Autonomous Operation and Human-Robot Interaction on an Indoor Mobile Robot

by

Callum J. Robinson

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Engineering
in Electronic and Computer Systems Engineering.

Victoria University of Wellington
2016

Abstract

MARVIN (Mobile Autonomous Robotic Vehicle for Indoor Navigation) was once the flagship of Victoria University's mobile robotic fleet. However, over the years MARVIN has become obsolete. This thesis continues the the redevelopment of MARVIN, transforming it into a fully autonomous research platform for human-robot interaction (HRI).

MARVIN utilises a Segway RMP, a self balancing mobility platform. This provides agile locomotion, but increases sensor processing complexity due to its dynamic pitch. MARVIN's existing sensing systems (including a laser rangefinder and ultrasonic sensors) are augmented with tactile sensors and a Microsoft Kinect v2 RGB-D camera for 3D sensing. This allows the detection of the obstacles often found in MARVIN's unmodified office-like operating environment.

These sensors are processed using novel techniques to account for the Segway's dynamic pitch. A newly developed navigation stack takes the processed sensor data to facilitate localisation, obstacle detection and motion planning.

MARVIN's inherited humanoid robotic torso is augmented with a touch screen and voice interface, enabling HRI. MARVIN's HRI capabilities are demonstrated by implementing it as a robotic guide. This implementation is evaluated through a usability study and found to be successful.

Through evaluations of MARVIN's locomotion, sensing, localisation and motion planning systems, in addition to the usability study, MARVIN is found to be capable of both autonomous navigation and engaging HRI. These developed features open a diverse range of research directions and HRI tasks that MARVIN can be used to explore.

Acknowledgments

I would like to express my deepest gratitude to all those who have offered me support, guidance and encouragement throughout this last year.

To Professor Dale Carnegie, my project supervisor. I am thankful for his endless support and enthusiastic guidance throughout this project. He always made time for me, even when there was none to give. I am especially grateful for his tireless efforts proofing this thesis.

To Dr Will Brown, for his guidance through the second half of this project. His kind words of encouragement motivated me to strive for success, particularly during the final crunch.

A big thanks must go to the ECS technicians, Tim Exley, Jason Edwards and James McVay, for their technical assistance, support and patience throughout the year. Their efforts made this project possible.

I would like to thank my fellow lab mates for providing a fun and supportive work environment. Especially my fellow masters student and friend, Lance Molyneaux, who provided many interesting conversations and a great source of ideas.

To my parents Ann and Alan. Without their constant support and encouragement I could not have made it this far. Special thanks to Ann for her brave work proofing the first draft of all my chapters.

Thank you to my family, especially my sister Kat. Their words of encouragement and offers of support helped me make it through this last year.

Thank you all.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

MARVIN was once the flagship of Victoria University's mobile robotic fleet. However, with the Mechatronics Group's focus shifting to Urban Search and Rescue (USAR), MARVIN's development slowed, with the last hardware upgrade performed in 2010. Since then, many of MARVIN's systems had become obsolete. As an attempt to remedy this and revitalise MARVIN, it went through a redevelopment in 2014. During this redevelopment, MARVIN's mobility platform, sensors and control software were replaced and upgraded. This thesis expands upon this redevelopment, bringing MARVIN back to relevancy.

MARVIN was originally developed as an autonomous mobile security device that would patrol the University's corridors, detecting and recording the activities of intruders, and recharging itself when necessary [7]. These goals were never successfully realised. Since then, this objective has expanded to include public relations, interacting with visitors or acting as a guide. To achieve these goals, MARVIN's development must combine the fields of autonomous mobile robots (AMRs) and human-robot interaction (HRI).

An autonomous mobile robot is a self contained system capable of moving in its environment and performing tasks or behaviours with a high level of autonomy. To facilitate this, an AMR must be capable of locomotion, sensing, localisation and motion planning.

Human-robot interaction is a multidisciplinary field combining human-machine interfaces (HMI) and robotics. To enable HRI, a robot must be able to communicate with humans in order to receive requests and provide responses. This is achieved through a combination of receptive and expressive elements. Receptive elements typically include: touch screens, keyboards, buttons, voice recognition and sensors (such as cameras for human detection and face tracking). Expressive elements typically include: LCD screens, LED matrices, lights, speech synthesis and actuation (such as the robotic humanoid torso used in this project).

1.1 Motivation

Robots are becoming more common place everyday. They are no longer thought of as just industrial machines confined to factories. As they become more common place, robots need to be integrated into society. A large part of this process is human-robot interaction. This creates a desire to perform further research into HRI.

There are a number of available robotic research platforms capable of complex autonomous tasks. Rethink Robotics manufacture Baxter [8], a robotic torso with dual manipulators designed specifically to cooperate with humans in the work environment. It learns how to conduct tasks by watching them being performed. Baxter uses an LCD screen to emulate facial expressions and can communicate through voice. However, Baxter has no locomotion capabilities, limited sensors and costs 25,000 USD (~38,000

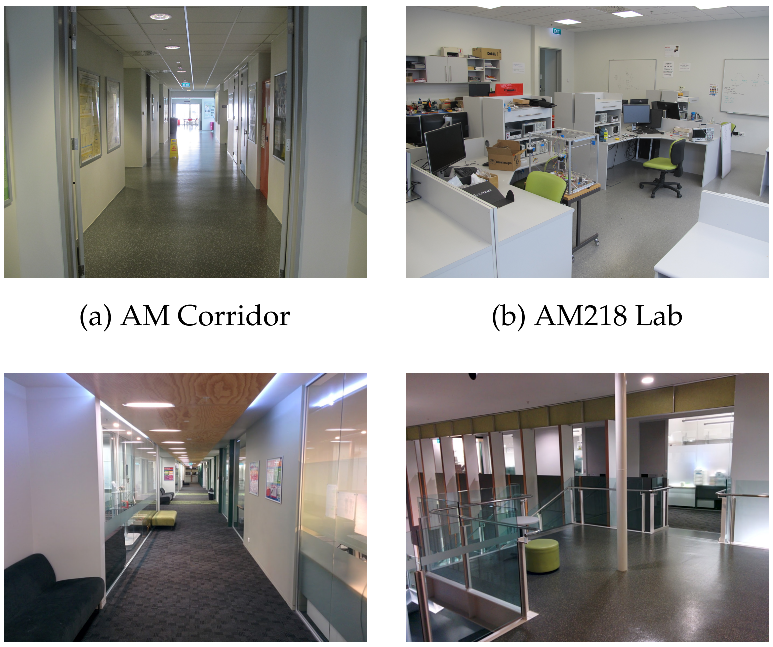
NZD). Willow Garage manufacture the PR2 [9], a mobile robot with dual manipulator arms and a full suite of 3D sensors for autonomous navigation. However it has no receptive or expressive elements and costs 280,000 USD ($\sim$420,000 NZD). Both these systems are between 1.6 m and 1.8 m tall making them a similar height to the average human (NZ males 1.77 m, NZ females 1.64 m [10]), which is appropriate for HRI. Their dual manipulating arms could be used as expressive elements or to perform tasks such as opening doors. However, as they are not required for the majority of the predicted tasks (as outlined in section 1.1.2), this unnecessarily inflates the price of these platforms. There are also a plethora of smaller and more affordable AMR research platforms (such as the Adept Robotics Pioneer [11]), but very few with human-robot interaction capabilities. They are also typically short (1 m), which makes them less appropriate for HRI. The majority of HRI research platforms are developed in-house (either fully, or based off an AMR research platform) and are not available for purchase, for example [12, 13, 14, 15, 16, 1]. This motivates the in-house development of a research platform, which aligns with MARVIN's current redevelopment.

