and opened the refrigerator. By using a reasoning system, it can be extrapolated that it is more likely the user will walk to a table than to the bedroom. Furthermore, (user specific) information, like daily routines, can be used as well. This information can be collected in the Smart Homes mentioned in the Introduction and described in more detail in [12]. The human motion model used in this thesis is described in Section 3.1.4.

Sometimes, the path of the user cannot be predicted or the user is not following any path. It may even seem he is just moving about randomly (and maybe he is). Though the movements are unpredictable, the robot should still stay clear of the user to avoid collisions. This is a real challenge for any collision avoidance technique. Most implementations solve this by falling back to a reactive paradigm. If an obstacle is suddenly in front of the robot, it will stop moving at once. All algorithms proposed in this thesis will fall back to this mechanism to prevent collisions.

3.1.3 Asteroid avoidance

As mentioned before, AA assumes a linear motion of people. The algorithm developed in this thesis is inspired on the work of Fulgenzi et al. [27]. The movement of people is predicted by the following function:

$$\begin{aligned}(x_{t+1}, y_{t+1}) &= (x_t + \Delta x, y_t + \Delta y) \quad \text{where} \\ \Delta x &= \frac{1}{n-1} \cdot \sum_{i=1}^{n-1} (x_{i+1} - x_i) \\ \Delta y &= \frac{1}{n-1} \cdot \sum_{i=1}^{n-1} (y_{i+1} - y_i)\end{aligned} \tag{3.1}$$

To predict the position of the user at time $t+1$ two things are needed. First is the position (x, y) of the user at time t. Second is the average movement

in both the x and y directions. n is the number of sample points, so if $n = 2$ the average movement equals the displacement between t and $t - 1$. Since position measurements can be noisy it is better to take n bigger than two. The optimal value of n depends on the frame rate of the measurements, the amount of noise and/or missing frames and the speed of the robot. The bigger n is, the better the algorithm can handle noise. However, the bigger n gets the slower it will react to changes in the users movements. As mentioned before, it can be assumed that the users have acceleration and deceleration constraints. So if n gets too big, the predictions will lag behind the movements of the user. Through experimentation a value of $n = 50$ was found to be optimal given a frame rate of 25 to 30 Hertz.

In cases where wrong predictions are made the recovery time depends on n and the distance between the predicted and actual position. If this distance is small (assuming n remains constant), the robot could recover by adjusting its path the next time step. However, when the predictions are way off, when for instance the person is making a sharp turn, the robot needs to adjust its path drastically. This usually results in a suboptimal path. Nonetheless, AA is already an improvement on the base algorithm because the robot will not get stuck in the collision loop described in Figure 3.1 as easily as the base algorithm.

3.1.4 Human motion model avoidance

HMMA predicts the position of people by using a priori information about the environment and the user. It creates a model of the user's movements. Remember that the model consists of possible paths the user can take and a classifier to determine which path the user is currently following. By providing the path the user is following to the prediction algorithm the new position can be estimated and passed on to the path planner, like in AA.

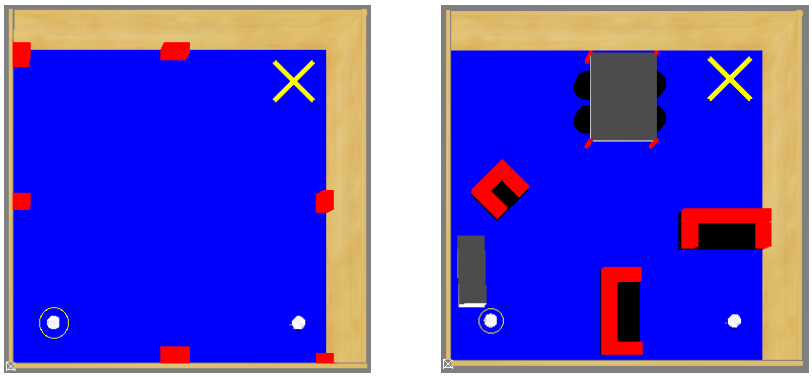
If the wrong path is provided by the classifier the predictions will be wrong and the route the robot will take is suboptimal, no matter how well the path planner performs. To fairly compare HMMA to the other obstacle avoidance methods, it is assumed the path is always classified correctly, making the predictions accurate.

By using paths for predicting the movement of the user, HMMA can handle more complex situations than AA. When the paths users take have many bends and turns the HMMA will outperform AA. However, if the paths are straight, there is no difference between the two algorithms since they both will predict the same future position.

While the predictions of HMMA are more accurate, if the wrong path is provided to the prediction algorithm or if the user is not following any (known) path HMMA has a harder time to recover than AA. This trade-off should be kept in mind in dealing with partially unknown environments, or in environments where the user's paths are not defined.

3.2 Simulation experiments

In this section the experiments used to determine the performance of the three non-interactive obstacle avoidance algorithms are described. First the used simulator and environments are described (Section 3.2.1 and Section 3.2.2).



(a) *Simple environment* **(b)** *Complex environment*

Figure 3.4: *The environments used in the experiments. The simple environment has no obstacles (a). The red squares are landmarks the robot can use for localization. The complex environment is decorated with furniture (b). The robot (marked with a circle) always starts in the left bottom corner. The other robot (bottom right corner) is used to represent a person. The goal is marked with an 'X'.*

Then, in Section 3.2.3 the conditions which represent different circumstances that are tested are specified. Finally the measures to determine the performance of the algorithms are defined in Section 3.2.4.

3.2.1 Simulator

The algorithms are first tested in a simulator. This is common practice in tests involving robots. By running the algorithms in a simulator, there is no chance of physical damage to the robot. Furthermore, the time to complete a run is shortened because there is no set-up or clean-up time and the simulations can run faster than real time.

The simulator used is Gazebo[4] [40]. It is a simulator designed for robotic simulations. It has a physics engine and provides 3D visualization. Furthermore, it is designed such that every algorithm that can run on the robot (via ROS) also can run on the simulated robot and vice versa.

In the simulated environment, the user is represented by another robot. This robot follows preprogrammed paths to mimic the movements of a user in the real world. However, it has no avoidance mechanisms so in case of a collision it will keep trying to move along the path.

3.2.2 Simulated environment

The algorithms are tested in two different environments. An *simple* and a *complex* environment. Both types are depicted in Figure 3.4.

In the simulator the simple environment (Figure 3.4a) is nothing more than an empty room. There is a lot of space for the robot to escape to in case of

[4]More information about Gazebo can be found at `http://www.ros.org/wiki/simulator_gazebo`

(possible) collisions. The red boxes are landmarks and are purely added so the robot can locate itself in the room. In the simulated complex environment (Figure 3.4b) furniture is added such that it resembles a typical living room. There is not much space for the robot to avoid the user so the obstacle avoidance algorithms are really challenged in this environment.

3.2.3 Conditions

The conditions in which the algorithms are tested represent different circumstances in which a good system should be able to perform. Every condition has the same global path. The robot has to travel 8 meters (straight line distance) to its goal. The start and goal position of the robot are the same in every condition. The global path planner can handle every obstacle that is already on the map. Since every algorithm uses the same global path planner, the only interesting conditions are the ones with obstacles which are not on the map. To simplify testing, the global path is a straight line, i.e, there are no obstacles directly between the start and goal positions, except for the ones introduced in the conditions. The conditions range from simple (no obstacles) to very complex (a person turning to the robot). The different conditions are as follows:

- [`NONE`] No obstacles.
- [`STAT`] A static obstacle.
- [`DYNx`] A moving person.
  - `1` A person crossing the path of the robot.
  - `2` A person coming right at the robot.
  - `3` A person turning to the robot.

For each condition a possible (desired) path is displayed in Figure 3.5.

No obstacles (`NONE`)

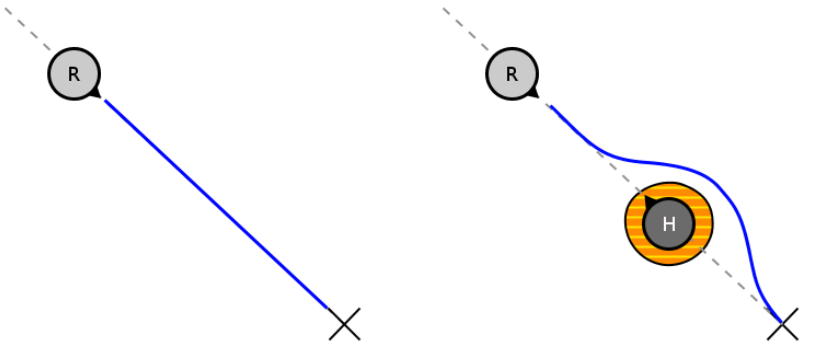
This is the most basic and simple scenario. In this condition there are no obstacles in the path of the robot, so the robot never has to adjust it (Figure 3.5a). All algorithms should be able to handle this. This condition is performed to establish a baseline comparison between the real world and the simulator.

A static obstacle (`STAT`)

In this condition a static (not moving) obstacle is placed in the path of the robot (Figure 3.5b). Since this object is not on the map, the robot has to avoid this obstacle using its local path planner. Again, all algorithms should perform well since the basic local path planner is designed to handle this situation well.

A moving person (`DYN_`)

There are several ways for a person to move into the planned path of the robot. He can either cross the path of the robot (`DYN1`, Figure 3.5c) or come directly at the robot (`DYN2`, Figure 3.5d). Also, the human can change his direction in the middle of the path (`DYN3`, Figure 3.5e). The person will move at a speed



(a) *No obstacles (`NONE`)*

(b) *Static obstacle (`STAT`)*

(c) *Person crossing the path of the robot (`DYN1`)*

(d) *Person coming right at the robot (`DYN2`)*

(e) *Person turning to the robot (`DYN3`)*

Figure 3.5: *The desired path of the robot in different conditions (in blue). The circle with R represents the robot, the one with the H the human. The small triangles indicate the heading of the robot and the human. The highlighted area around the human is the buffer area which the robot should avoid as well.*

of 0.3 m/s which resembles a slow walking pace. The robot can move slightly faster (maximum speed is 0.5 m/s) so it can catch up and overtake the human if necessary.

3.2.4 Measurements

The performance of the algorithms is measured in several ways. To evaluate the algorithms the following measures are used:

- number of successful trials
- time to get to the goal
- amount of detour (traveled distance - straight line distance)

A trial is one attempt of the robot to reach the goal. If the path planner cannot plan a path (because it is blocked by obstacles) three times, the trial ends as a failed trial. Furthermore if the robot has not reached its goal within three minutes (in the `NONE` condition navigation takes approximately 15-20 seconds) the trail fails as well.

One could also count the number of (near) collisions within a trial. However, it is hard to determine when one (near) collision ends and the following begins. Furthermore the amount of detour and time will reflect the number of near collisions. If there are many near collisions the robot needs to adjust its path often, or wait until the path is clear again. So either time or detour will increase with an increasing number of near collisions. Furthermore, if there is a full collision, the robot gets stuck and will not reach the goal. The number of failed trials will therefore reflect the number of full collisions.

3.3 Simulation results

In the simulation experiments the three non-interactive algorithms (the base algorithm, AA and HMMA) are compared on robustness and efficiency. The robustness was measured by counting the number of successful trials, the efficiency by measuring the amount of time it takes for the robot to reach the goal and the amount of detour it takes. First in Section 3.3.1 the results of the simulations in the simple environment are described. Then in Section 3.3.2.

3.3.1 Simple environment

The simple environment in the simulator is just an empty room, shown in Figure 3.4a. The robot has a lot of space to escape to for avoiding a collision. Below are the results of the experiments run in this environment.

Number of successes

Besides the comparison of **time** and **detour** on each of the conditions, the number of successful trials in navigation were counted. The trial was successful if the robot reached the goal within three minutes. Otherwise it was marked as a failure. Each combination of algorithm and condition was run 20 times. Figure 3.6 shows the trajectories of the robot in the first ten trials of each algorithm tested in the simulation experiments in each condition.

Most trajectories are very similar. The variation in the behavior of the robot is very small. In the conditions where the robot collides with the obstacle the variation is larger. However, 20 runs gives a good reflection of the performance of the algorithms, especially in collision free situations.

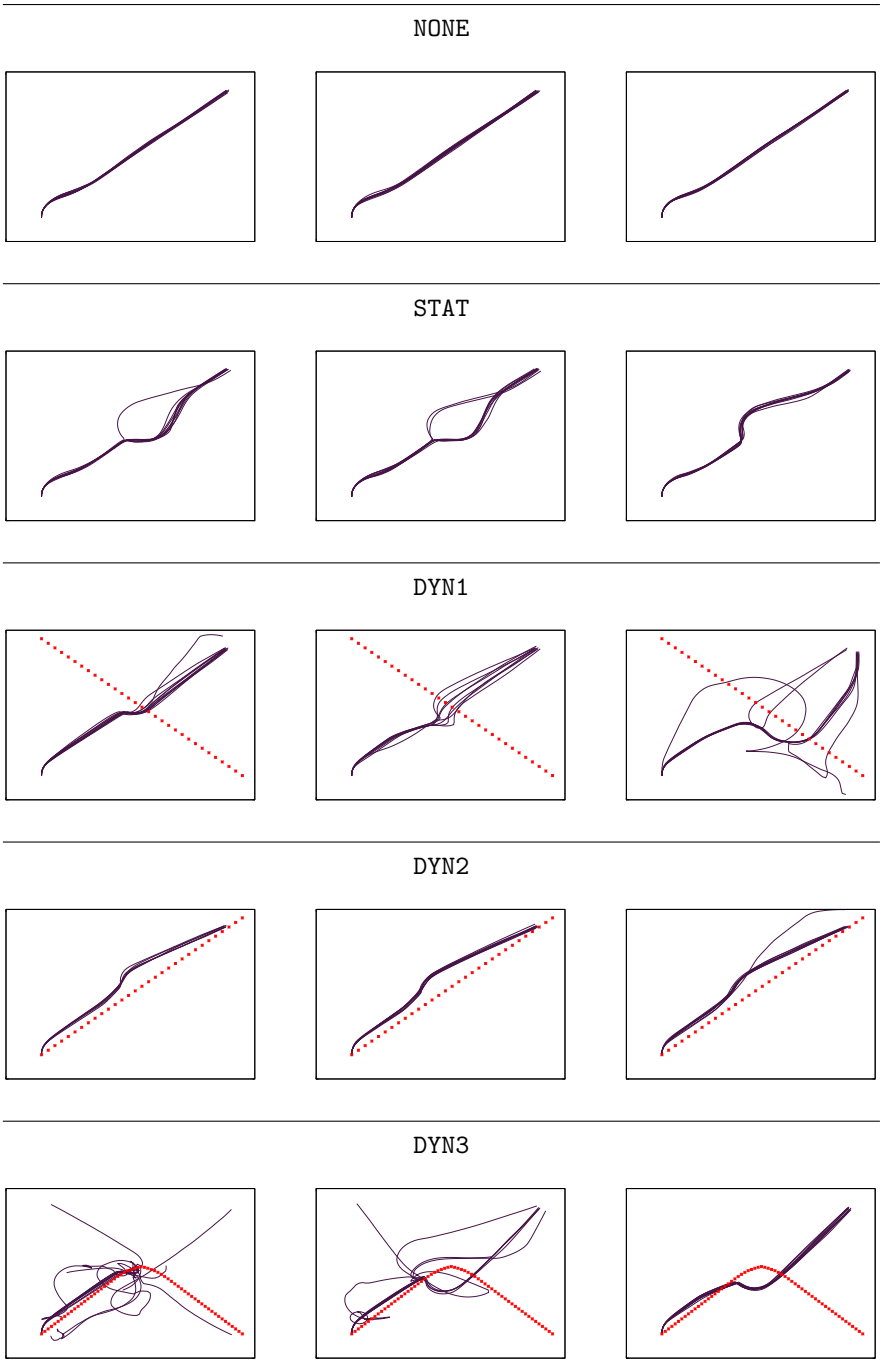


Figure 3.6: *Trajectories of the robot in the simulations in the simple environment of the various algorithms for each condition. From left to right are the base algorithm, AA and HMMA. The robot had to travel from (1,1) to (6.65, 6.65). Each line represents one trial, the red dotted line represents trajectory of the dynamic obstacle.*

Condition	BASE	AA	HMA	Average
NONE	100	100	100	100
STAT	100	95	100	98
DYN1	95	85	95	92
DYN2	80	85	95	87
DYN3	5	60	95	53
Total	60	77	95	77

Table 3.1: *The percentage of successful trials per algorithm per condition. The total score of the algorithms represents the percentage of successful trials in the DYNx conditions.*

Table 3.1 shows for each condition the percentage of trials that were completed successfully.