To develop MARVIN into a capable research platform for HRI, the operating environment and potential future research directions must be understood. Section 1.1.1 outlines MARVIN's operating environment and section 1.1.2 suggests some of the possible tasks that might be achieved with the developed research platform in the future. MARVIN is currently missing the control architecture and navigation software to allow fully autonomous operation. It is also lacking the sensing, receptive and expressive elements required for human-robot interaction. This thesis discusses the development of these features, enabling MARVIN's transformation into a fully autonomous mobile research platform for HRI.

1.1.1 Operating Environment

MARVIN will primarily be operating in the second floor corridors of the Cotton (CO) and Alan MacDiarmid (AM) buildings in the Kelburn campus of Victoria University of Wellington. These areas are not specifically designed to accommodate robot operation and cannot be modified.

Both localisation and obstacle detection rely on the robot's ability to sense its environment. MARVIN's office-like operating environment presents a number of sensory challenges. These challenges can be seen in the typical areas shown by figure 1.1.



(a) AM Corridor (b) AM218 Lab

(c) CO Corridor (d) AM/CO Foyer

Figure 1.1: Typical areas in the operating environment.

Surface materials that must be detected include: opaque walls, translucent (glass) walls, carpeted floors and vinyl floors. Glass walls are often invisible to optical based sensors, such as laser rangefinders, depth cameras and RGB cameras. The vinyl floors can cause reflections (as seen in figure 1.1a), which can also cause issues for optical sensors.

Obstacles that must be detected include: office furniture (such as, desks, benches, tables, chairs, stools and couches), thin obstacles (such as poles), glass banisters, stairways, free space (areas where there is no floor, such as in figure 1.1d), miscellaneous small objects found in labs and offices (such as rubbish bins) and people. Many of these obstacles have complex 3D shapes and differing heights which might not be fully detected by 2D sensors such as laser rangefinders. This is further discussed in 3.2.1.

The localisation problem can be simplified by modifying the environment through the addition of unique landmarks, which the robot can localise against. However, this results in the robot only being able to localise in specific predefined areas. A more general solution is desired that does not involve the modification of the operating environment. This will improve the versatility of the developed research platform.

1.1.2 Possible Tasks

To develop a successful research platform it is important to understand the types of tasks it might perform. As a research platform for human-robot interaction, MARVIN may perform tasks including: guiding people around the University Library to find areas or particular books, guiding new students and visitors around the University campus, acting as an information point for students and visitors or as a robotic security guard autonomously patrolling the University corridors.

To perform these tasks, MARVIN must have appropriate receptive and expressive elements for human-robot interaction and sufficient sensing and control systems for autonomous navigation in the operating environment outlined in section 1.1.1.

1.2 Aims and Objectives

The aim of this project is to transform the existing MARVIN system into a mobile autonomous research platform for human-robot interaction. This involves the combination of three primary objectives. First, MARVIN's autonomous navigation functionality must be extended. This includes: locomotion, sensing, localisation and motion planning, as detailed in section 1.2.1. Second, MARVIN's HRI capabilities must be developed. This involves the addition of both receptive and expressive elements, as discussed in section 1.2.2. Third, any health and safety concerns about the autonomous operation of MARVIN must be addressed, as explained in section 1.2.3. A summary of the three primary objectives and their related sub-objectives is presented in section 1.2.4.

1.2.1 Autonomous Navigation

As an autonomous mobile robot, MARVIN must be capable of: locomotion, sensing, localisation and motion planning. This section discusses the requirements of these functions.

Locomotion allows MARVIN to move through its environment. This is achieved through the inherited Segway RMP, further explained in section

3.1.1. Because the Segway RMP is a self balancing mobility platform, it must be characterised to determine safe operating conditions once all MARVIN's subsystems have been mounted on it (the additional weight from the mounted hardware will change the response of the Segway RMP).

Sensing provides MARVIN with information about its environment. As explained in section 1.1.1, MARVIN will operate in an unmodified office-like environment. This environment will contain obstacles with a range of differing heights and 3D structures. Many of these obstacles will not be detected by the inherited sensors, further explained in section 3.1.2. Because of this, MARVIN requires the addition of a 3D sensor for the detection of obstacles likely to be found in an office-like environment. As each of MARVIN's sensing systems will detect different surfaces and obstacles, sensor fusion will be required to enable consistent obstacle detection.

Localisation provides MARVIN with a pose (position and heading) estimate. As a fully autonomous robot, MARVIN must be able to perform high level tasks independently, without external help. This requires MARVIN to be aware of its position throughout its autonomous operation. To achieve this, a localisation system must be implemented that is capable of operating in unmodified office-like environments.

Motion planning takes the sensor and localisation data to generate movement commands for the locomotion system. MARVIN must be capable of making high level plans from its current position to any arbitrary achievable goal. While following this high level plan, MARVIN must also be capable of generating low level plans to avoid any detected obstacles.

1.2.2 Human-Robot Interaction

To facilitate human-robot interaction, MARVIN must be capable of detecting humans and tracking their location. The implemented sensors must be processed to enable this functionality.

MARVIN has inherited a robotic humanoid torso. The purpose of this torso is to emulate basic emotions. The high level control of this torso should be integrated with the human detection and other HRI elements to enable MARVIN to interactively change its pose in reaction to humans movement.

MARVIN requires the development of both receptive and expressive elements to facilitate its use as a HRI research platform.

1.2.3 Health and Safety

MARVIN is a large (~ 1.8 m tall) heavy (~ 90 kg) self balancing robot, operating in close proximity with humans. All reasonable steps must be taken to ensure people's safety, whether they are onlookers or directly interacting with MARVIN.

MARVIN's velocity must not exceed a safe speed. The velocity should be limited to the average walking speed of a human, which is 1.41 ms^{-1} [17]. This velocity limit should be reduced if MARVIN's current environment requires it (for example operating in confined spaces, such as cluttered offices or narrow corridors). These limits should be determined by the characterisation of the Segway RMP, as per section 1.2.1.

Redundant sensors should be implemented to reduce the risk of collision due to a human remaining undetected. This should include a combination of range and tactile sensors to further reduce this risk.