For the NONE and STAT conditions, every algorithm managed to find the goal (almost) every trial. The differences between the algorithms start to show in the scenarios with moving obstacles. The HMMA had an almost perfect success rate (19 out of 20 successes, 95%) for all three dynamic obstacle conditions. AA performed on the DYN1 and DYN2 conditions 85% of the trials successfully. However, the success rate dropped for the DYN3 condition to 60%. The same drop, but more drastic, can be found for the base algorithm. The base algorithm had a success rate of 95% and 80% respectively on the DYN1 and DYN2 conditions. However, in the DYN3 condition, in only one out of the 20 trials (5%) the robot reached its goal.

On the DYNx conditions, the HMMA had a higher success rate (95%) than AA and the base algorithm. The base algorithm performed worst, with a success rate of only 60%, AA follows with 77%.

Next, the data was analyzed to test which obstacle avoidance algorithm performed best in terms of amount of time and amount of detour. The means and standard deviations are plotted in Figure 3.7. First the performances of the algorithms were compared to each other by performing a one-way ANOVA test on **time** and **detour**. This was done for each condition separately. Then the ANOVA was performed on all conditions combined.

Condition: NONE

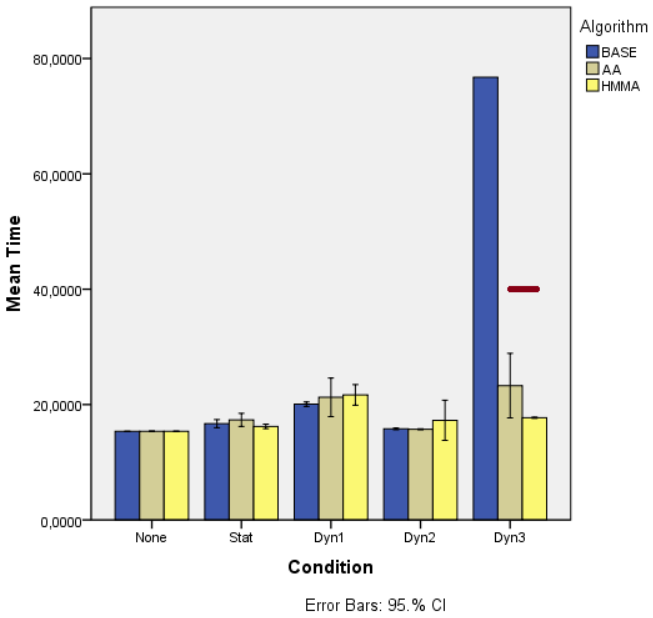
Both the differences in **time** and **detour** were not significant across the three algorithms, $F(2,57) = .515$, $p = .600$ respectively $F(2,57) = .066$, $p = .937$.

Condition: STAT

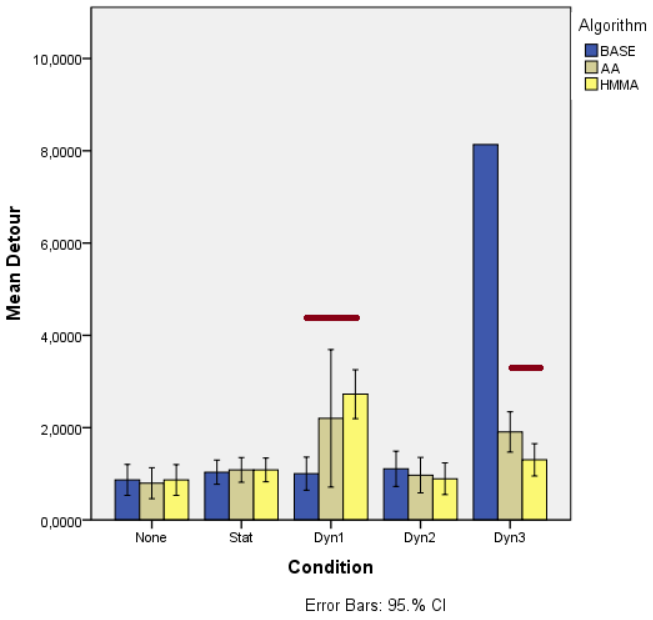
In the STAT condition, both the differences in **time** and **detour** were not significant across the three algorithms, $F(2,56) = 2.167$, $p = .124$ respectively $F(2,56)=.053$, $p = .948$.

Condition: DYN1

The difference in **time** was not significant in the DYN1 condition, $F(2, 52) = .734$, $p = .485$, but the difference in **detour** was, $F(2, 52)=4.625$, $p = .014$.



(a) *Time*



(b) *Detour*

Figure 3.7: *Plots of the simulation results in the simple environment. (a) shows the average time and (b) the average detour. The red lines above the bars indicate significant differences between the algorithms.*

Condition	Time	Detour
NONE	F(2,57)= 0.515 p = .600	F(2,57)= 0.066 p=.937
STAT	F(2,56)= 2.167 p= .124	F(2,56)=0.053 p=.948
DYN1	F(2,52)= 0.734 p=.485	F(2,52)=4.625 p=.014*
DYN2	F(2,49)= 0.723 p=.490	F(2,49)=0.391 p=.679
DYN3	F(2,29)= 57.22 p=.000*	F(2,29)= 44.41 .000*
ALL	F(2,255)= 0.240 p=.787	F(2,255)= 1.239 p=.292

Table 3.2: *The significance results of the one-way ANOVA comparisons of the measured **time** and **detour** in the simulation experiments. Results marked with (*) are significant.*

Tukey post-hoc comparisons show that HMMA (M= 2.72) made a significantly larger detour than the base algorithm (M= 1.00). The comparisons between AA (M= 2.20) and the other two algorithms were not statistically significant at $p = .05$.

Condition: DYN2

In the DYN2 condition, there was no significant difference for either **time**, $F(2, 49)$=.723, p =.490, or **detour**, $F(2, 49)$=.319, p = .679 between the algorithms.

Condition: DYN3

Both the differences in **time** and **detour** were significant across the three algorithms, $F(2,29) = 57.219$, p = .000 resp. $F(2,29) = 44.411$, p = .000. Since there were not enough completed trials of the base algorithm, a Tukey post-hoc comparison could not be made. However, a t-test to compare the AA and HMMA algorithms was performed. On the variable **time** the t-test showed that HMMA (M=17.71) was significantly faster than AA (M=23.28), with p=.009. Also, the amount of **detour** HMMA (M=1.30) took was significantly less than the amount of detour AA (M=1.91) took, with p=.028.

All conditions combined

To measure the overall performance, the data of all conditions were combined and a one-way ANOVA was used on the same variables (time and detour). Both the differences in **time**, $F(2,255)$=.240, p=.787, and **detour**, F=1.239, p=.292, were not significant.

Condition	BASE	AA	HMMA	Average
DYN1	90	85	95	90
DYN2	95	95	80	90
DYN3	20	90	95	68
Total	68	90	90	83

Table 3.3: *The percentage of successful trials per algorithm per condition in the hard environment.*

3.3.2 Complex environment

The complex environment in the simulator is an example of a typical living room. There is a television corner, a reading section and a big dining table. Figure 3.4 shows the used world. The robot has not a lot of space to escape to for avoiding a collision. Because in the NONE and STAT conditions the limited space has no influence on the performance of the algorithms, only the DYNx conditions were tested. Below are the results of the experiments run in this environment.

Number of successes

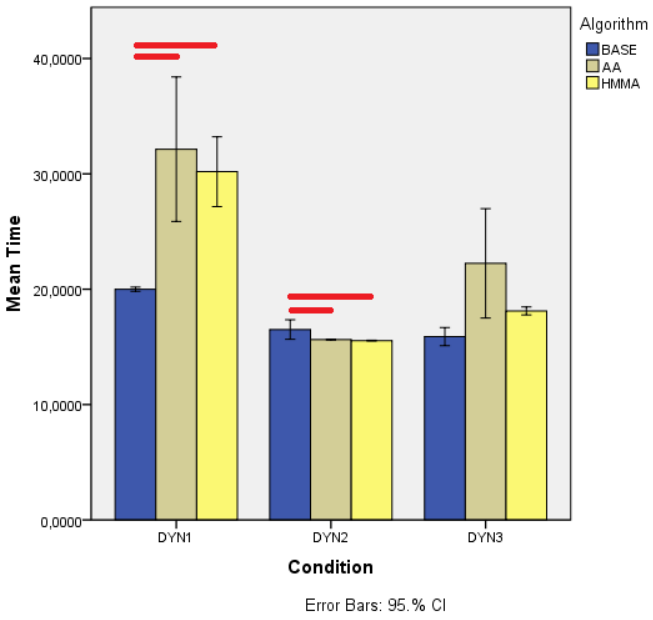
In the hard environment, the number of successful trials in navigation were also counted. Again the trial was successful if the robot reached the goal within three minutes. Otherwise it was marked as a failure. Each combination of algorithm and condition was again run 20 times. Table 3.3 shows for each condition the percentage of trials that were completed successfully.

The robustness of the algorithms in the hard environment is similar to their robustness in the easy environment. The most robust is again the HMMA algorithm. The worst is again the base algorithm, because the performance drops significantly in the DYN3 condition. The main difference is here that the AA algorithm is about as robust as the HMMA algorithm. In the easy environment, the percentage of completed trials in the DYN3 condition was 60%. Here this percentage lies at 90%.

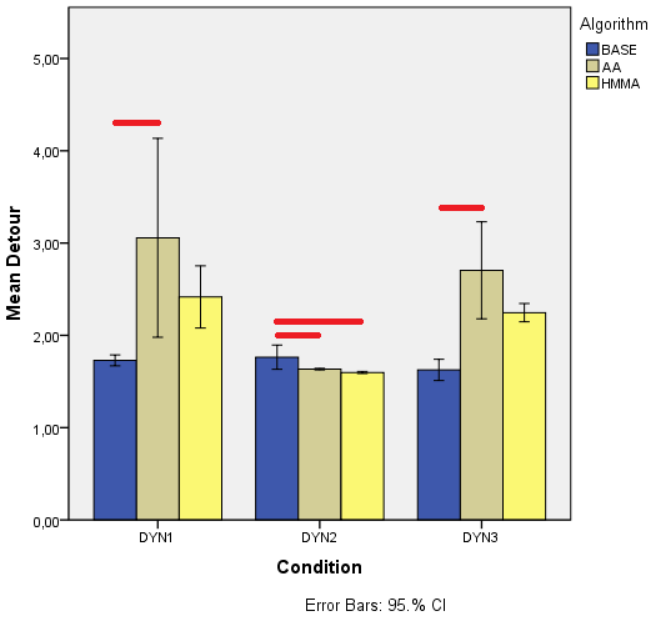
The data was analyzed also using a one-way ANOVA test to compare the difference in **time** and **detour** between the algorithms. Figure 3.8 shows the means and standard deviations of the measured data. The one-way ANOVA tests are performed for each condition separately.

The difference in **time** between the algorithms was significant in only the DYN1 and DYN2 conditions. Table 3.4 shows all the significance results. Tukey post-hoc comparisons showed that in the DYN1 condition, the base algorithm (M=20.00) was significantly faster at the goal position than both the AA (M=32.13) and HMMA (M=30.19) algorithms. The opposite is true in the DYN2 condition. Here post-hoc comparisons showed that the base algorithm (M=16.51) was significantly slower than the AA (M=15.63) and HMMA (M=15.54) algorithms.

For all DYNx conditions the difference in **detour** between the algorithms is significant. Tukey post-hoc tests show that in both the DYN1 and DYN3 conditions, the base algorithm (M=1.73 and M=1.63 respectively)) needed less



(a) *Time*



(b) *Detour*

Figure 3.8: *Plots of the simulation results of the trials done in the hard environment. (a) shows the average time and (b) the average detour. The red lines above the bars indicate significant differences between the algorithms.*

Condition	Time	Detour
DYN1	F(2,51)= 12.508 p=.000*	F(2,51)=4.957 p=.011*
DYN2	F(2,53)=5.031 p=.010*	F(2,53)=5.574 p=.006*
DYN3	F(2,38)= 2.717 p=.079	F(2,38)= 4.332 .020*

Table 3.4: *The significance results of the one-way ANOVA comparisons of the measured **time** and **detour** of the trials run in the hard environment in the simulation experiments. Results marked with (*) are significant.*

detour than the AA algorithm (M=3.06 and M=2.70 respectively). For the DYN2 condition was found that the base algorithm (M=1.76) was slower than both AA (M=1.63) and HMMA (M=1.60).

3.4 Discussion of the simulation results

First the robustness of the three non-interactive algorithms (the base algorithm, AA and HMMA) is discussed. This is measured by the percentage of completed trials over each condition. Then, the difference in efficiency (measured by both time and detour) is discussed. Finally, the overall performance of the algorithms is discussed. This is done for both the simulations in the simple and the complex environment.

3.4.1 Simple environment

As described in Section 3.2.2 the simple environment in the simulations was an empty room where no obstacles, except for the one introduced in the different conditions, were present. The robot had a lot of space to escape to if it needed to avoid the dynamic obstacle. Below the results of the simulation are discussed.

Robustness

The HMMA algorithm is the most robust over all conditions. It fails only one out of twenty trials. The reason for those failures was because the moving obstacle collided with the robot. The robot detected the moving obstacle and paused for a short amount of time to calculate a new path. Since the moving obstacle has no obstacle avoidance mechanism, it ran into the robot. In the real world, these collisions will most likely not occur, because humans (and other moving obstacles) do posses an avoidance mechanism. However, it has to be noted that the robot did not clear the path of the obstacle in time. Earlier detection of the obstacle or faster processing time can solve this problem. Nevertheless, in most trials the robot did not clear the path in time.

The worst performing algorithm was thebase algorithm. It performed very bad on the DYN3 condition. This is again because the moving obstacle collides with the robot. Because the base algorithm cannot predict where the obstacle is going it remains in the path of the obstacle. When it finally notices the obstacle

coming its way, the robot has not enough time to move out of the way. AA and HMMA can make a prediction and therefore move out of the way in an earlier stage. Furthermore, the prediction of HMMA is more accurate (especially in the `DYN3` condition) and is therefore more robust than AA. These results are in line with the results found by Svenstrup et al. [79]. They found that their version of HMMA could correctly identify when and how to avoid a human. The results of the experiments conducted in this thesis conclude that both AA and HMMA can handle the situations they are designed for, while the base algorithm is not very well equipped to handle any situation containing moving obstacles.

Efficiency

First of all, in the conditions `NONE` and `STAT` no significant difference in efficiency between the algorithms is present, for both **time** and **detour**. This is as expected since the ability to predict the obstacle's future position is superfluous in these conditions, because the obstacle is absent or stationary. The baseline performance of all three algorithms is thus similar.

For the efficiency measured by the **time** it took to reach the goal, in the `DYN1` and `DYN2` conditions there is no significant difference. This can have multiple causes. The first possibility is that the distance between the start and goal positions was too short to yield a significant result. However, since navigation in indoor home-like environments is usually short distance, testing the algorithms on larger distances is not recommended. A second explanation for the lack of significant results could be that the robot and the moving obstacle are in each others' paths only a short amount of time. In the `DYN1` condition especially, the robot could wait a short amount of time and let the moving obstacle pass before it. The HMMA algorithm predicted where the obstacle would be and planned a path around it rather than to wait. Therefore, the amount of detour HMMA took is larger than the amount of detour the base algorithm and AA took. With the last algorithm the basic layer (see Figure 3.2) overrode the prediction layers because the obstacle was too close. Fine tuning the parameters and sensors could allow for earlier detection of the obstacle, such that the basic layer does not inhibit the predictions. This could result in a more time efficient navigation algorithm than the base algorithm.

The only condition that yielded significant differences in time was the `DYN3` condition. Very clear distinctions between the algorithms arise. The average time of the base algorithm[5] in this condition is four times larger than in the other conditions. This is because the robot collided with the moving obstacle. This made the localization lose the (correct) estimated location of the robot. This happened in all trials of the base algorithm but one. In that trial the robot came (by accident) close to the goal. However, before the goal was reached, the robot's position was re-estimated (incorrectly) several times. This costs relatively much time, since the robot needs to re-plan its global path with every new location estimation. The collisions also happened with AA, although less often. The average navigation time lies therefore between the average time of the base algorithm and the average time of HMMA. The last algorithm managed to avoid a collision in almost all cases. The HMMA did not have to go through the time consuming behaviors for location estimation and path re-planning.

[5] Actually, only one trial ended successfully. Therefore, the average time equals the time of that one trial.

Kruse et al. [41] found similar results. They found that their version of HMMA performed more efficient than a basic obstacle avoidance algorithm. Furthermore, Fulgenzi et al. [27] found that their AA algorithm performed well in partially occluded environments, although in some cases collision situations could have been avoided.