Additional control systems should be implemented to monitor the commands sent to the locomotion system. This should be used to prevent unauthorised control sources from commanding movement and to prevent unsafe commands from being executed. Reactive control should be implemented to command an immediate stop if a collision is detected, in addition to the motion planning outlined in section 1.2.1.

1.2.4 Summary of Objectives

In summary, for this project to be considered successful the following objectives must be met:

1. **Autonomous Navigation**
   (a) Characterise the Segway RMP to determine safe operating conditions.
   (b) Implement a 3D sensor for the detection of obstacles in an office-like environment.
   (c) Implement sensor fusion to enable coherent obstacle detection.
   (d) Implement a localisation system to maintain pose estimates in an unmodified office-like environment.
   (e) Implement motion planning for high level goals and low level obstacle avoidance.
2. **Human-Robot Interaction**
   (a) Detect and track humans to enable HRI.
   (b) Develop interactive pose control for the inherited robotic humanoid torso.

   (c) Implement receptive elements for receiving requests from humans.

   (d) Implement expressive elements for providing responses and other information to humans.

3. **Health and Safety**

   (a) Limit MARVIN's maximum velocity should be limited to a safe speed, not exceeding the average human's walking speed (1.41 ms^{-1}) [17].

   (b) Implement additional sensing systems capable of detecting humans.

   (c) Monitor and control MARVIN's mobility platform to ensure safe operation.

1.3 Thesis Overview

This thesis is presented as follows:

Chapter 2 Presents several examples of existing autonomous mobile robots with a focus on human-robot interaction. It analyses their locomotion, sensing, localisation, motion planning and human-robot interaction capabilities. From this analysis conclusions are drawn and a list of specifications are made.

Chapter 3	Introduces MARVIN's inherited hardware platform and discusses the augmentations made to meet the objectives outlined in section 1.2. This includes the mobility platform, sensing systems, control electronics and receptive and expressive elements for human-robot interaction.
Chapter 4	Discusses the steps taken to process the sensor data into a form usable by the navigation stack and human-robot interaction. This includes the processing of laser range finder data and depth images.
Chapter 5	Details MARVIN's navigation stack. This includes localisation, obstacle detection and motion planning.
Chapter 6	Describes MARVIN's control architecture. This encompasses the entire control hierarchy: from hardware interfaces to the high level decision making.
Chapter 7	Explores the process of developing a human-robot interaction application on MARVIN. This is used as a method for evaluating MARVIN as a research platform. MARVIN's human-robot interaction abilities are evaluated through a usability study.
Chapter 8	Presents the results of the Segway characterisation, navigation stack evaluation and human detection analysis.
Chapter 9	Concludes the project with recommendations for future work and a summary of achievements.

The software Chapters 4, 5, 6, and 7 use a number of block diagrams to explain how various software systems work and communicate with one another. These block diagrams follow the convention shown in figure 1.2.

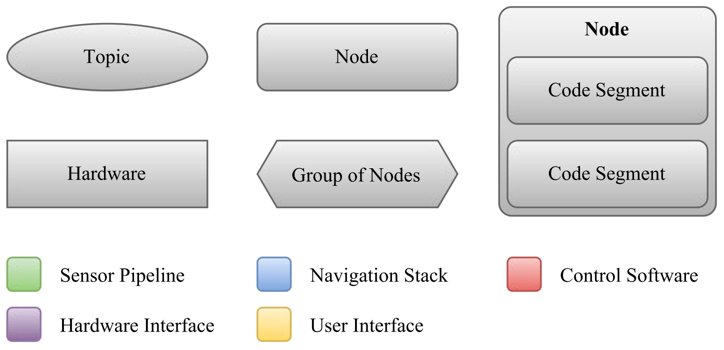


Figure 1.2: Block diagram convention used in Chapters 4, 5, 6, and 7.

- **Topic:** The message topics used by ROS to communicate between nodes.
- **Hardware:** The physical hardware units which make up MARVIN.
- **Node:** Individual software systems in ROS are called nodes.
- **Group of Nodes:** Used to describe a group of nodes which work together. For example, the navigation stack.
- **Node Split into Code Segments:** Used to explain the inner workings of a node.

Chapter 2

Background

As discussed in Chapter 1, the aim of this project is to transform MARVIN into a research platform for human-robot interaction. For this aim to be met, MARVIN's autonomous navigation and HRI capabilities must be developed. To inform the development of these systems, existing autonomous mobile robots with a focus on HRI are researched. This chapter presents this research.

Over the past two decades, a large number of mobile autonomous robots have been developed with a focus on human-robot interaction. These systems are often designed as robotic guides for museums [12, 13, 14, 15, 16] or as research platforms for universities and other research institutes [1]. However, no matter which task they are designed to perform, they all have the same basic functionality. As explained by Siegwart and Nourbakhsh [18], to enable autonomous navigation a robot must be capable of locomotion, sensing, localisation and motion planning. To facilitate HRI a robot must be capable of communicating with humans. This requires receptive and expressive elements.

Due to this list of necessary requirements, many of the developed platforms have similar features. Therefore, this chapter will focus on three key examples, each of which implement different approaches to facilitate their autonomous navigation and HRI capabilities. These examples are explored in section 2.1. Further analysis is performed with regard to sensing (section 2.2), localisation (section 2.3), motion planning (section 2.4) and human-robot interaction (section 2.5)

2.1 Related Works

This section provides an overview of the three key platforms: Jinny, Robox and the CoBots. Jinny is discussed due to its comprehensive set of HRI elements and its interesting approach to motion planning. Robox is discussed because it has the most hours of operation of all the investigated platforms. This provides a wealth of experimental data and demonstrates that Robox was a successful platform, despite its fairly limited HRI elements. The Cobots are discussed as have been used as a research platform since 2010, with over 30 papers published about them. Through this research a number of ideas relevant to MARVIN have been investigated, including approaches to: sensing, localisation and human detection.

Section 2.1.1 discusses Jinny, section 2.1.2 discusses Robox and section 2.1.3 discusses the Cobots.

2.1.1 Jinny

Jinny is a robot tour guide developed by the Intelligent Robotics Research Centre at the Korea Institute of Science and Technology [16]. The focus of Jinny's development was human-robot interaction and autonomous navigation. A photo of the Jinny system is shown in figure 2.1.

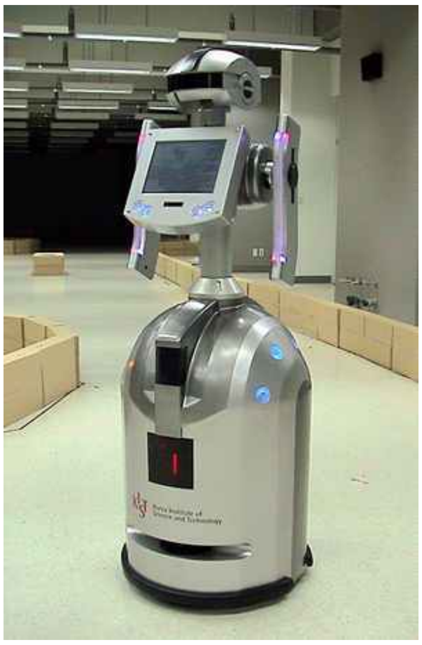


Figure 2.1: Photo of the Jinny system [19].