Overall

As expected no significant differences can be found between the algorithms in the two simplest conditions (`NONE` and `STAT`). Over all other conditions, the base algorithm performs least consistent. The complexer the condition[6], the more (almost) collisions occur. AA performs more consistent than base algorithm. It has less collisions than the base algorithm and is more efficient than the base algorithm. However, it still has some trouble handling the most complex condition (`DYN3`). HMMA is most consistent over the conditions. It performs relatively well in all conditions. However, it seems that the HMMA algorithm takes an unnecessarily large detour in the `DYN1` condition. Nevertheless, since the time is not significantly different, this might suggest the taken route is more fluent and perhaps more human like.

All in all, the HMMA is most robust and thus most efficient over all conditions combined. Then follows AA, which is in turn followed by the worst performing algorithm, the base algorithm.

3.4.2 Complex environment

The complex environment in the simulations resembled a typical living room. The furniture present in the room reduced the available space the robot can move in to when it has to avoid the dynamic obstacle. Figure 3.4b shows the setup of the room. The results of these simulations are discussed below. A comparison with the results obtained in the simulations in the simple environment is made as well.

Robustness

The robustness results in the complex environment are about the same as in the simple environment. All algorithms perform well in the `DYN1` and `DYN2` conditions. The main differences again start to show in the `DYN3` condition. The base algorithm fails in the majority of the trials, whereas the AA and HMMA algorithms succeed in almost all trials.

A striking difference with the robustness of the algorithms is that the number of successful trials is higher in the complex environment than in the simpl environment. The most logical explanation is that the localization is easier in the complex environment, simply because there are more landmarks present. The robot can recover more easily from a collision and therefore reaches its goal more often. The number of collision is therefore not very well reflected by this measure in the complex environment.

[6] `DYN3` is complexer than `DYN2`, which is complexer than `DYN1`.

Efficiency

The efficiency of the algorithms was again measured by comparing the amount of **detour** and **time** the robot needed to avoid the dynamic obstacle and reach the goal. Each condition is discussed separately.

First is the `DYN1` condition. In both **time** and **detour**, the base algorithm was the most efficient. This is reflected in the trajectories of the robot shown in Figure 3.9. The base algorithm did not deviate much from the global path, compared to the AA and HMMA algorithms. The last two algorithms tried to avoid the obstacle in advance. Because of the furniture, these algorithms get stuck locally for a moment and need to find their way back to the global path. Since the only way back is the way the robot came this results in inefficient paths. The base algorithm performs best here because it stops and 'waits' for the obstacle to pass. It reacts to the fact that the obstacle is suddenly close and stops. The robot remains out of the path of the obstacle and therefore a collision is prevented. In a cluttered environment like the one simulated here, waiting might be the best option. The AA and HMMA algorithms could therefore be improved by incorporating a decision mechanism so the robot can decide to wait instead of moving into an area that is hard to get out of and waste time and resources.

In the `DYN2` condition, the results are the other way around. Here the base algorithm performs worse than the AA and HMMA algorithms. Both AA and HMMA deviate very little from the global path. The difference is that the base algorithm makes larger **detours**. These detours appear at three different locations[7] along the global path, in three different trials which are not at the collision site. The only possible explanation for this is that the stochastic nature of the simulator caused the two robots to encounter each other at different areas than the intended collision area. Nevertheless, the amount of detour (and consequently time) taken by the base algorithm is still bigger because the robot did not efficiently avoid the dynamic obstacle.

For the `DYN3` condition, only the difference in **detour** was significant[8]. The base algorithm again needs the least amount of detour. However, since only the completed trials are taken into account in these results, the outcome is skewed. Figure 3.9 shows that the trajectories of the robot using the base algorithm deviate very much from the global path. These are the cases where a collision occurred and the robot lost its position in the room. When these unsuccessful trials would be taken into account, the results would be different and the HMMA would probably come out as the best.

Overall

The overall performance of the algorithms in the complex environment is similar to the overall performance of the algorithms in the simple environment. The HMMA algorithm is better capable to avoid moving obstacles than the base algorithm and in the `DYN3` condition better than AA. However, the predictions by AA and HMMA can cause the robot to move into areas which are hard to get out of. In these cases the robot should either escape to another area when

[7] Except for the one in the collision area, which is intended to happen.
[8] The difference in **time** was almost significant

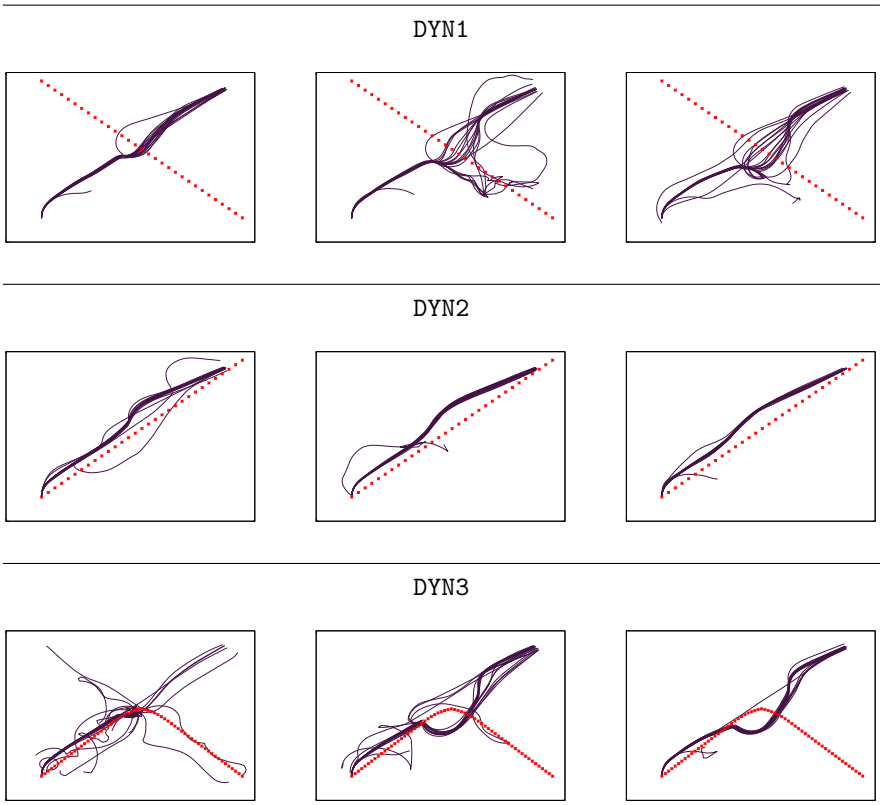


Figure 3.9: *Trajectories of the robot in the simulations in the complex environment of the various algorithms for each condition. From left to right are the base algorithm, AA and HMMA. The robot had to travel from (1,1) to (6.65, 6.65). Each line represents one trial, the red dotted line represents trajectory of the dynamic obstacle.*

an obstacle is coming its way, or wait until it has passed. Future research could explore the possibilities for those approaches.

Chapter 4

Interactive collision prevention

The algorithms are also tested in the real world, in a Wizard of Oz experiment to determine the usability of gesture based interaction in ICP. Section 4.2 describes this experiment. Then, in Section 4.3 the results of the tests are presented. Finally, Section 4.4 provides a discussion about the results.

4.1 Interactive collision prevention with gesture recognition

When there are no paths present (or detected) and the users are not moving in a (predictable) linear fashion none of the proposed obstacle avoidance algorithms perform efficient. Our assumption is that in such cases ICP can be a good solution. In order for ICP to work, a body language interpreter would be ideal. As mentioned in the Introduction, people signal subtly were they are going to prevent collisions with others. However, this type of body language is hard to detect, especially when both the users and the robot are moving. Moreover, as the results of the chalearn gesture recognition challenge indicate, only by using a fixed camera position and non-moving subjects, reasonable recognition accuracies are achieved.

Therefore, in this thesis explicit gestures produced by "stationary" subjects are used to control the robot. If and when a suitable person tracker and body language interpreter are available, a more natural interaction with moving subjects can be incorporated. Because of this, the incoming laser scan data is *not* modified as is in the AA and HMMA algorithms. There is no need to predict the future position of the user and to remove the user at the current position. The avoidance algorithm is thus the same as in the base algorithm. The difference is in the ability of the robot to understand the gestures of the users. Algorithm 4 shows the modified algorithm for navigation. The cursive lines are the lines that differ from the original algorithm as described in Algorithm 1.

Algorithm 4 The modified navigation algorithm for ICP

```
function NAVIGATION
    goal ← main goal
    Plan global path
    while goal not reached do
        if user close then
            STOP and WAIT for command
            if command is STOP then WAIT for 5 seconds
            else if command is LEFT or RIGHT then
                goal ← new way point
                Plan global path
            end if
        end if
        Read sensor data
        Adapt cost map
        Plan local path using DWA
        if no local path possible then
            Execute recovery behaviors
        else
            Generate velocity commands for local path
            Move robot
        end if
        if close to way point then
            goal ← main goal
            Plan global path
        end if
    end while
end function
```

4.1.1 ICP and gesture controls

We have designed three gesture categories as described below in Section 4.1.1. How these three gesture classes are incorporated in our ICP framework is also described in that section. Furthermore, two gesture recognition algorithms are implemented. The first method is trajectory based gesture recognition, which is a form of incremental gesture recognition that compares the current trajectory frame by frame to the templated trajectory. Section 5.1.2 describes the algorithm in more detail. The second method used is offline feature based gesture recognition. This algorithm is described in Section 5.1.3.

The robot starts by navigating from the start point to the specified goal point. When the robot comes close to a user, it stops and waits for a command from the user. To explore the feasibility of our approach, three gesture classes are defined: `LEFT, RIGHT` and `STOP`. If the `STOP` gesture is performed, the robot will remain at its position for a short, predefined period of time. In this thesis this delay is set to five seconds. This allows for the user to pass the robot as he or she finds appropriate. After five seconds the robot resumes its path. One can adapt the behavior of the robot such that it waits an t amount of time after the user has gone out of the collision zone. Setting such a parameter is a typical problem of shared autonomy, which determines when the robot is allowed to take

over control. Shared autonomy in mobile robots is a research field on its own. Examples of research in shared autonomy are provided in (semi)autonomous wheelchairs [15], teleoperation [38] and interactive tasks [61]. Issues on shared autonomy are not explored in this thesis, but are highlighted as possible future research topics in the concluding chapter.

The `LEFT` or `RIGHT` commands indicate at which side of the user the robot should pass. The speed of the gesture determines at which distance the robot should pass. Figure 4.1 shows the way points for each command. The robot should always keep a minimum distance d_{min} to the user. The field of proxemics indicates that humans prefer a larger distance in front and back of them than to the sides. Therefore, the minimal distance indication is shaped like a ellipse. Depending on the speed of the performed gesture, the way point may be positioned further away from the user. In our data collection procedure, subjects will be asked to perform a gesture in one out of three different speeds: `SLOW, NORMAL` and `FAST`. The intuition behind this approach is that the slower the user produces the gesture, the more relaxed he or she is. Therefore, the robot can pass the user using a smaller distance. This distance is the minimum distance to the user (d_{min}). On the other hand, if the user is hasty or stressed, the gestures are performed faster and the robot should keep a larger distance. If the gesture is performed at a normal pace, a distance d_{add} will be added to the minimum distance. The minimum distance is increased again with d_{add} if the gesture is performed fast. The robot has to pass through the way point before it can continue its path to the goal. The same path planners are used to calculate the path to the way point as to calculate a path to the goal.

A performed gesture, which is classified in a class with a certain speed, is called a *command*. A total of seven commands can be given to the robot: both the `LEFT` and `RIGHT` gestures can be performed at three different speeds (`SLOW, NORMAL` and `FAST`), and the `STOP` command.

The `LEFT` and `RIGHT` gestures are the minimal amount of gesture types required for steering the robot off from a possible point of collision. The `STOP` command is necessary if no escape route is possible and the best option for the robot is to wait. The `LEFT` and `RIGHT` gestures are added to ensure the efficiency of the robot. If only one directive gesture (`LEFT` or `RIGHT`) is chosen, the user cannot intuitively control the robot. He has to remember which of the two is available. When more than the three proposed gestures are available for classification, it is harder to find a robust gesture set combined with a robust classifier.

If one of these gestures is performed, a new (temporary) navigation goal is given to the robot, called a way point. The direction of the way point will correspond to the indicated left or right direction of the produced gesture. As will be explored in the subsequent chapters, users may produce different variants of a `LEFT` and `RIGHT` gesture. Typically the trajectory shape, velocity, and amplitude are varied [62] when producing human movements in different conditions. One of our explorations will consider whether the speed of movement may be used as an additional control parameter.

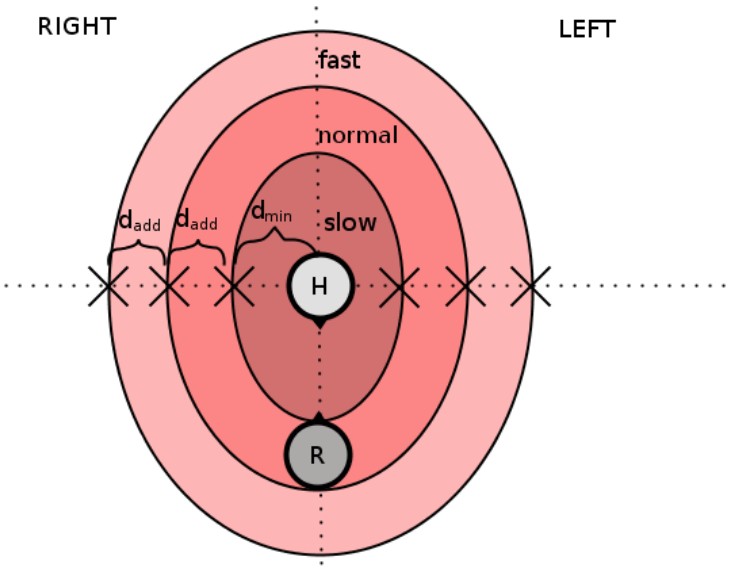


Figure 4.1: *The way points with relation to the user ('H') for each of the seven possible commands that can be given. The way points are represented by 'X'. If the* `STOP` *command is given, the robot ('R') remains at its position.*

4.2 Usability explorations

The algorithms are tested in a laboratory at Philips. There a robot and a room are provided. However, the gesture recognition module is not robust enough to be used in the user tests. Therefore, a Wizard of Oz setup is created, wherein the experimenter acts as the gesture recognition system and manually provides input to the ICP algorithm. The distinction between gesture speeds is guessed by the experimenter. The setup is further described below.

4.2.1 Robot platform

The robot used for testing is called Rafael. Rafael stands for Robot Assistant For Aiding the ELderly and is based on Willow Garage's Turtlebot[1] [29]. Its base is a non-holonomic drive which is controlled by sending velocity commands. These commands consist out of a translational and rotational component. The base has cliff sensors to prevent damage due to driving off a ledge and a bumper to detect collisions. Rafael has a Hokuyo[2] laser range finder for obstacle detection and localization. A Kinect[3] sensor is mounted on top so it has a clear line of sight to detect people. Figure 4.2 shows a picture of Rafael.

A second Kinect is positioned such that it has an overview of the collision site. A user tracker makes use of this Kinect's data to detect the people in the room. The separate Kinect is used because the user tracker does not function properly on a moving platform.

[1] `http://www.turtlebot.com`
[2] `http://www.hokuyo-aut.jp/02sensor/#scanner`
[3] `http://en.wikipedia.org/wiki/Kinect`

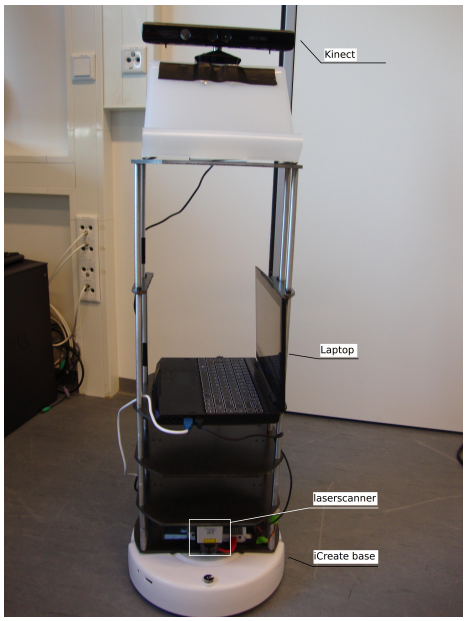


Figure 4.2: *Picture of Rafael, the robot used in the experiments.*

To facilitate communication between the different sensors and drivers a distributed computing environment is needed. ROS[4] provides access to the hardware and hides the complexities of transferring data between components [65]. Furthermore, as mentioned before ROS has a well integrated simulator. Therefore it is used as a middleware platform for the robot.

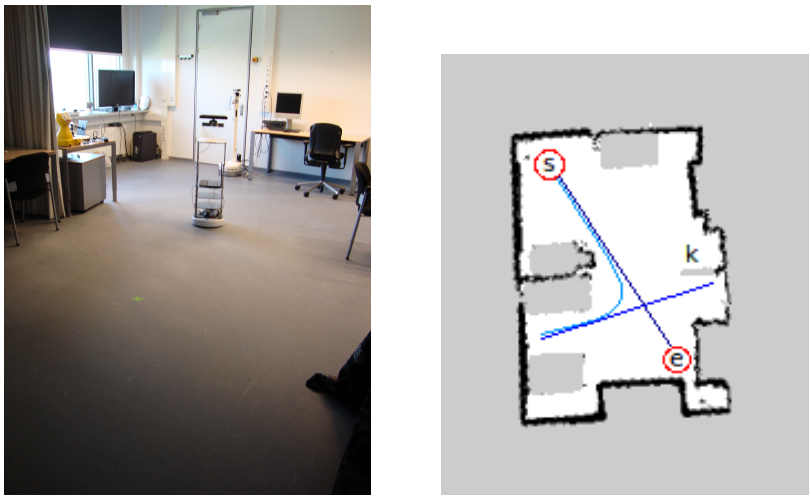
4.2.2 Environment

The usability tests are conducted in a lab room at Philips. Figure 4.3a shows a picture of the lab setting, including the robot. The robot has to travel from one corner to the opposite diagonal corner (approximately 7 meters). Participants will cross the robot's path in different ways, which resemble the `DYN1, DYN2` and `DYN3` conditions described in Section 3.2.3. Figure 4.3b shows a top down representation of the room. The start and end position are represented by the letters 's' and 'e' respectively. The approximate paths the participants are instructed to take are drawn using different colors. The participants are instructed to try and maintain a constant speed while crossing the path of the robot. However, if they feel uncomfortable with the robot's presence (e.g. they think the robot is going to hit them), they are free to step aside.

4.2.3 Experimental design

This section describes the experimental procedures for the user tests. The flow of the experiment is depicted in Figure 4.4.

[4]ROS (Robot Operating System - `http://www.ros.org`) is an open-source project to build a meta-operating system for mobile manipulation systems



(a) **(b)**

Figure 4.3: *On the left is a photo of the room in which the experiments will take place. Rafael is driving from his start position towards the goal (where the photographer is standing. On the right is a top down map of the room. Black pixels represent obstacles, white represents free space and grey represents unknown areas. The start and goal positions are indicated with respectively 's' and 'e'. This map is used to calculate the global path. The 'k' indicates the position of the extra Kinect. The blue lines are the paths the participants have to walk for the various conditions.*

After signing the consent form, the participants are introduced to the robot. The experimenter explains that the robot will be used in a home-like environment and will therefore need to avoid people. Then, the participants are shown which sensors are used and how the robot will detect them. Finally the experimental flow, as depicted in Figure 4.4 is explained. After experiencing each algorithm, the participants have to fill out a small questionnaire about the performance of that algorithm. They are handed the questionnaire before the trials begin so they know which aspects to focus on. On the questionnaire, there are four five-point Likert items, ranging from "I don't agree" to "I agree", the participants have to rate for each algorithm. The items in the questionnaire are as follows:

1. The robot was acting smart.
2. I thought the robot was going to run into me.
3. I trust the robot.
4. I would allow this robot in my home.

At the end of the experiment, the participant has to answer two more questions: "Which algorithm did you like the best?" and "Why?".

Before beginning the experiments, the subjects are shown a demonstration of the robot traveling from the start position to the goal. Then, several trials[5]

[5]A trial is one attempt of the robot to travel from the start to the end position.

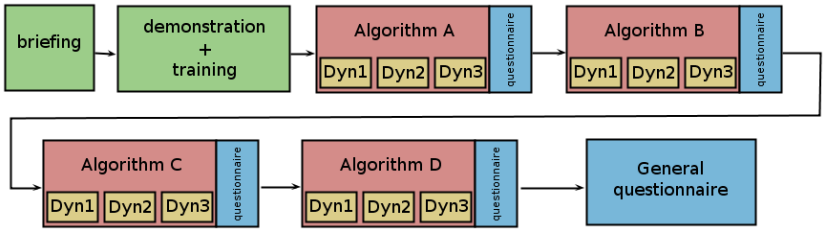


Figure 4.4: *The flow chart of the user test procedure. The order of the algorithms is randomized. The last block is always the ICP algorithm.*

using the base algorithm are performed to make the participant more familiar with the movements of the robot and prevent learning effects. During the trial the participant will remain stationary, so he or she can observe the robot closely.

Each participant experiences every algorithm (the base algorithm, AA, HMMA and ICP) once in every condition ($4 \times 3 = 12$ trials). The order of the algorithms is randomized, however for practical reasons the ICP is always last.

Once an algorithm is selected, all conditions within that algorithm are tested sequentially. First is the `DYN1` condition, then the `DYN2` and finally the `DYN3` condition. The conditions are always tested in this order. Each sequence of the three conditions is called a block. After each block, the participant has to fill in a small questionnaire about the likability of the robot and level of trust the participant has in the robot.

Besides the questionnaires, the time and detour of the algorithm is again measured. The results of the measurements and the questionnaire are described in Section 4.3.1 and Section 4.3.3.

4.3 Usability results

The results of the usability tests are divided in three parts. First, in Section 4.3.1 the measured time and detour results are presented. In Section 4.3.2 a comparison is made between the simulator and the real world. Finally, Section 4.3.3 contains the results of the conducted survey.

4.3.1 Efficiency measurements

The efficiency of the algorithm was measured in the usability tests. The results are presented below. The robustness of the algorithms was not measured. The robot could always reach its goal. When it could not, it was the results of a technical failure. The trial was then reset and attempted again. On average, this happened once with every participant.

Efficiency results

The algorithms are again compared on the **time** it took for the robot to reach the goal and the amount of **detour** the robot made. However, in contrast to the simulation experiments, the number of successful trials is *not* counted. Failed trials in the simulator were a result of collisions of the robot with the second

Condition	Time	Detour
STAT	-	-
	p=.766	p=.736
DYN1	F(2,12)=1.127	F(2,12)=.039
	p=.356	p=.961
DYN2	F(2,11)=5.445	F(2,11)=.273
	p=.023	p=.766
DYN3	F(2,11)=2.136	F(2,11)=.377
	p=.165	p=.695

Table 4.1: *The significance results of the one-way ANOVA comparisons of the measured **time** and **detour** in the user tests.*

robot. This is because the second robot, which represented the moving person in the simulations, would continue to follow its path no matter what. However, in the user test trials participants would avoid the robot if the robot does not avoid them. Therefore, the only unsuccessful trials were a result of malfunctioning hardware or random initialization errors. These trials were excluded from analysis and performed again. Plots of the mean and standard deviations of the data can be found in Figure 4.5.

Condition: NONE

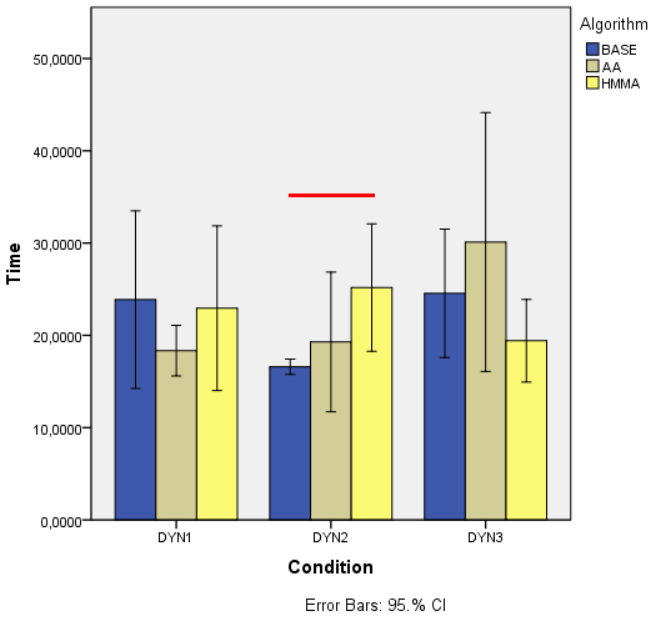
The simulation experiments yielded no significant differences in the performance of the algorithms. Therefore, no experiments were ran testing the AA and HMMA algorithms in the NONE condition. Consequently no comparisons can be made.

Condition: STAT

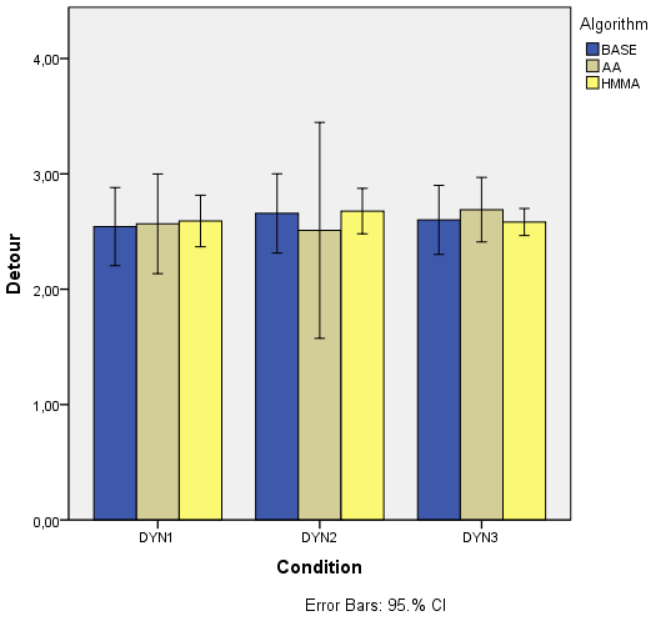
The AA and HMMA algorithms are designed to act the same around a stationary person. Combined with the fact that in the simulations no significant differences between the algorithms were found, only the base and AA algorithms were tested in this condition. The results were compared using an independent sample t-test. Both the differences in **time** and **detour** were not significant across both algorithms, $p = .766$ respectively $p = .736$.

Condition: DYN1, DYN2 and DYN3

The dynamic obstacle conditions were tested by all three non-interactive obstacle avoidance algorithms. A one-way ANOVA was performed to compare the performances of the algorithms. For all three dynamic obstacle conditions there were no significant differences in **detour**. However, there was a significant difference in **time** between the algorithms in the DYN2 condition, $F(2,11)=5.445$, $p=.023$. Tukey post-hoc comparisons showed that the base algorithm ($M=16.59$) reached the goal significantly faster than the HMMA ($M=25.18$). All significance results can be found in Table 4.1.



(a) *Time*



(b) *Detour*

Figure 4.5: *Plots of the usability tests. (a) shows the average time and (b) the average detour. The red lines above the bars indicate significant differences between the algorithms.*

4.3.2 Baseline comparison

To test whether the results gathered from the simulator tests are transferable to the real world several additional analyses are performed. The performance of the base algorithm in the `NONE` and `STAT` conditions in the simulator and the real world are compared.

To compare the simulation results with the real world results, an independent t-test was performed. Only the performance of the base algorithm is compared to eliminate the influence of the user tracker on the results. Since the straight line distance to the goal was not the same in the simulated and the real world, the time and detour are corrected for this difference. Each variable is divided by the straight line distance to the goal. Figure 4.6 shows the mean time and detour.

For the `NONE` condition the difference in both **time** and **detour** was significant (respectively p=.003 and p=.000). In the `STAT` condition both variables were significant too (respectively p=.018 and p=.000). The implications of these results can be read in Section 4.4.2.

4.3.3 Survey results

In the usability tests the participants interacted with all four obstacle avoidance algorithms. After interacting with each algorithm, they filled out a short survey about the amount of trust they placed in the robot. Also, at the end of the experiment the participants had to chose which algorithm they liked best. The results are described below.

The performance of the gesture recognition system was not at an acceptable level at the time of the user tests. The experimenter therefore acted as the gesture recognition system and manually inputted the performed gesture by the participants. The participants had no knowledge of this and thought the robot could recognize their gestures.

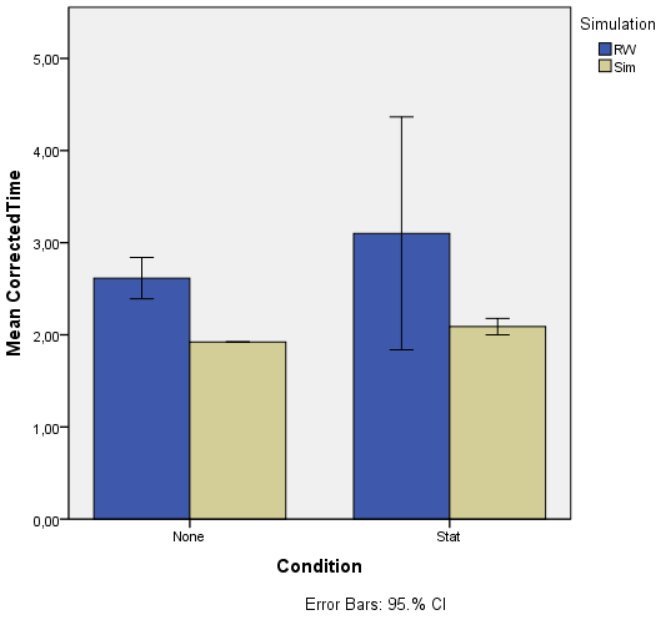
Participants

For the final usability tests, five participants were recruited. They were all interns working at Philips Research and studying a non computer science or robotics related field. Their age ranged from 23 to 26 years. In the survey they indicated how much experience they have interacting with a (mobile) robot. All participants had at least some experience with interacting with robots. The participants did not receive any compensation for participating in the experiment.

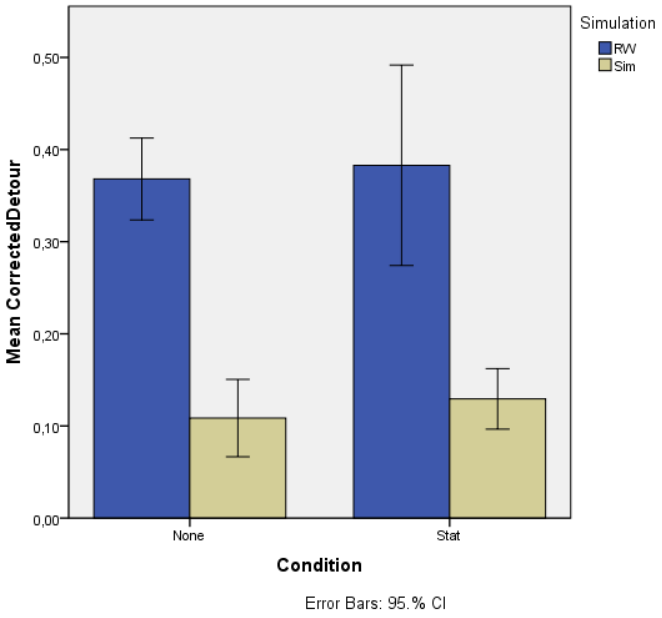
Survey scores

The scores on the Likert scale can be analyzed as ordinal or interval data. There is some debate about which is best [36,53]. However, I agree with Jamieson [36], that Likert scales should be treated as ordinal data.

Figure 4.7 shows the medians of the given answers per question per algorithm. A Kruskal-Wallis test showed that the differences in perceived trust are not significant (χ^2=4.120, p=.259).



(a) *Time*



(b) *Detour*

Figure 4.6: *Plots of the baseline comparison. (a) shows the average corrected time and (b) the average corrected detour.*

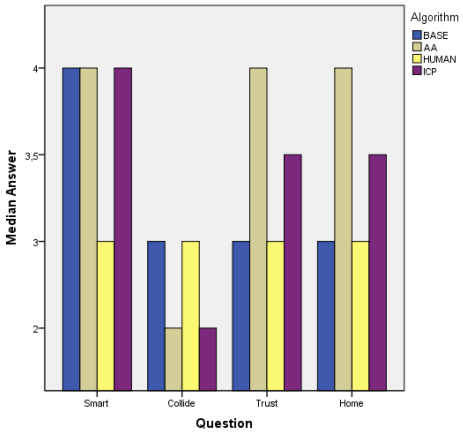


Figure 4.7: *Medians of the answers in the survey.*

Final choice

At the end of the experiment, the participants had to choose which algorithm they liked the best. Three out of five participants chose ICP. Some explanations of why they chose ICP are:

> *"Number 4 came less close and had a better reaction. Because of [the] interaction you know what the robot is going to do."*
>
> *"I liked to steer him."*

One participant preferred AA ("[The r]obot was reacting most smart on my movement.") and one chose HMMA ("It feels very natural when the robot turns right/left. I didn't see the trick of algorithm 4 [ICP]. It seems the robot just stopped and waited for me, but it didn't give any feedback about my gesture."). No participants preferred the base algorithm.