The Jinny platform uses a two wheeled differential drive mobile base, which has a maximum speed of 1.0 ms^{-1} and a maximum acceleration of 0.5 ms^{-2}. This makes Jinny slower than average human walking speed of 1.41 ms^{-1} [17], which could frustrate humans if they are following. The system has a total height of 1.5 m and a radius of 0.6 m. Jinny's height is similar the average human's height (NZ males 1.77 m, NZ females 1.64 m [10]), which would improve interaction. Its battery packs provide an operating time of 8 hours. Jinny's processing is handled by three separate on-board computers; one for navigation, one for controlling the mobility platform and one for human-robot interaction. This improves redundancy (if one system fails the others can keep running) and performance (as multiple processes can run in parallel across the individual systems) through distributed computation.

Jinny uses an adaptive navigation stack, which changes its motion planner according to the conditions of the environment, further discussed in section

2.4. This improves the versatility and overall performance of the navigation stack as each specialised motion planner is more adept at its specific task than a generalised planner. It uses probabilistic localisation techniques based on Monte Carlo localisation, further discussed in section 2.3. This navigation stack makes use of data from laser rangefinders (the primary obstacle detection and localisation sensor), infrared scanners (to detect obstacles of varying heights that might be missed by the laser rangefinders plannar detection zone), a gyroscope (augments the odometery from the wheels to reduce the effects of odometry drift) and bumper sensors (used to detect collisions for the reactive controller). These sensors are further discussed in section 2.2.

Jinny's human-robot interaction features include: speech synthesis and recognition, face tracking (used to face the user, increasing their engagement), touch screen, 12 LED buttons, LED matrix for expression of emotion and gestures through the use of two 1-DOF (degree of freedom) arms and a 2-DOF neck. The expression of emotion and gestures can be used to humanise Jinny, improving the users engagement. These features are further discussed in section 2.5.

2.1.2 Robox

Robox is a autonomous mobile robot platform developed for human-robot interaction by the Autonomous Systems Lab at the Swiss Federal Institute of Technology Lausanne [15]. Eleven Robox systems operated for up to 12 hours a day for 159 days at the Swiss national Exhibition Expo in 2002. During this time they jointly travelled more than 3315 km and interacted with over $686,000$ visitors. Figure 2.2 shows one of the Robox systems.

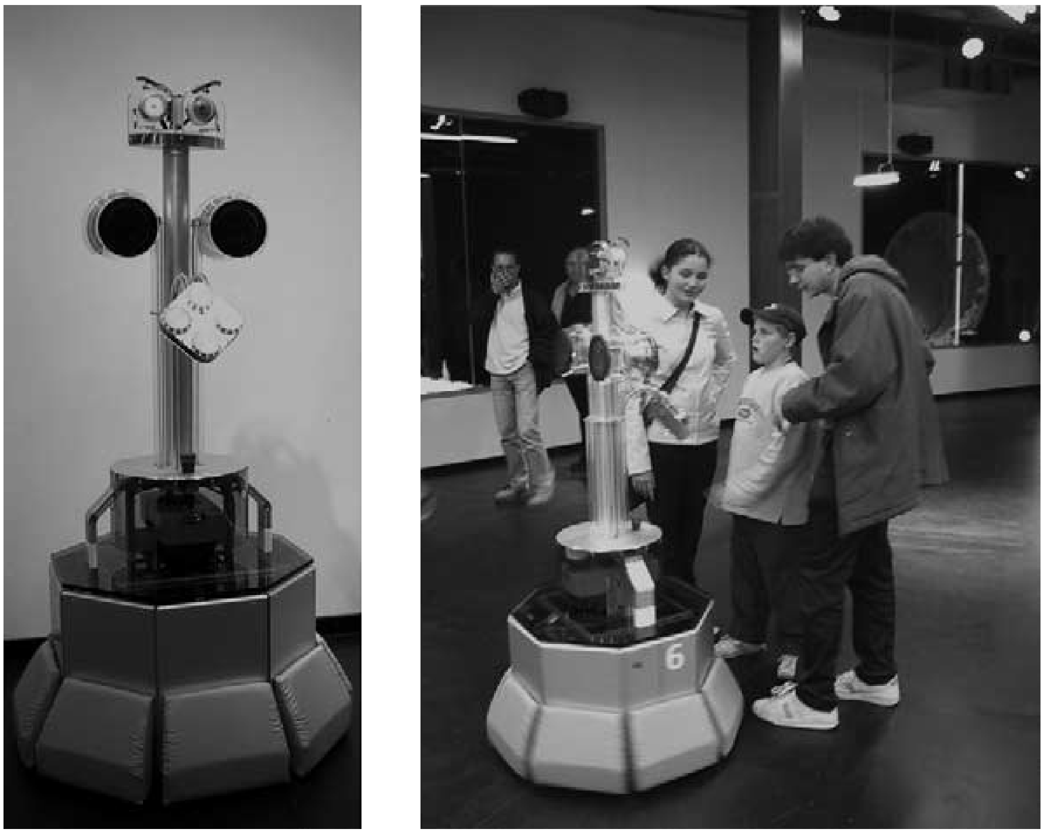


Figure 2.2: Photo of a Robox system.

The Robox system uses a two wheeled differential drive mobile base, similar to Jinny. The system has a total height of 1.65 m, which (like Jinny) is comparable to the average humans height, thus improving interaction. Robox's batteries provide an operating time of up to 12 hours (with a charge time of 12 hours). Similarly to Jinny, Robox's processing is distributed across multiple computers. There are two on-board computers; one handles human-robot interaction and the other handles the mobility platform, sensors and navigation. Again, this improves robustness and performance. Because multiple Robox systems might be running simultaneously, it also communicates with an off-board supervision computer via radio Ethernet. This allows the Robox systems to be managed to cooperate together. It also enables a manual override in the event of an emergency or if the robot gets stuck.

Robox's navigation stack uses a dynamic window approach (DWA) for motion planning and a probabilistic feature-based localisation technique. These techniques are further discussed in sections 2.4 and 2.3. The navigation stack uses the data from two laser scanners, a RGB camera and bump sensors. These sensing systems are discussed in section 2.2.

Robox's human-robot interaction features include: speech synthesis and limited speech recognition, face tracking, expression of emotion through an LED matrix and 1-DOF eyebrows, two eyes that can individually pan and tilt, and four large input buttons. The face tracking, LED matrix, 1-DOF eyebrows and actuated eyes help to increase the engagement of users. These features are further discussed in section 2.5.

2.1.3 CoBots

CoBots are a group of robots developed by the CORAL group at the Carnegie Mellon University. As with the Jinny and Robox systems, CoBots are developed as autonomous mobile robots with a focus on human interaction. CoBots are programed to know their limitations (both in terms of intelligence and physical abilities) and proactively ask humans for help [20]. The CORAL group have published over 30 papers on their CoBot systems, ranging from their first paper "WiFi Localisation and Navigation for Autonomous Indoor Mobile Robots" in 2010 [21] to their latest paper "Localisation and Navigation of the CoBots Over Long-term Deployments" in 2013 [22]. In 2014 the CoBot robots jointly reached 1000 km of autonomous localisation and navigation [1]. Figure 2.3 shows four of the CoBot systems.

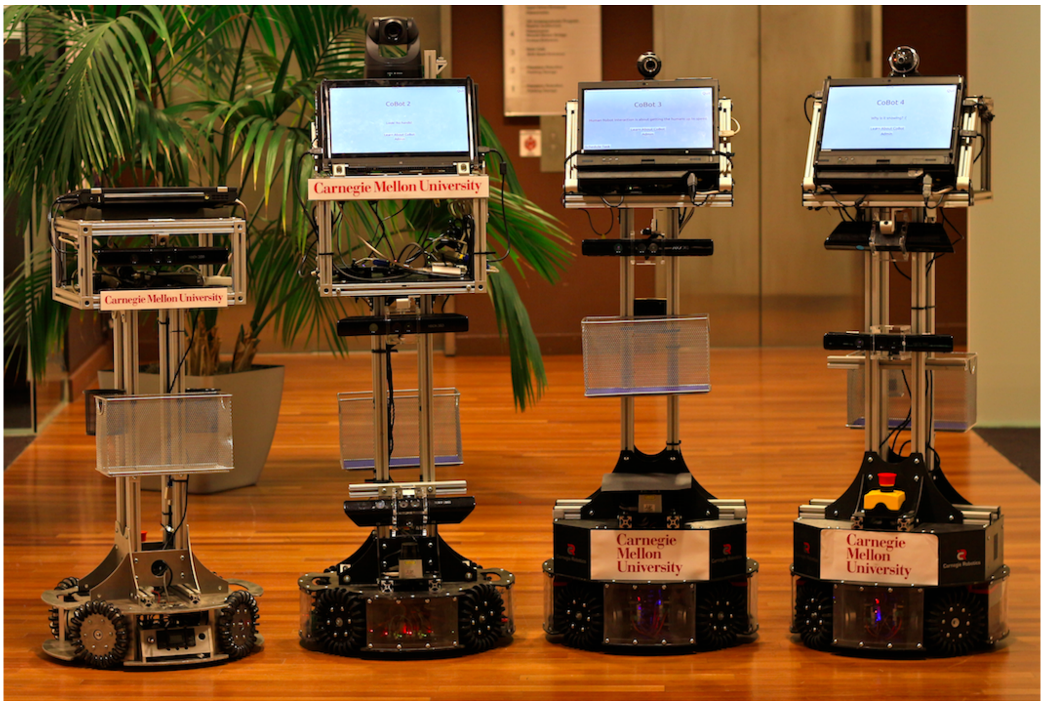


Figure 2.3: Photo of the CoBot systems [1].

The CoBot's mobile base is a scaled up version of the of the CMDragons small-size soccer robots [23], which use an omni-directional (holonomic) drive system with four driven wheels. This allows the robots to drive in all directions without turning, unlike differential drive systems (used by Jinny and Robox) which must turn to change direction. This could be particularly useful for human-interaction, as the robot can move parallel to a human while having the screen facing them at all times. It is also more manoeuvrable, particularly in tight spaces.

CORAL have released numerous papers on the different autonomous localisation and navigation approaches implemented on the CoBots. These approaches include the use of laser rangefinders, depth cameras and WiFi. As with Jinny, the CoBots also make use of Monte Carlo localisation. Each of these approaches are discussed in section 2.3.

The CoBot's human-robot interaction is achieved through a top mounted laptop or touch screen (depending on the CoBot) and voice recognition and synthesis. RGB cameras and depth cameras are also used for human detection and tracking. Unlike Jinny and Robox, the CoBots make no attempt to portray emotion.

2.2 Sensing

Autonomous robots must be capable of sensing their environment in order to make informed decisions through localisation, motion planning and human-robot interaction. Through the literature review it was found that three kinds of sensors are typically used: range sensors, tactile sensors and image sensors (cameras). Range sensors are used by the majority of autonomous mobile robots. They are particularly useful for obstacle detection, motion planning and localisation. There are three forms of range sensors that are most commonly used, laser rangefinders (discussed in section 2.2.1), ultrasonic sensors (discussed in section 2.2.2) and depth cameras (discussed in section 2.2.4). Tactile sensors are another common form of sensor used by mobile robots. They inform the system if a physical object is in contact with the sensor. Because of this, they are typically used to trigger an emergency stop. Tactile sensors are further discussed in section 2.2.5. RGB cameras are found on many autonomous mobile robots. They can be used in various kinds of localisation algorithms, such as RatSLAM [24], but are more typically used for object classification and detection, such as human or face recognition. RGB cameras are further discussed in section 2.2.3.

2.2.1 Laser Rangefinder

As explained in section 3.1.2, MARVIN has inherited a SICK LMS100 laser rangefinder. Laser rangefinders, often referred to as LiDARs, use lasers to measure distance. The laser is aimed by reflecting it against a mirror. The mirror is rotated rapidly providing the laser rangefinder with a 2D field of view (FOV). The measurements are then converted into real-world values by associating the angle of the mirror with its corresponding range measurement. This provides polar measurements that can be converted into Cartesian coordinates that a robot's control system can use, as explained in Chapter 4. Laser range finders often use low intensity infrared (IR) lasers to prevent damaging onlookers' eyes. They typically have a wide FOV, fast response time, long range, high accuracy, good precision and have a small processing overhead. However they are expensive, large, heavy, high power consumption and only have a 2D planar detection area.

Laser rangefinders' FOV, range and accuracy make them particularly useful for localisation and are used by all three of the key examples introduced in section 2.1. However, the 2D FOV makes laser rangefinders unreliable if used as the exclusive sensor for obstacle detection, as any objects above or below its planar FOV will not be detected. This issue is exacerbated in unmodified office-like environments (such as MARVIN's operating environment, as explained in section 1.1.1) which are typically filled with complex 3D obstacles like tables, desks and chairs. Some robotic systems have overcome this issue by periodically tilting a laser rangefinder up and down, effectively turning it into a 3D laser scanner [25, 26]. However, the faster the laser rangefinder is tilted, the larger the vertical distance between each horizontal scan plane. This has two major effects. Firstly, the vertical resolution is inversely proportional to the tilt speed. Secondly, the robot's maximum velocity (while ensuring obstacles are correctly detected) is directly proportional to the laser rangefinder's tilt speed. This introduces a trade-off between the vertical resolution and the robots maximum speed, which means robots using these sensors typically have to move more slowly and can struggle to reliably detect dynamic obstacles. This trade off can be avoided by using a depth camera, as discussed in section 2.2.4.