4.4 Discussion of the usability explorations

Each of the result sections is discussed below. First are the measured performances in Section 4.4.1. Then the baseline comparison is discussed in Section 4.4.2. Finally, the user preferences are discussed in Section 4.4.3

4.4.1 Discussion of the real world measurements

The robustness of the algorithms in the real world will not be discussed here, because the number of samples is too small. However, the relations between the performances of the algorithms will be discussed.

Efficiency

As expected there were no significant differences found between the algorithms in the `STAT` condition. The robot should behave (approximately) the same for every algorithm and it did.

In the dynamic obstacle conditions there was no significant difference in **detour** between the algorithms. Observations showed that the robot usually was too slow to benefit from its detour. By the time the robot adjusted its trajectory, the user had already done so and passed the robot. The robot then had no need to detour, because there were no more obstacles in its way. This also explains the difference in results between the trials in the simulated and the real world. In the simulator, the moving obstacle would not react to the robot in any way. In the real world the participants did.

Because the users (un)intentionally made way for the robot, the differences in average **time** were also not significant, except in the `DYN2` condition. In this condition the base algorithm reached the goal position significantly faster than the HMMA algorithm. There were quite some problems with using the user tracker. The second Kinect (responsible for detecting the user) was positioned such that it had a good view of the collision site. However, the area it could see was quite limited. The user was therefore detected late. Earlier detection of the user should result in better results. Furthermore, the robot was detected as a user sometimes. This caused the robot to suddenly stop (there is an obstacle detected at its position) and get stuck. Finally, there was a bug in the user position detector in the HMMA algorithm. If there was no user detected, the position would be set to (0,0) and the closest point in the path was presented as the user's position, which was in the middle of the room.

The points mentioned all contributed to the relatively bad performances of the AA and HMMA algorithms. The base algorithm did not have these problems, and thus performed as expected.

4.4.2 Comparison of the simulator to the real world

In this section the generalizability of the data gathered in the simulator to the real world is discussed. First, the absolute results (described in Section 4.3.2) are discussed. Finally, the comparison results of the analyses of both the simulations (Section 3.3) and usability tests (Section 4.3.1) are compared.

Absolute results

The performance of the algorithms in the simulations differed significantly from the performance in the real world on both time and detour. First possible explanations for the difference in detour will be discussed, then possible explanations for the difference in time.

The difference in the results in **detour** can be explained by the fact that the position estimations, and thereby the calculation of the detour is different in both situations. In the simulator, the position of the robot is a precise measurement. It is directly taken from the simulator and is therefore completely accurate. In the real world, the positions of the robot were estimated by the localization module. Because these are estimations, a certain amount of uncertainty is introduced. This uncertainty in the position estimates is reflected in the larger detour values. In order to more accurately test whether the simulated and the real world are similar, a more precise position measurement for the robot has to be used, for instance an tracker system or a top-down camera. This way,

the detour measurements in the real world will be (almost) as accurate as the measurements in the simulator and a proper comparison can be made.

A second explanation for the difference in detour could be that the robot did make a larger detour in the real world experiments than in the simulations. This was caused by the testing environment. In the simulator there were no obstacles (except for the intended one). In the lab setting, there were desks, chairs and other obstacles. Although the path is cleared as much as possible, the robot sometimes came too close to a desk - mainly because the position estimation was wrong - and had to adjust its path to avoid that desk. A lab environment with no obstacles could thus yield other results.

The differences in the time measurements can be caused by a multitude of possibilities. The first possibility could be that the computers on which the robot and the simulator ran have different processing power. The computer on which the simulations ran was very powerful. It could run the simulations faster than real time. The laptop on the robot in the user tests was less powerful. Furthermore, the input commands (setting a new goal) was done on a separate computer. This separate computer was also used for the visualization of the map and sensor data. Since this computer is not that powerful, it had some trouble to keep up the frame rate. Furthermore, the data was sent back and forth to the laptop and the computer via WiFi. The usage of WiFi is slower than running everything on one computer, so this could explain the differences as well.

Another explanation could be that the robot drove slower on some sections, or made a detour. As with explained in the previous paragraph, the robot sometimes encountered a desk in its path. When it comes close to an obstacle, the robot will slow down or stop. This costs time. Also, the robot has to adjust its path and travel a longer distance. This will also take up extra time.

The distance correction applied to compare the measured time between the simulator and the real world might not be fair as well. Since the distance traveled in the real world is shorter (approximately 6 to 7 meters instead of 8) the start-up (initial acceleration) and slow down (final deceleration to the goal position) in each trial relatively large in relation to the total time traveled. By applying the correction the proportions of the start-up and slow down time become skewed. This is disadvantageous for the shorter distance and thus the real world trials.

Ranking comparison

In this section the rankings of the algorithms in the simulated and the real world are compared. Even when the absolute differences between the simulated and the real world are significant (i.e. not comparable), the simulator can prove useful to determine the ranking of the algorithms. The only comparisons that will say anything about the ranking differences are the ones where both the simulated and real world trials have significant differences between the performances of the algorithms within the same condition. Otherwise, it is only established that in both situations there is a lack of results.

Unfortunately, none of the conditions yielded significant differences on either time or detour in both the simulated and the real world. Therefore, no conclusions about the generalizability of the ranking of the algorithms concerning

time or detour can be made.

4.4.3 Discussion of the user preferences

ICP cannot be compared directly to the other algorithms with objective measures. Because the algorithm has to wait for input the time and detour measurements will not say anything about the performance of the algorithm. Therefore, the algorithms were compared to each other by the participants based on trust and likability. The survey the participants had to fill in did not yield any significant results, however. This is probably because the participants did not have a clear view on the differences in behavior between the algorithms. In order to keep the observations objective, the workings of the algorithms were not fully explained to the participants. Because of technical difficulties (already discussed in Section 4.4.1) the AA and HMMA algorithms did not perform as intended. Furthermore, each participant only had three trials (one in each `DYNx` condition) to form a representation of the behavior of the algorithms. This was not sufficient to form a clear a idea of what the robot will do in certain situations.

When the participants had to choose the algorithm they preferred, the majority chose the ICP algorithm. All participants that chose ICP, said they liked it best because it was interactive and most transparent. They knew where the robot would go when they controlled it. This is a typical reaction for user of perceptive systems. However, these results might be different if the participants have a better understanding of how the robot will behave using the other algorithms, because over time the behavior of the robot will become more transparent to its users.

The participants who did not choose ICP said they thought it was not that smart of the robot to just top and wait for a command. Since (initially) stopping was necessary for the gesture recognition this objection will be nullified once the original intention is implemented. Remember that the intention was to let the robot read the body language of the users *while driving*. How this can be approached is described in Section 6.

Chapter 5

Explorations in gesture recognition for ICP

In the envisaged scenarios of home assistance, ICP is equipped with an interaction component via which the user can provide the robot with directional clues. Several ways to implement the interaction interface exist, such as remote controls [50], speech-based communication [81], and gesture-based interaction [71]. As recently underlined in [71], the use of gestures in socially interactive robotics is crucial, but yet largely unexplored. Gestures provide a natural means to indicate deictic signals. For specifying directions or locations, as in our context of human-robot navigation, gestures are preferred over speech-based control [56]. In this chapter, the possibility of using gestures for providing directional controls is examined. To further pursue our approach of gesture recognition for ICP, we have designed three directional gestures (LEFT, RIGHT, and STOP). A dataset has been acquired from seven users. Each user has performed each of the gesture classes in three velocities. Based on this dataset, we will explore whether robust gesture recognition can be achieved. More specifically, the following questions will be anwsered:

1. Using different classification methods known from the literature, what accurracies can be achieved?
2. Is the velocity feature appropriate as a distinctive control dimension?
3. Is the concept of one-shot learning feasible for our purposes?
4. Can users consistently reproduce samples from this limited gesture repertoire?

As explained in Section 4.1, gesture interaction is assessed via a set up where users stand still in front of a Kinect camera. In Section 5.2, the set ups for the data acquisition and gesture recognition experiments are described. The results are presented in Section 5.3 and discussed in Section 5.4.

5.1 Gesture recognition algorithms

In this thesis, two types of gesture recognition (GR) are explored. The first is *trajectory-based* gesture recognition using dynamic time warping (DTW), which is described in Section 5.1.2. The second type of algorithms are based on *feature-based* gesture recognition and concern a multi-layered perceptron (MLP) and a k-nearest neighbor classifier (KNN). Section 5.1.1 first explains the representation used in this thesis of the gestures. Subsequently, in Sections 5.1.2 and 5.1.3, the methods used for gesture recognition are described. The performance of the DTW, MLP, and KNN classifiers is tested in several experiments, which are described in Section 5.2.3.

5.1.1 Gesture recording and representation

As mentioned in Section 2.3 there are three main components in gesture recognition: the sensor, feature extraction and analysis. For our data acquisition process, a Kinect sensor is used to create a depth image of the scene. From this image, the OpenNi body pose tracker[1] can extract the positions of the joints of the user[2]. A joint j is represented as a point at a certain time instance t in the three-dimensional interaction space: $x^j(t), y^j(t), z^j(t)$. In our set up, users are requested to perform each gesture in one second. Therefore, at a frame rate of 25 Hz, each gesture is specified as a sequence of six joint coordinates, each consisting out of *n=25* frames:

$$\{x^j(t), y^j(t), z^j(t)\} \quad \text{with} \quad t \in [1, n] \quad \text{and} \quad j \in [1, 6] \tag{5.1}$$

Before data recording, the tracker needs to be calibrated. For calibration, the user has to stand in the 'psi-pose' (see Figure 5.1). This calibration step must be performed every time the user has left the view of the Kinect. After the calibration, the position of every joint is available for processing. The current gesture recognition implementation uses the shoulder, elbow and hand joints from both the left and the right arm. Furthermore, in order to compensate for the rotation and the relative position of the user to the Kinect, all the joints' positions are taken relative from the position of the neck joint.

5.1.2 Trajectory based recognition

The implementation of the first gesture recognition system (which is explained below) is ported from [91][3]. Using one-shot learning [47], for each gesture a template is recorded. This template consists out of 25 frames. Each frame contains the relative positions the the six joints mentioned above. Once the templates are recorded, the gestures can be compared to them. Each unknown input frame is matched with the poses in every template. The poses are compared with each other using the Euclidean distance measure. The Euclidean distance between

[1] `http://ros.org/wiki/openni_tracker`

[2] The joints defined by OpenNi are: head, neck, torso, left shoulder, left elbow, left hand, right shoulder, right elbow, right hand, left hip, left knee, left foot, right hip, right knee and right foot.

[3] Our code was ported from Processing (a language derived from Java) to C^{++}.

Figure 5.1: *The psi pose needed to calibrate the body pose tracker.*

two pose vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as follows:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})} \tag{5.2}$$

The performance of this distance measure in pose recognition can be improved by altering this equation slightly. For instance, some poses require only one arm to be used. The location of the other arm is in such cases less or not important. By multiplying the distance of the joints on the left ($\mathbf{x}_{left}$) with a weight (w_l) and the joints on the right ($\mathbf{x}_{right}$)with another weight (w_r), the importance of each side can be varied. The weights can be set to a value between zero and one. Furthermore, the sum of w_l and w_r has always to be two. This is to ensure the costs are never zero. In this thesis there are gestures performed by both a single arm (`LEFT` and `RIGHT`) or two arms (`STOP`). To reduce the number of false positives the weights of both sides are set to one.

Besides setting weights to the left and right side of the users, weights can be set for each of the three dimensions in space. These weights w_{dim} can be used to reduce the importance of one dimension. In this thesis all dimensional weights are set to one, and thereby making every dimension equally important. This is done because each participant can define his or her gesture themselves. Some may prefer one direction, while others do not. By incorporating the weights in the distance measure, the new distance measure will become:

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) = &\sqrt{w_l((w_{dim}(\mathbf{x}_{left} - \mathbf{y}_{left}))^T (w_{dim} \circ (\mathbf{x}_{left} - \mathbf{y}_{left}))) \dots} \\ &\overline{\dots + w_r((w_{dim}(\mathbf{x}_{right} - \mathbf{y}_{right}))^T (w_{dim} \circ (\mathbf{x}_{right} - \mathbf{y}_{right})))} \end{aligned} \tag{5.3}$$

For the comparison of gestures (instead of poses) the temporal structure needs to be taken into account as well. The speed in which the gesture is performed is important for use in ICP. *Dynamic Time Warping* (DTW) is therefore a suitable technique to use. DTW takes varying speed and/or acceleration into account. The standard definition of DTW distance between two time series $\mathbf{x}_m$ and $\mathbf{y}_n$ (with M and N elements respectively and m and n representing the

frame indexes) is:

$$\phi(m,n) = \Phi(m,n) + min(\phi(m-1,n-1), \phi(m-1,n), \phi(m,n-1)) \quad (5.4)$$

where $\phi(m,n)$ is a $(M+1) \times (N+1)$ matrix with $\phi(0,n)$ and $\phi(m,0)$ are initialized as zero. The cost function $\Phi(m,n)$ is the distance measure defined above in Equation 5.3 as $\Phi(m,n) = d'(\mathbf{x}_m, \mathbf{y}_n)$. The DTW distance between two series is equal to $\phi(M+1, N+1)$.

Once the cost matrix is built, it is possible to find the optimal path using backtracking. The cost matrix is traversed from the end point (M,N) to the start point $(1,1)$. If the optimal path travels along the diagonal of the cost matrix, the gesture is performed at the same speed as the stored gesture (normal speed). If there are more vertical steps the gesture is performed faster than the template. If there are more horizontal steps, the gesture is performed slower. The gesture speed is discretized in three classes: `SLOW, NORMAL` and `FAST`. The classified gesture together with the speed provides input for ICP.

5.1.3 Feature based recognition

The second type of gesture recognition employed in this thesis computes a set of distinctive features from each gesture representation. These features have been tested on various pen-based gesture recognition tasks [89]. The input representation contains features derived from a complete gesture trajectory, consisting of a number of *n (x,y,z)* coordinates for each of the six joints measured. Each feature vector contains the following elements:

d_0^j	–	distance between start-end sample
d_x^j	–	max horizontal distance between start and all other samples
(c_x^j, c_y^j, c_z^j)	–	the centroid of each of the six joints j
v^j	–	the average velocity of each joint

Two common machine learning algorithms are trained and tested on the recorded gesture samples: a multi-layered perceptron (MLP) and a k-nearest neighbor (KNN) classifier (for a description of both classifiers, see, e.g., [21]). These systems are explored in a within-user and between-user set up. The first explores how well a trained gesture recognition system is able to recognize data from one single user (as would be a typical task in our chosen application scenarios). The latter explores how the system can distinguish between gestures from a heterogeneous population of users.

5.2 Data acquisition

In this section the set up of the data acquisition is described. The recorded data are used to train and test the gesture recognition algorithms.

5.2.1 Data acquisition guidelines

The participants are briefed about the intended application of the software. They are instructed to imagine that the robot is standing in front of them. The

template-based gesture recognizer is subsequently initialized by recording one template gesture for each gesture class, `STOP`, `LEFT` and `RIGHT`. The participants are allowed to create their own gesture shapes while keeping some guidelines in mind:

1. The `STOP` gesture should be performed by lifting two arms simultaneously in front of the user.
2. The `LEFT` and `RIGHT` gestures should be performed using only the respective arms.
3. The gestures should be easy to reproduce.
4. The user has to be able to perform each gesture at a slower pace and a faster pace.
5. All gestures should feel natural to the user.

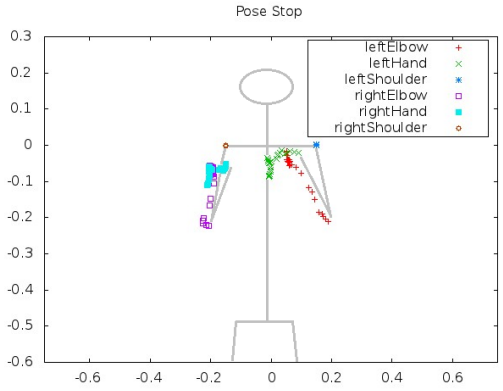
The first two guidelines ensure that the gestures are easy to distinguish from each other. The third guideline is to make the users aware that they need to perform the same gesture several times, and that they should therefore generate a movement that can easily be remembered. The fourth guideline ensures that within each of the three gesture classes, a distinction can be made between different categories of velocity.