2.2.2 Ultrasonic Sensors

Ultrasonic sensors use high frequency (greater than 18 kHz) sound waves to measure distances using the time-of-flight technique. They consist of a transducer and a receiver. The transducer emits a ultrasonic pulse which reflects off the measured surface and the resulting echo is measured by the receiver. The time difference between the emitted pulse and received echo is used to determine the measured distance, assuming the speed of the sound wave is a known constant. As they use sound waves, they are not affected by the optical properties of the measured surface. This enables the detection of transparent materials, such as glass. However, they also have a number of disadvantages including: changing atmospheric conditions (such as temperature, humidity and pressure) can change the speed of the sound wave introducing systematic errors, multiple ultrasonic sensors can interfere with one another due to echos, false measurements can be cause by specular reflections, and they typically have a wide measurement beam introducing uncertainties in the position of the measured object.

2.2.3 RGB Cameras

RGB cameras are sensors that convert visible light into individual red, green and blue signals. They typically consist of a lens which focuses the light onto a receiver. The receiver absorbs the light and produces signals that a computer can understand. The computer uses these signals to produce colour images. When selecting a camera for a mobile robot, there are three major considerations: resolution (the number of pixels in the image), FOV and frame rate (the number of images that can be taken in a second, typically measured in frames per second (FPS)). A higher resolution camera is capable of obtaining more detailed images, which the robot can use to detect features. A large FOV enables the camera to detect features in a larger area. A higher frame rate helps detect movement, and reduces the effects of motion blur (further explained in section 2.3.1). The amount of data produced is dependent on the resolution and frame rate. This introduces a trade off between computation overhead and image detail. This is an important consideration as image processing techniques are typically computationally expensive.

RGB cameras have been used by Jinny, Robox and the CoBots. These platforms have made use of the colour images for object recognition, human detection and face tracking. RGB cameras have also been used for localisation and mapping techniques, further explained in section 2.3. The use of RGB cameras will be investigated for these applications in section 3.2.1.

2.2.4 Depth Cameras

Depth cameras operate in a similar fashion to standard RGB cameras; however, rather than producing a colour image, they produce a depth image. A depth image is a grey scale image where each pixel represents depth. Depth cameras have the same major selection criteria for use with mobile robotics as RGB cameras: resolution, FOV and frame rate. Because they have both a vertical and a horizontal FOV, they are effective for detecting the complex 3D obstacles that 2D laser rangefinders often miss.

Depth cameras typically use an IR emitter to illuminate the scene. They then measure the reflected light using a IR camera to determine the distance. This can be done using intensity, triangulation or time-of-flight. In comparison to laser rangefinders, depth cameras usually have a narrower horizontal FOV, smaller range, lower accuracy, lower precision and much larger processing overhead. However, they have a similar response time to a 2D laser rangefinder and the vertical FOV makes them effective for obstacle detection. In recent years, the price of depth cameras has dropped dramatically, making them a viable choice for mobile autonomous robots. CORAL have published numerous papers on the use of the Microsoft's depth camera, the Kinect, on their CoBots for obstacle detection, localisation and navigation [27, 28] as well as for human detection [29].

Because of their effectiveness for obstacle detection, the use of depth cameras on MARVIN is explored in section 3.2.1.

2.2.5 Tactile Sensors

Tactile sensors are used by many mobile robots, including Jinny and Robox, as a simple method for detecting obstacles that might be missed by range sensors. They are typically used to initiate an immediate stop and are used as part of the reactive control loop, rather than the deliberative control loop. The use of tactile sensors for MARVIN is investigated in section 3.2.1.

2.3 Localisation

Localisation is the process of estimating the robot's pose (position and rotation) in a map. This estimate is required by the robot's control system to make high level plans from the current pose to a goal pose. This high level planner is referred to as a global planner and is further explained in section 5.3.1. For localisation to work effectively, an accurate map is required. If the robot is working in a known environment, a pre-built map can be used, otherwise the process of building a map and localising within that map must be performed at the same time. This is know as simultaneous localisation and mapping (SLAM). As MARVIN is operating in a known environment (as explained in section 1.1.1) SLAM is not required. However, the process of pre-building maps of the operating environment can make use of SLAM techniques, as explained in section 5.4.

There are three general categories of localisation methods typically use by mobile autonomous robots; visual based localisation (discussed in section 2.3.1), wireless signal based localisation (discussed in section 2.3.2) and range based localisation (discussed in section 2.3.3).

2.3.1 Visual Based Localisation

Visual based localisation techniques use image data to localise. A common example of this is RatSLAM [24]. RatSLAM uses artificial neural networks (ANN) to estimate the robot's pose from a stream of images. It is inspired by the way a rat's brain processes visual information to localise. RatSLAM has been demonstrated to work on an indoor humanoid robot in [30]. This paper discussed the issue of multiple false positive loop closes. This is caused by the robot's jerky vertical and lateral movement producing blurry images. Blurry images lead to loss of detail and washed out features. This reduces the effectiveness of visual SLAM techniques as the features used to localises can be missed. As discussed in section 3.1.1, MARVIN's mobility platform is a Segway RMP. Due to the Segway's self balancing, the problem of blurry images would be exacerbated. For this reason, visual localisation techniques will not be used.

Another visual localisation technique is to use 3D reconstruction to produce 3D points from multiple images (either from stereo cameras or from multiple locations). This process looks for key points in two frames and uses the camera properties and position to estimate the key points in 3D coordinates. The 3D points can then be used to localise through the use of range based localisation techniques such as those explained in section 2.3.3, for example [31]. However, visual localisation techniques are typically less robust and less computationally efficient than range based localisation (for example, localisation using laser rangefinders) [32].

2.3.2 Wireless Signal Based Localisation

Wireless signal based localisation uses external wireless signals to localise within an area. The most common form of this is the global positioning system (GPS). GPSs connect to multiple satellites to triangulate their positing. As such they are typically used for outdoor localisation as it is difficult to acquire reliable satellite signals indoors. Another common technique is the use of WiFi signal strength to localise. This has been shown as an effective localisation system inside buildings with multi-node networks (such as in MARVIN's operating environment) in CORAL's paper [21]. CORAL also discuss the usefulness of this technique in combination with range based localisation [33]. They demonstrate that the WiFi strength localisation method can provide more accurate estimates in open areas (where the range sensors provide less information), but range based techniques are more accurate in confined areas.