The fifth and final guideline requires special attention. In general, "natural" interactions are one of the most challenging topics of human-computer interaction [18, 69]. In particular in contexts where users are instructed to generate an unspecified intentional movement with a specific semantic connotation, the questions are: (i) to what extend users are able to generate consistent movements at all and (ii) whether sufficient movement characteristics are shared across users. These issues entail that it is uncertain whether recognition technology can be designed which is robust for between-user variance in case of natural interactions. In general, different users generate different movement characteristics if they are not instructed properly [90]. On the other hand, if the instructions contain precise directions as to how a movement should be produced, between-user variability can be controlled [52].

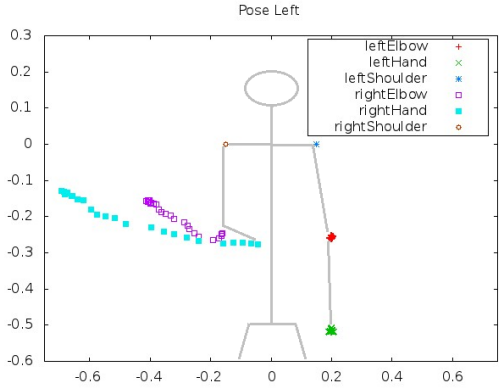
In a scenario where a companion/assistance robot is introduced in a home environment, the collection of a set of user-specific command gestures is part of the process of tuning the robot for the task of assisting a human user. Using the data collected in our experiment, we will be able to explore whether this calibration phase can be performed with sufficient precision for different users. For each of the three gesture classes and for each of the three speed categories, each user is requested to generate one template gesture for each of the three gesture classes and a further 10 corresponding gesture samples per command[4]. So, in total each user will generate 93 gestures. The order of the nine commands is randomized.

As an example of gesture set, the gesture trajectories of one participant for each of the gestures are depicted in Figure 5.2. Note that the depth information is not plotted here.

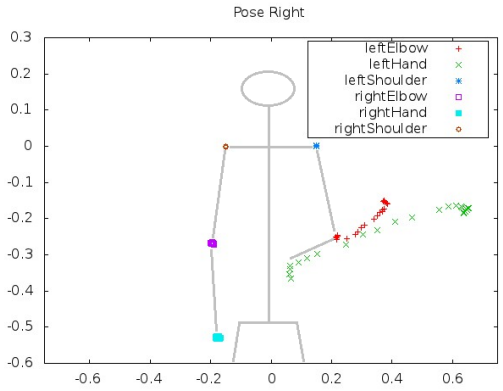
[4] A command is a combination of a gesture class and gesture speed. There are nine commands, each a combination from three gesture classes and three gesture speeds.



(a) *STOP*



(b) *LEFT*



(c) *RIGHT*

Figure 5.2: *Example gestures made by one participant. The depth information is removed. The data is shown for the position of the joints in the two dimensional plane of the front of the user. (a) shows a* `STOP` *gesture, (b) a* `LEFT` *gesture and (c) a* `RIGHT` *gesture. The skeleton does not represent the user, but is inserted for viewing convenience.*

5.2.2 The data acquisition process

After the instruction phase and recording of a template gesture for each of the gesture classes, the participants are allowed to practice with the gesture recognizer. This practice phase is used to train users how to consistently produce each gesture shape. Note that it is possible to record a template gesture again if the gesture recognition has a lot of false positives or false negatives for that gesture, or if the participant thinks another gesture would be better.

During each session the positions of the used joints (shoulder, elbow and hand on both sides) are recorded at 25 frames per second. The participants have no real-time feedback whether the gesture recognition system recognizes their gestures. Afterward the data will be played back through the gesture recognizer. The output of the gesture recognition is recorded together with the label of the performed gesture. In Section 5.3 the results are described.

5.2.3 Gesture recognition experiments

As outlined above, three gesture recognition systems will be used for gesture recognition. Using the acquired data, the following gesture recognition experiments will be performed:

1. *within-subject recognition*: all three recognizers will be trained and tested on data from each individual subject. A distinction will be made between the three main gesture classes and the nine commands.

2. *between-subject recognition*: the same set up will be used, but all recognizers will be trained on the data acquired for all subjects.

3. *k-shot learning*: whereas the DTW (trajectory-based) classifier uses one-shot learning, it is known from machine learning that the more training data is available, the better the performance of the classification system. This holds in particular for the MLP classifier. To explore how much data is required for achieving a certain recognition performance, the k-shot learning experiments examine to what extent data availability influences recognition. The results of these experiments provide insight into the afore-mentioned tuning process of our ICP approach: how many templates should a user produce before the robot is able to recognize gestures with high accuracy?

The results of each of the gesture recognition systems are described below.

5.3 Gesture recognition performances

A dataset was collected for testing the gesture recognition (GR) software. For seven participants, data was acquired as described in Section 5.2. This data was later used to test (offline) both the trajectory based and feature based GR systems. Below, the recognition results for the DTW, MLP, and KNN gesture recognition systems are described. But first, a description of the acquired data set will be given.

5.3.1 The collected data set

Unfortunately, due to unforeseen circumstances, several recorded files appeared to be empty after the data collection phase. The following distribution of collected samples per subject was yielded:

class	subj2	subj3	subj4	subj5	subj6	subj8	subj9	total
LF	10	10	10	0	10	0	0	40
LN	10	10	10	10	10	10	10	70
LS	10	10	10	9	10	7	10	66
RF	10	10	10	0	0	0	0	30
RN	10	10	10	10	10	10	10	70
RS	12	12	12	0	1	0	0	37
SF	10	10	10	0	0	0	0	30
SN	10	10	10	10	10	10	10	70
SS	0	0	0	11	11	11	11	44
total	82	82	82	50	62	48	51	457

Table 5.1: *Distribution of collected samples per subject. L, R, S means, respectively left, right, stop. F,N,S means, respectively, fast, normal, slow.*

In total 457 gesture samples have been recored. As can be observed, only three of the nine gesture classes was completely recorded for all subjects. A post-hoc analysis of the recording software showed that one of the responsible ROS modules did not always close the data file correctly. Fortunately, for all "normal" speed conditions, all data was recorded. As a consequence, the following experimental set ups were defined:

1. nine-class experiments: all available data was used for distinguishing between the 3x3 gesture classes

2. three-class experiments: only the data acquired in the "normal" speed condition was used

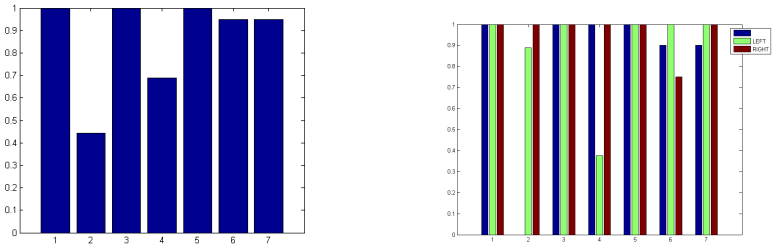
5.3.2 Performance of the trajectory based GR (DTW)

As is explained in Section 5.2.3 the performance of the trajectory based GR system is compared on two levels: (i) within-subject and (ii) between-subject.

Within-subject, three classes

The recorded data was processed by the DTW system as follows: the recorded template gesture (at normal speed) was used as prototypical training sample. For each of the three classes $c \in \{RN, LN, SN\}$, all ten samples x_c were classified by the closest match $minidx(\phi(T_{RN}, x_c), \phi(T_{LN}, x_c), \phi(T_{SN}, x_c))$, where $minidx$ returns the index of the smallest DTW-distance between the unknown input x and the templates T, as explained in Section 5.1.2. Figure 5.3 shows the recognition rates for each participant.

For three out of seven participants, the recognition rate of the trajectory based classifier was perfect (100%). This shows that it is very much possible for DTW to be a robust method to classify the three defined classes.



(a) *Overall performance* **(b)** *Performance per gesture class*

Figure 5.3: *The recognition rate of the trajectory base GR system. (a) Each bar represents the average recognition rate over all gesture classes, for each participant. (b) The recognition rates per gesture class per participant.*

For two participants, the recognition rate was far below 90%. In Figure 5.3b are the recognition rates per participant per gesture class shown. The bad performance of the classifier on participant two can be explained by the inability of the classifier to recognize the `STOP` gesture. The gestures are not similar enough to the template and are therefore never recognized. A new template could have improved the performance. The same goes for participant four with the `LEFT` gesture.

When the speed is involved, the recognition rate drops drastically. Figure 5.4 shows the recognition rate per participant. The DTW classifier is thus not good at classifying the speed of a gesture within subjects.

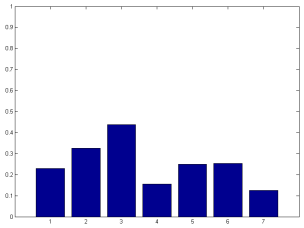


Figure 5.4: *The recognition rate of the trajectory base GR system. Each bar represents the average recognition rate over all gesture classes, for each participant. The speed is also taken into account for the recognition rate.*

Between-subject (for all classes and three classes)

The performance rate over all subjects was 86%. This results is when the speed is not taken into account. When the speeds are counted as separate classes, the performance rate drops to 33%. This is consistent with the findings in the within-subject comparisons.

5.3.3 Performance of the feature based MLP and KNN

The template-based approach of the DTW classifier is similar to k-nearest neighbor. To make a similar comparison, experiments with k=1 are reported below. Furthermore, to compare the one-shot learning approach of the DTW method, classification results of the MLP also include learning based on 1 single example only. However, since in particular MLPs are known to require relatively much training data, a more elaborate experimental set up was employed, where the number of training samples per class was varied in different ways. For each of these conditions, the average performances for 100 random selections of training and testing data are reported.

Similar to the previous section, results are reported below for within-subject and between-subject settings.

Within subject (3 classes)

Table 5.2 shows the performance rate per test subject for different training set size.

	Number of training samples per class								
user	1	2	3	4	5	6	7	8	9
u2	64.8	87.1	96.4	97.2	96.7	95.0	95.6	97.5	98.3
u3	70.2	92.3	98.8	99.7	100.0	99.2	100.0	100.0	100.0
u4	78.5	99.4	100.0	100.0	100.0	100.0	100.0	100.0	100.0
u5	61.9	89.6	97.4	98.3	98.7	97.9	100.0	100.0	100.0
u6	51.1	97.5	100.0	100.0	100.0	100.0	100.0	100.0	100.0
u8	67.4	92.3	92.9	93.3	93.3	90.4	95.0	94.2	93.3
u9	65.9	89.6	94.0	95.0	93.3	92.9	91.1	95.8	96.7
avg	65.7	92.5	97.1	97.7	97.4	96.5	97.4	98.2	98.3

Table 5.2: *Performance of the MLP trained with a varied amount of training samples per class.*

Table 5.2 indicates that at least three training samples per class are required to achieve a reasonable high recognition rate. However, the best recognition rates are achieved for eight or nine samples per class.

Similarly to the MLP experiments, different training samples per class were varied for the KNN. The results are shown in Table 5.3 below. As can be observed, for eight or more training samples, zero error rates can be achieved.

user	Number of training samples per class								
	1	2	3	4	5	6	7	8	9
u2	96.3	95.8	95.2	94.4	100.0	100.0	100.0	100.0	100.0
u3	96.3	95.8	95.2	94.4	93.3	100.0	100.0	100.0	100.0
u4	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
u5	74.1	95.8	95.2	94.4	100.0	100.0	100.0	100.0	100.0
u6	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
u8	92.6	91.7	90.5	94.4	100.0	100.0	100.0	100.0	100.0
u9	92.6	95.8	95.2	94.4	86.7	100.0	88.9	100.0	100.0
avg	93.1	96.4	95.9	96.0	97.1	100.0	98.4	100.0	100.0

Table 5.3: *Performance of the 1-NN classifier trained with a varied amount of training samples per class.*

Within subject (all classes)

Random sampling was used for different fractions of training and test data. The average recognition results for 100 random configurations of train and test data per fraction are reported below.

user	Fraction of training samples, lumped over all classes								
	.1	.2	.3	.4	.5	.6	.7	.8	.9
u2	38.1	47.6	57.5	66.1	61.0	72.1	71.2	78.8	90.0
u3	30.3	48.8	64.2	64.5	75.6	77.0	79.2	82.5	80.0
u4	35.1	55.2	63.2	73.5	79.0	83.0	83.2	91.2	95.0
u5	45.3	52.0	68.0	68.0	74.4	81.0	76.0	74.0	92.0
u6	43.2	54.8	58.1	70.3	66.5	67.2	70.5	71.7	63.3
u8	54.4	67.9	77.6	83.5	79.2	82.1	84.3	78.0	92.0
u9	48.7	61.0	81.1	80.0	80.8	88.0	89.3	92.0	96.0
avg	42.2	55.3	67.1	72.3	73.8	78.6	79.1	81.2	86.9

Table 5.4: *Within-subject performance of the MLP trained with a varying fraction of training samples per class.*

Compared to the results of the DTW classifier, the decrease in performance when also distinguishing the three different speed categories (slow, normal, fast) is much less dramatic. One explanation is that the former uses one single template as prototypical example, whereas for this experiment, different sets of randomly drawn samples are selected. For a total of 457 gestures, the amount of training data ranges between approximately 46 (for fraction=0.1) to 411 (f=0.9).

Below, the results for this experiment with the KNN (k=1) classifier are depicted.

user	Fraction of training samples, lumped over all classes .1	.2	.3	.4	.5	.6	.7	.8	.9
u2	40.0	51.2	56.8	59.6	67.8	61.2	65.6	60.0	67.5
u3	44.1	56.1	64.6	62.9	73.7	74.5	70.4	73.8	80.0
u4	37.8	66.4	63.9	67.3	77.1	75.8	78.4	80.0	85.0
u5	49.3	59.5	64.6	66.7	64.8	67.0	68.0	68.0	72.0
u6	43.9	66.0	63.7	69.7	65.2	63.2	62.1	68.3	70.0
u8	60.5	64.2	84.1	86.2	85.8	80.0	88.6	86.0	96.0
u9	43.5	65.9	76.1	80.6	88.8	91.0	89.3	90.0	92.0
avg	45.6	61.3	67.7	70.4	74.7	73.2	74.6	75.2	80.4

Table 5.5: *Within-subject performance of the 1-nn trained with a varying fraction of training samples per class.*

Between subject (3 classes and all classes)

Again, random sampling was used for different fractions of training and test data. The average recognition results for 100 random configurations of train and test data per fraction are reported below.

# classes	Fraction of training samples, lumped over all classes 0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
three (MLP)	93.8	96.8	98.0	97.9	98.1	97.9	97.6	98.6	97.6
all (MLP)	48.3	59.5	65.3	68.1	70.9	73.7	74.2	75.9	77.6
three (KNN)	94.3	96.4	96.9	97.6	97.4	96.9	97.1	97.9	97.1
all (KNN)	50.0	57.2	64.9	66.7	70.1	71.5	72.7	75.1	76.7

Table 5.6: *Performance of between subject classification for KNN and MLP*

5.4 Discussion of the gesture recognition systems

The results from the gesture recognition experiments show that the performance of the GR system is acceptable. In particular in the within-subject experiments, which mimic the tuning phase of a classifier for one particular user, high performances can be achieved. The DTW classifier uses one-shot learning, employing just one single training template. This appears to be sufficient for three subjects, but for other subjects apparently more (or other) data is required.

Experiments with "k-shot learning" have been performed for the two feature-based classifiers (MLP and KNN). These show that for eight training samples or more, no errors are achieved. This is an important finding from our study, since: (i) it indicates that robust gesture recognition is indeed possible for this limited gesture repertoire, but also (ii) that tuning a gesture recognition system for a new subject requires more than one single sample. It should be noted that the DTW classifier can also be equipped with more training samples, but no empirical evaluation of this factor has been employed, which was due to the particular implementation of this classifier.

It is not clear why the performance of the DTW classifier drops so drastically (compared to the MLP and KNN) when the speed categories have to be distinguished. However, recognition performances of the KNN and MLP classifiers also indicate that the distinction in speed categories is difficult.

An analysis of the the consistency of velocity patterns of different users showed that apparently, some participants experience trouble with replicating the speed of the template gesture. As an example of the variability of the speed with which users generate gestures, consider Figure 5.5 below.

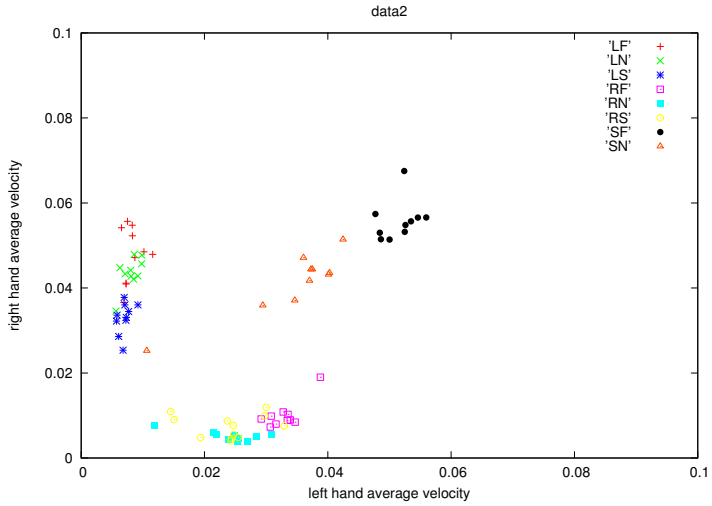


Figure 5.5: *Example of the variability in which the speed of a gesture is produced. Although a clear distinction in the main classes LEFT, RIGHT, and STOP can be observed, within those classes a considerable overlap in speed is contained.*

The average performance of some participants dropped drastically (in case of the DTW classifier) because for one gesture class 0% recognition is achieved. However, other participants showed that 100% recognition is possible if speed is not taken into account. The users of the system should therefore be trained better to replicate the templated gestures.

Nevertheless, our explorations show that it is possible to design a robust gesture recognition system, albeit that the amount of gesture classes is restricted to three.

Chapter 6

Conclusion

In order to be safely deployed in the homes of people, mobile robots have to avoid people in their path in a way that is both efficient for the robot and comfortable for people. Several people avoidance methods were explored in this thesis. The algorithms were:

- The base algorithm on which all other algorithms are based. The base algorithms treats moving obstacles as if they are static.
- Two dynamic obstacle avoidance algorithms which predict the movement of the obstacle (and the future position) using two different methods. Asteroid Avoidance (AA) uses linear prediction and Human Motion Model Avoidance (HMMA) uses known paths people can take to predict the future position of a user.
- An interactive obstacle avoidance algorithm, called Interactive Collision Prevention (ICP), which allows the users to indicate with a gesture where the robot could pass them.

These algorithms were all tested in several situations in which a dynamic obstacle (or person) would cross the path of the robot. The complexer the situation, the more intelligent the obstacle avoidance algorithm has to be in order to maintain efficiency. We saw that the base algorithm performs well only in a static environment. The AA algorithm can handle dynamic environments, but only if the moving obstacles follow a path that can be described by a linear function. When the paths of the moving obstacles can be detected (and are not linear), HMMA is a better choice.

Remember the research questions posed at the beginning of this thesis. Below are the answers to each of these questions:

- **Which algorithm is the most robust?**
  The most robust algorithm tested is probably ICP, because of its interactive nature. The user can decide where the robot should go. Collisions will not occur and the robot can reach its goal without any trouble. However, this is a responsibility of the user to give proper commands. Furthermore, simulations show that HMMA is the most robust of the non-interactive algorithms.

- **Which algorithm is the most efficient?**
  The efficiency of the algorithms depends on the environment. If there is enough room for the robot to avoid the user, the HMMA is most efficient, followed by AA. If there is not much room, the base algorithm is more efficient. The efficiency of ICP is hard to measure, since the amount of detour and time the robot needs to reach the goal depend on the command it gets from the user.

- **Which algorithm do users prefer?**
  The users prefer ICP the over the other algorithms. The main reasons were because they could control the robot and the behavior of the robot was more transparent using ICP than using the other avoidance algorithms.

- **Does the new concept of gesture-based ICP provide promising opportunities for interactive collision avoidance?**
  Since it is possible to robustly classify three gesture classes with the proposed classifiers, there is a possibility to use gestures in ICP. Some research is still needed about the user experiences with gesture based ICP however. Nevertheless, the current implementation can be expanded for use in more complex situations.

Further explorations are needed to reach the full potential of ICP however. In the next chapter different possible directions for future work are summed up.

Chapter 7

Future Research

Future research concerning human avoidance algorithms could be pursued in several directions. First of all, the conducted user tests were very short term. User tests conducted over a longer period of time could yield different user preferences. Furthermore, improvements on the current implementations of the algorithms can be made. The user tracker had some problems detecting the users. Also the online gesture recognition performance could be improved. Another improvement could be made by 'smartening up' the ICP. By creating a module that can determine a good way point, the robustness of ICP will improve. Finally, ICP itself can be implemented in ways other than the one proposed in this thesis. Each of these research directions are explained in more detail below.

7.1 Long term experiments

The user's preferences for an algorithm are highly influenced by the representation the user has of the behavior of the robot. The survey showed that users preferred ICP because it was the most transparent method. Since the users could experience each algorithm only once per condition, the behavior of the robot for each algorithm was not clear. A longer term test could improve the understanding of the algorithms. Also, the lab setting could feel unnatural. Furthermore, the nature of the tasks could bias the user towards ICP because the only task of the participants was to watch (and control) the robot. Experiments conducted in the users' homes, over a longer period of time, could yield different results than the results presented in this thesis.

These experiments could follow the same patterns as the user tests described in Section 4.2. Each algorithm will be tested for a longer period of time, for instance a week, in the home of the participant. After each period, the user will fill in a survey about the performance of the robot. After all algorithms are experienced by the user, a final survey about the relative performance of the algorithms and the user's preferences can be conducted.

Because the robot will be unsupervised for a longer period of time, the performance of the algorithms should be at an acceptable level. Also, mechanisms that enable recovery from a bad performance have to be present. The long term test will enable the participants to make a fully informed decision about which

algorithm they prefer.

7.2 User detection and tracking

As said before, the user detection and tracking modules did not perform as desired. Although many algorithms for people tracking (on mobile robots) already exist, none of them were available for the setup used in this thesis. And even though an extra, stationary sensor can be deployed for detecting and tracking people, this is not desirable. An important advantage of using only sensors present on the robot is that the users know when they are being observed and when not. If stationary sensors are used, users might feel violated in their privacy because they feel watched all the time. Especially when the sensors are deployed in the home of people, they might object to them. A test using a different setup, which could support the existing user trackers, or a decent user tracker could be conducted in future work. If the user can be tracked accurately, the predictions made by the AA and HMMA algorithms are more accurate and the performance of these algorithms will be improved. Furthermore, the user tracker and all algorithms that use it only can track one person at a time. The algorithms could all be extended to handle multiple people in the view of the robot. This will make the algorithms more robust in their intended deployment areas, where it is very likely more than one person can be in view at a time.

7.3 Improved gesture recognition performance

The GR systems proposed in this thesis show it is possible to easily classify the three gesture classes (`LEFT`, `RIGHT` and `STOP`) proposed in Section 4.1.1. Furthermore, feature based GR shows that a within-subject classification of gesture speed is possible too. However, the trajectory based GR could not classify the speed of a gesture correctly. By increasing the amount of training samples the recognition of the feature based classifiers improved, but this could also be the result of overfitting due to the small number of classes. Future research could therefore try to incrementally increase the interaction possibilities for ICP, starting with distinguishing the different speeds of the gestures. Also, new gesture classes can be added when ICP is extended towards a more teleoperation-like mechanism (controlling the robot directly). New gestures to set a new goal by pointing to an area could be added as well.

7.4 Shared autonomy

Another direction for further research is to improve the intelligence of the robot. In ICP, the user can indicate which way the robot should pass. The way point the user indicates could be set at an unreachable point. Currently, there is no control mechanism that will prevent this from happening. The robot will try to reach this point, and since it cannot be reached, get stuck until it receives another command (or a timeout cue is given). The main problem that has to be solved here is how the robot should handle these situations. There are several possibilities that come to mind. The robot could pick a point that is close to the intended way point and go there. Where this new way point should be relative

to the commanded point should be the main focus. Another option is to indicate to the user that the command given is invalid and that the user should give a new one. However, this solution may become time-consuming and tedious. A combination of both techniques could also be explored further.

The behavior of the robot when no command is given has to be explored as well. The robot could stop, or deploy one of the other obstacle avoidance mechanisms (but which one?). Figure 3.2 shows a possible hierarchy for this situation. However, this setup is not implemented yet, because of the limitation in the user tracker and gesture recognition modules. Once those module perform at an acceptable level, this structure of module could be explored further as well.

7.5 Other implementations of ICP

In the Introduction, the original idea behind ICP is explained. ICP makes use of subtle body language to predict the direction where the user is going. However, because of the limitations in various necessary modules mentioned earlier, this could not be implemented. Instead a more explicit interaction method was chosen, namely arm gestures. However, this interaction method can be implemented in different ways. For instance speech recognition could be used, possibly in combination with the gestures, for ICP as well.

Also, a specific module to detect the body language can be built. It would have to be able to detect the subtle cues while both the robot and the user are moving. This will be a very challenging research direction to go into. However, when it is accomplished this form of ICP will increase the user experience, because the communication between the robot and the user will be as natural as the communication between two people.

All in all, in a dynamic home-like environment, accurate prediction of the movements of an object will increase the efficiency of navigation of a mobile robot. First explorations in ICP show promising results and should therefore be researched in more detail.

Acknowledgments

First of all I want to thank my supervisors Louis Vuurpijl and Dietwig Lowet for helping me in my research and write my thesis. I want to thank Philips for providing the necessary hardware for my experiments and Frank van Heesch for repairing the robot on several occasions. Furthermore, I want to thank Mijke Burger for being my pilot test subject to test the real world implementations of my algorithms. Finally, I want to thank everyone who took the time to read and give feedback on my thesis, with a special thank you for Martijn van Otterlo.

Bibliography

[1] M. Beetz, U. Klank, A. Maldonado, D. Pangercic, and T. Rühr. Robotic roommates making pancakes-look into perception-manipulation loop. In *IEEE International Conference on Robotics and Automation (ICRA), Workshop on Mobile Manipulation: Integrating Perception and Manipulation*, pages 9–13, 2011.

[2] J. van den Berg, Stephen J. Guy, M. C. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *International Symposium on Robotics Research*, 2009.

[3] M.K. Bhuyan, D.R. Neog, and M.K. Kar. Hand pose recognition using geometric features. In *2011 National Conference on Communications (NCC)*, pages 1–5. IEEE, 2011.

[4] A. Bleiweiss, D. Eshar, G. Kutliroff, A. Lerner, Y. Oshrat, and Y. Yanai. Enhanced interactive gaming by blending full-body tracking and gesture animation. In *ACM SIGGRAPH ASIA 2010 Sketches*, page 34. ACM, 2010.

[5] R. A. Bolt. "Put-that-there": Voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '80, pages 262–270, New York, NY, USA, 1980. ACM.

[6] F. Bonin-Font, A. Ortiz, and G. Oliver. Visual navigation for mobile robots: A survey. *Journal of Intelligent & Robotic Systems*, 53:263–296, 2008.

[7] M.D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. van Gool. Online multiperson tracking-by-detection from a single, uncalibrated camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(9):1820 –1833, sept. 2011.

[8] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[9] V. Buchmann, S. Violich, M. Billinghurst, and A. Cockburn. FingARtips: gesture based direct manipulation in Augmented Reality. In *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE '04, pages 212–221, New York, NY, USA, 2004. ACM.

[10] D. Bullock and J. Zelek. Towards real-time 3-D monocular visual tracking of human limbs in unconstrained environments. *Real-Time Imaging*, 11(4):323–353, 2005.

[11] M. Chan, D. Estève, C. Escriba, and E. Campo. A review of smart homes: Present state and future challenges. *Computer Methods and Programs in Biomedicine*, 91(1):55–81, 2008.

[12] Y.H. Chen, C.H. Lu, K.C. Hsu, L.C. Fu, Y.J. Yeh, and L.C. Kuo. Preference model assisted activity recognition learning in a smart home environment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009.*, pages 4657–4662. IEEE, 2009.

[13] A. Corradini. Dynamic time warping for off-line recognition of a small gesture vocabulary. In *Proceedings. IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 2001.*, pages 82–89. IEEE, 2001.

[14] G. de Croon, E. de Weerdt, C. De Wagter, and B.D.W. Remes. The appearance variation cue for obstacle avoidance. In *2010 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1606–1611. IEEE, 2010.

[15] E. Demeester, M. Nuttin, D. Vanhooydonck, and H. Van Brussel. Fine motion planning for shared wheelchair control: Requirements and preliminary experiments. In *11th International Conference on Advanced Robotics (ICAR) 2003*, pages 1278–1283, 2003.

[16] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[17] M. Dimitrijevic, V. Lepetit, and P. Fua. Human body pose recognition using spatio-temporal templates. In *ICCV workshop on Modeling People and Human Interaction*, volume 2, 2005.

[18] A. Dix. *Human-computer interaction*. Prentice hall, 2004.

[19] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):743 –761, april 2012.

[20] W. Du and H. Li. Vision based gesture recognition system with single camera. In *5th International Conference on Signal Processing Proceedings, 2000. WCCC-ICSP 2000.*, volume 2, pages 1351 –1357 vol.2, 2000.

[21] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[22] M. Elmezain, A. Al-Hamadi, and B. Michaelis. Hand trajectory-based gesture spotting and recognition using HMM. In *16th IEEE International Conference on Image Processing (ICIP), 2009*, pages 3577–3580. IEEE, 2009.

[23] M. Fiala and A. Ufkes. Visual Odometry Using 3-Dimensional Video Input. *Computer and Robot Vision, Canadian Conference*, 0:86–93, 2011.

[24] P. Fiorini and Z. Shillert. Motion Planning in Dynamic Environments using Velocity Obstacles. *International Journal of Robotics Research*, 17:760–772, 1998.

[25] J. Forlizzi and C. DiSalvo. Service robots in the domestic environment: a study of the roomba vacuum in the home. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 258–265. ACM, 2006.

[26] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *Robotics & Automation Magazine, IEEE*, 4(1):23–33, 1997.

[27] C. Fulgenzi, A. Spalanzani, and C. Laugier. Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid. In *In Proceedings of the IEEE International Conference onf Robotics and Automation*, pages 1610–1616, 2007.

[28] P. Garg, N. Aggarwal, and S. Sofat. Vision based hand gesture recognition. *World Academy of Science, Engineering and Technology*, 49:972–977, 2009.

[29] B. Gerkey and K. Conley. Robot Developer Kits [ROS Topics]. *Robotics & Automation Magazine, IEEE*, 18(3):16–16, 2011.

[30] I. González, M.F. Ndez, M. Fernández, J. Maestre, et al. *Service Robotics Within the Digital Home: Applications and Future Prospects*, volume 53. Springer Verlag, 2011.

[31] E. T. Hall. *The Hidden Dimension*. Doubleday, Garden City, 1st edition, 1966.

[32] C. Harshith, K. R. Shastry, M. Ravindran, M. V. V. N. S. Srikanth, and N. Lakshmikhanth. Survey on various gesture recognition techniques for interfacing machines based on ambient intelligence. *International Journal of Computer Science & Engineering Survey*, abs/1012.0084, 2010.

[33] D. Hennes, D. Claes, W. Meeussen, K. Tuyls, H.B. Ammar, M.E. Taylor, K. Tuyls, G. Weiss, H.B. Ammar, M.E. Taylor, et al. Multi-robot collision avoidance with localization uncertainty. In *Proceedings of 11th International Conference on Adaptive Agents and Multi-agent Systems (AAMAS 2012)*, pages 593–600. Springer-Verlag, 2012.

[34] M. Isken, B. Vester, T. Frenken, E.E. Steen, M. Brell, and A. Hein. Enhancing mobile robots's navigation through mobility assessments in domestic environments. *Ambient Assisted Living*, page 223, 2011.

[35] H. Iwata and S. Sugano. Design of human symbiotic robot TWENDY-ONE. In *IEEE International Conference on Robotics and Automation, 2009. ICRA'09.*, pages 580–586. IEEE, 2009.

[36] S. Jamieson. Likert scales: how to (ab)use them. *Medical Education*, 38(12):1217–1218, 2004.

[37] Y.M. Jeong, K.T. Lim, and S.E. Lee. mGlove: Enhancing User Experience through Hand Gesture Recognition. *Advances in Electronic Engineering, Communication and Management Vol. 1*, pages 383–386, 2012.

[38] K. Kinugawa and H. Noborio. A shared autonomy of multiple mobile robots in teleoperation. In *10th IEEE International Workshop on Robot and Human Interactive Communication, 2001. Proceedings.*, pages 319–325. IEEE, 2001.

[39] B. Kluge and E. Prassler. Reflective navigation: individual behaviors and group behaviors. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 4172 – 4177, 2004.

[40] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004*, volume 3, pages 2149–2154. IEEE, 2004.

[41] T. Kruse, A. Kirsch, E.A. Sisbot, and R. Alami. Exploiting human cooperation in human-centered robot navigation. In *RO-MAN, 2010 IEEE*, pages 192–197. IEEE, 2010.