2.3.3 Range Based Localisation

Range based localisation uses the data from range sensors to estimate the robot's position, typically by comparing the range measurements to a known map. From sections 2.3.1 and 2.3.2, it is determined that range based localisation is a more general and robust localisation solution than visual or wireless signal based methods. Both Jinny and CoBots have shown that the Monte Carlo localisation (MCL) technique is effective in indoor environments, similar to MARVIN's operating environment. For this reason, the use of (MCL) is explored in section 5.1.

2.4 Motion Planning

The majority of mobile autonomous robots, including all three of the key examples, make use of a hybrid navigation system. Hybrid navigation systems combine both deliberative and reactive control. Deliberative control is the high level plan, generated by the global planner discussed in section 5.3.1. The reactive controller uses the sensor data to attempt to avoid obstacles. The dynamic widow approach (DWA) is a popular reactive collision avoidance technique [34]. It has been demonstrated an effective method by Robox, CoBots and [25].

Jinny extends the idea of the hybrid approach by adaptively switching the motion planner according to the condition of the robot's environment. During normal operation Jinny makes use of an extended version of Konolige's gradient method [35]. The extensions deal with path blocking and goal occupation. During this operation the optimal path is continually updated; however, if a large number of people are detected, path updating is disabled. This introduces a "stop and wait" behaviour, which is deemed safer than constantly trying to navigate around people. If Jinny is attempting to navigate through a narrow area, then a wall following method is used instead. This method was found to be less affected by localisation errors, and performed better in confined areas. If the other three methods fail, a tele-operator can take over the control of Jinny.

The use of the DWA as MARVIN's reactive controller (referred to as the local planner) is explored in section 5.3.2. The possible extensions of adaptive hybrid control are discussed in section 9.2.

2.5 Human Robot Interaction

Human-robot interaction requires the implementation of both receptive and expressive elements. Receptive elements receive requests from the user, while the expressive elements provide the response to the user. Different implementations of these elements are discussed sections 2.5.1 and 2.5.2.

2.5.1 Receptive Elements

All three of the key examples introduced in section 2.1 make use of limited speech recognition. These robots all operate in environments where unpredictable noise can be generated at any time (for example, groups of people talking). This makes recognising speech reliably difficult. Because of this, they typically use key-word recognition as opposed to natural language processing.

Both Jinny and the CoBots make use of a touch screen to get inputs from the user. This is an effective method as the developers can dynamically change the possible inputs to the system, depending on the current situation (unlike buttons which are static). Touch screens also provide a much wider variety of inputs (for example gestures like "pinch to zoom" or "slide to scroll"). However, buttons have the advantage of ease of use and tactile feed back.

Systems focused on human interaction must be able to detect when a human is present. This been achieved through motion tracking with laser range scanners (Robox), face tracking with RGB cameras (Jinny and Robx) as well as human tracking through depth cameras (CoBots). Motion tracing detects all moving objects (not just humans) and as such is prone to false positives. Face tracking requires the user to be facing the camera and is prone to false negatives if the person is not well positioned. Human detection with depth images is typically a more robust method and is explored in section 4.4.3.

2.5.2 Expressive Elements

Speech synthesis has been used successfully by all three key examples. It has the advantage that people can hear the robot's response from any position, whereas LEDs or screens require line of sight. However, they have the disadvantage that the synthesised voice may be drowned out by background noise or difficult to understand for some people (especially if they primarily speak a language foreign to the robot). The use of voice synthesis is investigated in Chapter 7

Both Jinny and the CoBots use touch screens as both a receptive and expressive element. As explained in section 2.5.1, they have improved versatility in comparison with static elements such as LEDs.

Both Jinny and Robox use LED matrices and actuation to emulate emotion. The LED matrices are used to produce facial expressions. Robox extends the facial expressions through the use of actuated eyebrows, while Jinny makes use of a 2-DOF neck and two 1-DOF arms. This ability to emulate emotion makes the robots more personable which can produce more natural human-robot interactions. As explained in section 3.1.4, MARVIN has inherited an actuated torso for this reason. The use of this torso is explored in section 6.2 and Chapter 7.

Chapter 3

MARVIN Platform

This chapter introduces MARVIN's hardware platform. For MARVIN to successfully operate as a research platform for human-robot interaction, it must be capable of locomotion, sensing, control and human-robot interaction hardware. MARVIN's platform comprises of a combination of inherited and newly augmented hardware. Section 3.1 introduces MARVIN's inherited hardware and section 3.2 discusses the augmented hardware.

3.1 Inherited Platform

MARVIN was first developed as a "large autonomous robotic vehicle that would be suitable for use as a security device". This original development is discussed by Daniel Loghnane's 2001 thesis [36]. It was in this thesis that the name "MARVIN" was coined, which stands for Mobile Autonomous Robotic Vehicle for Indoor Navigation. Since then, MARVIN has undergone numerous developments. This includes the addition of an actuated humanoid torso in collaboration with Robotechnology, as described by Ashil

Prakash's thesis [37]. This robotic torso is discussed in section 3.1.4. In 2014 MARVIN's original differential drive mobility platform was replaced with a Segway robotic mobility platform, as explained in [3, 38]. This mobility platform is further discussed in section 3.1.1. The 2014 redevelopment also replaced MARVIN's original sensing systems and control hardware, as explained in sections 3.1.2 and 3.1.3. The humanoid torso is the only remaining feature of the developments conducted prior to 2014 and is further discussed in section 3.1.4.

3.1.1 Inherited Locomotion

MARVIN makes use of a Segway robotic mobility platform (RMP) for locomotion. The Segway RMP is based on the self-balancing Segway Human Transporter (HT), which uses the inverted pendulum model to maintain balance [39]. The Segway RMP is specifically designed for robotic research and is controlled using serial communication via a universal serial bus (USB) 2.0 connection. The inherited platform is the Segway RMP200 shown in figure 3.1. The specifications of the Segway RMP200 are outlined in table 3.1.

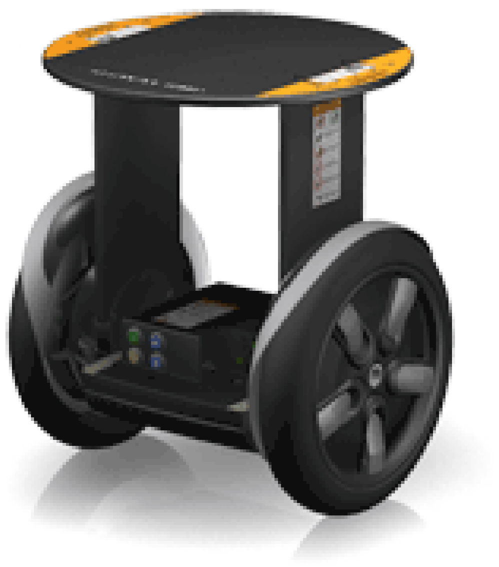


Figure 3.1: Segway RMP200 [2].

Table 3.1: Key characteristics of the Segway RMP200 [2].