[42] Chi-Pang Lam, Chen-Tun Chou, Kuo-Hung Chiang, and Li-Chen Fu. Human-centered robot navigation: Towards a harmoniously human-robot coexisting environment. *IEEE Transactions on Robotics*, 27(1):99 – 112, 2011.

[43] D. Makris and T. Ellis. Path detection in video surveillance. *Image and Vision Computing*, 20(12):895–903, 2002.

[44] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige. The office marathon: Robust navigation in an indoor office environment. In *IEEE International Conference on Robotics and Automation (ICRA), 2010*, pages 300–307. IEEE, 2010.

[45] C. McCarthy, N. Barnes, and R. Mahony. A robust docking strategy for a mobile robot using flow field divergence. *IEEE Transactions on Robotics*, 24(4):832–842, 2008.

[46] J. Meyer, M. Brell, A. Hein, and Gessler S. Personal Assistive Robots for AAL Services at Home - The Florence Point of View. `http://www.hitech-projects.com/euprojects/florence/project-overview.html`, 2009.

[47] Erik G. Miller, Nicholas E. Matsakis, and Paul A. Viola. Learning from one example through shared densities on transforms. In *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 464–471, 2000.

[48] S. Mitra and T. Acharya. Gesture recognition: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(3):311–324, 2007.

[49] T.B. Moeslund, A. Hilton, and V. Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding*, 104(2):90–126, 2006.

[50] P. Nadrag, L. Temzi, H. Arioui, and P. Hoppenot. Remote control of an assistive robot using force feedback. In *15th International Conference on Advanced Robotics (ICAR)*, pages 211–216, 2011.

[51] C. Nakajima, M. Pontil, and T. Poggio. People recognition and pose estimation in image sequences. In *International Joint Conference on Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS*, volume 4, pages 189–194. IEEE, 2000.

[52] R. Niels, D. Willems, and L. Vuurpijl. The NicIcon database of handwritten icons. In *11th International Conference on the Frontiers of Handwriting Recognition (ICFHR 2008)*, pages 296–301, Montréal, Canada, August 2008.

[53] G. Norman. Likert scales, levels of measurement and the laws of statistics. *Advances in health sciences education*, 15(5):625–632, 2010.

[54] T. Odashima, M. Onishi, K. Tahara, K. Takagi, F. Asano, Y. Kato, H. Nakashima, Y. Kobayashi, T. Mukai, Z. Luo, and S. Hosoe. A Soft Human-Interactive Robot RI-MAN. In *International Conference on Intelligent Robots and Systems, 2006 IEEE/RSJ*, page 1, oct. 2006.

[55] S. Oh, Y. Lee, K. Hong, K. Kim, and K. Jung. View-point insensitive human pose recognition using neural network. In *Proceedings of World Academy of Science, Engineering and Technology*, volume 34, pages 289–292, 2008.

[56] S. Oviatt, Ph. Cohen, L. Wu, J. Vergo, L. Duncan, B. Suhm, J. Bers, T. Holzman, T. Winograd, J. Landay, J. Larson, and D. Ferro. Designing the user interface for multimodal speech and pen-based gesture applications: state-of-the-art systems and future research directions. *Human-Computer Interaction*, 15(4):263–322, December 2000.

[57] E. Pacchierotti, H.I. Christensen, and P. Jensfelt. Evaluation of passing distance for social robots. In *The 15th IEEE International Symposium on Robot and Human Interactive Communication, 2006. ROMAN 2006.*, pages 315 – 320, 2006.

[58] C. Pantofaru, L. Takayama, T. Foote, and B. Soto. Exploring the role of robots in home organization. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, HRI '12, pages 327–334, New York, NY, USA, 2012. ACM.

[59] C.B. Park and S.W. Lee. Real-time 3D pointing gesture recognition for mobile robots with cascade HMM and particle filter. *Image and Vision Computing*, 29(1):51–63, 2011.

[60] V.I. Pavlovic, R. Sharma, and T.S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, 1997.

[61] B. Pitzer, M. Styer, C. Bersch, C. DuHadway, and J. Becker. Towards perceptual shared autonomy for robotic mobile manipulation. In *IEEE International Conference on Robotics and Automation (ICRA), 2011*, pages 6245–6251. IEEE, 2011.

[62] R. Plamondon. A kinematic theory of rapid human movements. *Biological Cybernetics*, 72:309–320, 1995.

[63] R. W. Poppe. Evaluating Example-based Pose Estimation: Experiments on the HumanEva Sets. Technical Report TR-CTIT-07-72, Centre for Telematics and Information Technology University of Twente, Enschede, October 2007.

[64] P. Prévost, I. Yuri, G. Renato, and B. Alain. Spatial invariance in anticipatory orienting behaviour during human navigation. *Neuroscience letters*, 339(3):243–247, 2003.

[65] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[66] J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *26th IEEE Symposium on the Foundation of Computer Science*, pages 144–153, 1985.

[67] Z. Ren, J. Meng, J. Yuan, and Z. Zhang. Robust hand gesture recognition with Kinect sensor. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 759–760. ACM, 2011.

[68] J. Richarz, C. Martin, A. Scheidig, and H.M. Gross. There you go!-estimating pointing gestures in monocular images for mobile robot instruction. In *The 15th IEEE International Symposium on Robot and Human Interactive Communication, 2006. ROMAN 2006.*, pages 546–551. IEEE, 2006.

[69] Y. Rogers, H. Sharp, and J. Preece. *Interaction design: beyond human-computer interaction*. Wiley Publishing, 2011.

[70] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent ASIMO: System overview and integration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002.*, volume 3, pages 2478–2483. IEEE, 2002.

[71] M. Salem, S. Kopp, S. Wachsmuth, K. Rohlfing, and F. Joublin. Generation and evaluation of communicative robot gesture. *International Journal of Social Robotics*, 4(2):201–217, 2012.

[72] T. Sasaki, D. Brscic, and H. Hashimoto. Human-observation-based extraction of path patterns for mobile robot navigation. *IEEE Transactions on Industrial Electronics*, 57(4):1401 –1410, april 2010.

[73] T. Schlömer, B. Poppinga, N. Henze, and S. Boll. Gesture recognition with a Wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, TEI '08, pages 11–14, New York, NY, USA, 2008. ACM.

[74] S.A.H. Shah, A. Ahmed, I. Mahmood, and K. Khurshid. Hand gesture based user interface for computer using a camera and projector. In *IEEE International Conference on Signal and Image Processing Applications (ICSIPA), 2011*, pages 168 –173, nov. 2011.

[75] H. Sidenbladh, M. Black, and D. Fleet. Stochastic tracking of 3D human figures using 2D image motion. *Computer Vision ECCV 2000*, pages 702–718, 2000.

[76] E.A. Sisbot, L.F. Marin-Urias, R. Alami, and T. Simeon. A human aware mobile robot motion planner. *IEEE Transactions on Robotics*, 23(5):874–883, 2007.

[77] Y. Song, D. Demirdjian, and R. Davis. Continuous body and hand gesture recognition for natural human-computer interaction. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2(1):5, 2012.

[78] H.I. Suk, B.K. Sin, and S.W. Lee. Hand gesture recognition based on dynamic Bayesian network framework. *Pattern Recognition*, 43(9):3059–3072, 2010.

[79] M. Svenstrup, S.T. Hansen, H.J. Andersen, and T. Bak. Adaptive human-aware robot navigation in close proximity to humans. *International Journal of Advanced Robotic Systems*, 8(2):1, 2011.

[80] P. Tandler and T. Prante. Using incremental gesture recognition to provide immediate feedback while drawing pen gestures. In *14th Annual ACM Symposium on User Interface Software and Technology (UIST 2001)*, pages 11–14, 2001.

[81] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, San Francisco, CA, 2011.

[82] S. Thompson, T. Horiuchi, and S. Kagami. A probabilistic model of human motion and navigation intent for mobile robot path planning. In *4th International Conference on Autonomous Robots and Agents, 2009. ICARA 2009.*, pages 663–668. IEEE, 2009.

[83] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. Intelligent robotics and autonomous agents. MIT Press, 2005.

[84] R. Timofte, M. Mathias, R. Benenson, and L. van Gool. Pedestrian detection at 100 frames per second. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 0:2903–2910, 2012.

[85] A. Uribe, S. Alves, J.M. Rosario, B. Perez-Gutierrez, et al. Mobile robotic teleoperation using gesture-based human interfaces. In *Robotics Symposium, 2011 IEEE IX Latin American and IEEE Colombian Conference on Automatic Control and Industry Applications (LARC)*, pages 1–6. IEEE, 2011.

[86] M. van den Bergh, D. Carton, R. De Nijs, N. Mitsou, C. Landsiedel, K. Kuehnlenz, D. Wollherr, L. van Gool, and M. Buss. Real-time 3D hand gesture interaction with a robot for understanding directions from humans. In *RO-MAN, 2011 IEEE*, pages 357–362. IEEE, 2011.

[87] D. Vanhooydonck, E. Demeester, A. Hüntemann, J. Philips, G. Vanacker, H. Van Brussel, and M. Nuttin. Adaptable navigational assistance for intelligent wheelchairs by means of an implicit personalized user model. *Robotics and Autonomous Systems*, 58(8):963–977, 2010.

[88] N. Villaroman, D. Rowe, and B. Swan. Teaching natural user interaction using openni and the microsoft kinect sensor. In *Proceedings of the 2011 conference on Information technology education*, SIGITE '11, pages 227–232, New York, 2011. ACM.

[89] D. Willems, R. Niels, M. van Gerven, and L. Vuurpijl. Iconic and multi-stroke gesture recognition. *Pattern Recognition*, 42(12):3303–3312, 2009.

[90] D.J.M. Willems and L.G. Vuurpijl. Pen gestures in online map and photograph annotation tasks. In *Proceedings of the tenth International Workshop on Frontiers in Handwriting Recognition*, pages 397–402, 2006.

[91] Matthias Wölfel. Kinetic space. User manual, `http://code.google.com/p/kineticspace/`, Januari 2011.

[92] S. Yun, C.G. Kim, M. Kim, and M.T. Choi. Robust robot's attention for human based on the multi-modal sensor and robot behavior. In *IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO), 2010*, pages 117–122. IEEE, 2010.

[93] X. Zabulis, H. Baltzakis, and A. Argyros. Vision-based Hand Gesture Recognition for Human-Computer Interaction. In *The Universal Access Handbook*, Human Factors and Ergonomics, pages 34.1–34.30. Lawrence Erlbaum Associates, Inc., June 2009.

[94] Z. Zafrulla, H. Brashear, T. Starner, H. Hamilton, and P. Presti. American sign language recognition with the Kinect. In *Proceedings of the 13th international conference on multimodal interfaces*, pages 279–286. ACM, 2011.

Appendix A

Data

A.1 Simulations

					95% Conf. interval	
		N	Mean	Std. dev	Lower bound	Upper bound
NONE	BASE	20	15.36	0.02	15.35	15.37
	AA	20	15.37	0.03	15.35	15.38
	HMMA	20	15.36	0.02	15.35	15.37
STAT	BASE	20	16.69	1.52	15.98	17.40
	AA	19	17.33	2.36	16.19	18.47
	HMMA	20	16.21	0.84	15.82	16.61
DYN1	BASE	19	20.07	0.83	19.67	20.46
	AA	17	21.27	6.51	17.92	24.61
	HMMA	19	21.68	3.72	19.88	23.47
DYN2	BASE	16	15.79	0.29	15.64	15.94
	AA	17	15.72	0.16	15.63	15.80
	HMMA	19	17.27	7.23	13.78	20.76
DYN3	BASE	1	76.76	-	-	-
	AA	12	23.28	8.80	17.69	28.87
	HMMA	19	17.71	0.24	17.60	17.83
ALL	BASE	76	17.79	7.15	16.15	19.42
	AA	85	18.17	5.32	17.02	19.32
	HMMA	97	17.61	4.16	16.77	18.45

Table A.1: *The results of the simulations of time.*

					95% Conf. interval	
		N	Mean	Std. dev	Lower bound	Upper bound
NONE	BASE	20	0.87	0.72	0.53	1.20
	AA	20	0.80	0.72	0.46	1.13
	HMMA	20	0.87	0.71	0.53	1.20
STAT	BASE	20	1.03	0.56	0.77	1.30
	AA	19	1.08	0.55	0.82	1.35
	HMMA	20	1.08	0.55	0.83	1.34
DYN1	BASE	19	1.00	0.74	0.64	1.36
	AA	17	2.20	2.90	0.71	3.69
	HMMA	19	2.72	1.10	2.20	3.25
DYN2	BASE	16	1.11	0.72	0.73	1.49
	AA	17	0.97	0.75	0.58	1.35
	HMMA	19	0.89	0.71	0.55	1.23
DYN3	BASE	1	8.13	-	-	-
	AA	12	1.91	0.69	1.47	2.34
	HMMA	19	1.30	0.72	0.95	1.65
ALL	BASE	76	1.09	1.06	0.85	1.33
	AA	85	1.33	1.50	1.01	1.66
	HMMA	97	1.37	1.03	1.16	1.57

Table A.2: *The results of the simulations of detour.*

					95% Conf. interval	
		N	Mean	Std. dev	Lower bound	Upper bound
DYN1	BASE	18	20.00	0.36	19.82	20.18
	AA	17	32.13	12.19	25.87	38.40
	HMMA	19	30.19	6.28	27.16	33.22
DYN2	BASE	19	16.51	1.76	15.66	17.35
	AA	19	15.63	0.08	15.60	15.67
	HMMA	18	15.54	0.08	15.51	15.58
DYN3	BASE	4	15.89	0.49	15.11	16.68
	AA	18	22.25	9.54	17.51	26.99
	HMMA	19	18.12	0.71	17.77	18.46

(a) *Time*

					95% Conf. interval	
		N	Mean	Std. dev	Lower bound	Upper bound
DYN1	BASE	19	1.73	0.12	1.67	1.79
	AA	17	3.06	2.10	1.98	4.14
	HMMA	19	2.42	0.70	2.08	2.75
DYN2	BASE	16	1.76	0.27	1.63	1.89
	AA	17	1.63	0.02	1.62	1.64
	HMMA	19	1.60	0.02	1.59	1.61
DYN3	BASE	1	1.63	0.07	1.51	1.74
	AA	12	2.70	1.06	2.18	3.23
	HMMA	19	2.25	0.21	2.15	2.34

(b) *Detour*

Table A.3: *The results of the simulations of time (a) and detour (b) of the trial run in the complex environment.*

A.2 Real-world experiments

					95% Conf. interval	
		N	Mean	Std. dev	Lower bound	Upper bound
STAT	BASE	5	17.63	2.93	-	-
	AA	5	18.24	3.32	-	-
DYN1	BASE	5	23.87	7.75	14.24	33.49
	AA	5	18.34	2.21	15.60	21.08
	HMMA	5	22.94	7.18	14.01	31.86
DYN2	BASE	5	16.59	0.67	15.76	17.42
	AA	4	19.28	4.75	11.72	26.85
	HMMA	5	25.18	5.57	18.26	32.09
DYN3	BASE	5	24.55	5.60	17.59	31.50
	AA	5	30.11	11.30	16.08	44.14
	HMMA	4	19.41	2.82	14.93	23.90

(a) *Time*

					95% Conf. interval	
		N	Mean	Std. dev	Lower bound	Upper bound
STAT	BASE	5	2.21	0.21	-	-
	AA	5	2.25	0.13	-	-
DYN1	BASE	5	2.54	0.27	2.20	2.88
	AA	5	2.57	0.35	2.13	3.00
	HMMA	5	2.59	0.18	2.37	2.82
DYN2	BASE	5	2.66	0.28	2.31	3.00
	AA	4	2.51	0.59	1.57	3.45
	HMMA	5	2.68	0.16	2.48	2.87
DYN3	BASE	5	2.60	0.24	2.30	2.90
	AA	5	2.69	0.23	2.41	2.97
	HMMA	4	2.58	0.19	2.47	2.70

(b) *Detour*

Table A.4: *The results of the user tests of time (a) and detour (b).*

	Condition	Mode	N	Mean	Std. dev
Corrected Time	NONE	Sim	9	1.92	.003
		RW	5	2.61	.180
	STAT	Sim	10	2.24	.164
		RW	5	3.10	1.018
Corrected Detour	NONE	Sim	9	0.01	.001
		RW	5	0.37	.036
	STAT	Sim	10	0.06	.014
		RW	5	0.38	.088

Table A.5: *The means and standard deviation of both **time** and **detour** in both the simulations and the user tests. The time and detour is corrected for the straight line distance to the goal. Mode indicates whether the data is acquired in the simulations (Sim) or in the real world (RW).*