Parameter	**Value**
Top plate diameter	61 cm
Overall height	74 cm
Wheel diameter	47 cm
Distance between wheels	47 cm
Weight	64 kg
Top speed	16 km/h
Payload	91 kg
Turning radius	0 m
Range under good conditions	19 km
Recharge time (from empty)	6 hours

The top speed of 16 km/h ($4.44\ ms^{-1}$) exceeds a typical human's walking speed ($1.41\ ms^{-1}$ as discussed in section 2.1). This allows MARVIN to travel at a comfortable walking speed when guiding people. The $0\ m$ turning radius (like the differential drive and omni-direction drive systems discussed in section 2.1) allow MARVIN to navigate the enclosed spaces found in indoor environments. The $61\ cm$ diameter is smaller than the width of a typical wheel chair (600 mm to 700 mm)[40]. This enables MARVIN to move through the majority of areas in the operating environment, which is designed to have disabled access (as discussed in section 1.1.1). The maximum payload of $91\ kg$ can carry MARVIN's torso, sensors and control electronics with spare load for future developments.

The Segway RMP has better versatility when compared to typical mobility platforms, such as the differential drive and caster systems used by *Jinny* and *Robox* or the omni-directional drive system used by the *CoBots* (discussed in sections 2.1.1, 2.1.2 and 2.1.3). This is due to its self-balancing abilities, which allow it to climb over small obstacles that would typically stop caster based systems and climb slopes that would make non-balancing systems unstable. It can also account for external disturbances (such as being pushed), which would destabilise non-balancing systems, potentially overturning them causing physical damage.

However, this self-balancing feature also introduces a number of challenges when operating MARVIN. One challenge is the Segway's constantly changing pitch, which increases the complexity of sensor processing (as discussed in Chapter 4). Another challenge is the increased risk of operation. The Segway can only maintain balance when its control system is operating and its motors are powered. If it fails to maintain balance, it can become a safety hazard to onlookers and could cause damage to MARVIN and its environment. This was addressed by the addition of tilt limiters in [3, 38] (displayed in figure 3.2), and is further discussed through the control systems described in Chapter 6.

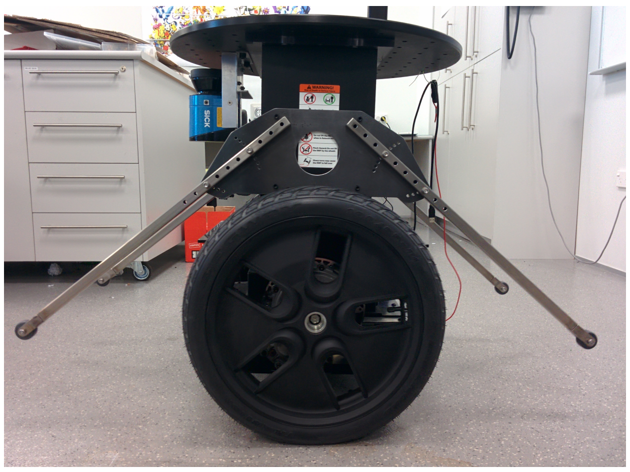


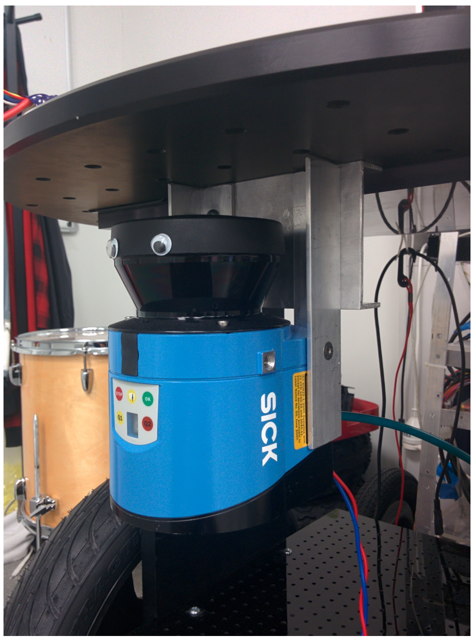
Figure 3.2: Segway RMP augmented with tilt limiters [3].

3.1.2 Inherited Sensors

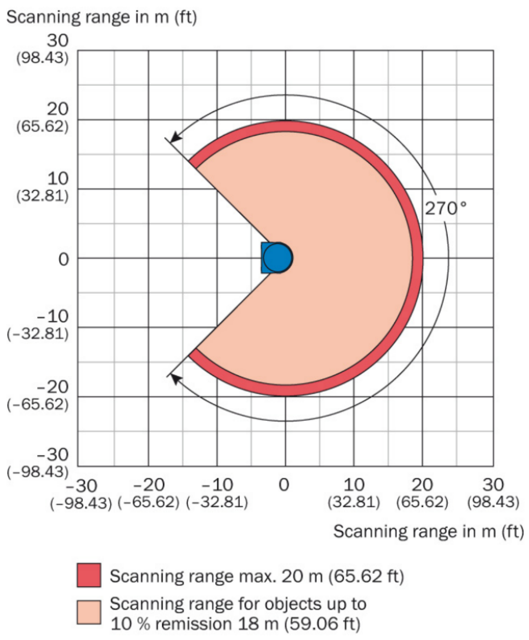
MARVIN has inherited three sensing systems: a laser rangefinder, an ultrasonic network and a set of stairway detectors. Each of these systems were added to MARVIN during the 2014 development, as explained in [3, 38].

Laser Rangefinder

The inherited laser rangefinder is a SICK LMS100, which is shown mounted at the front of MARVIN in figure 3.3a. This laser rangefinder provides MARVIN with a $270°$ planar FOV, as visualised in figure 3.3b. The sensor has an operating range of 0.5 to 20 m, an angular resolution of $0.5°$ a maximum systematic error of ±30 mm and a maximum statistical error of ±12 mm [41].



(a) Mounted Laser Rangefinder [38]



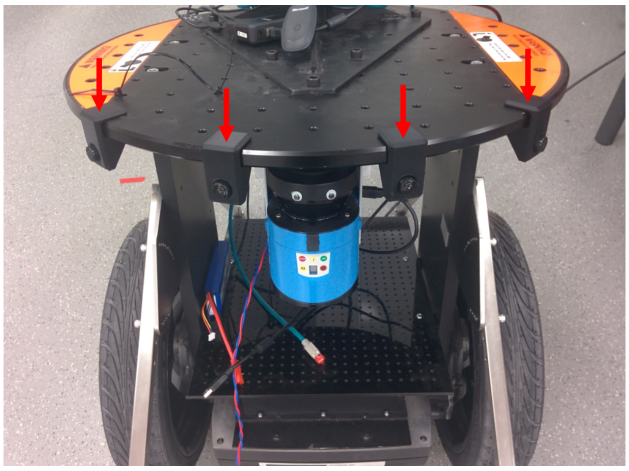
(b) Detection Area [41]

Figure 3.3: MARVIN's SICK LMS100 Laser Rangefinder.

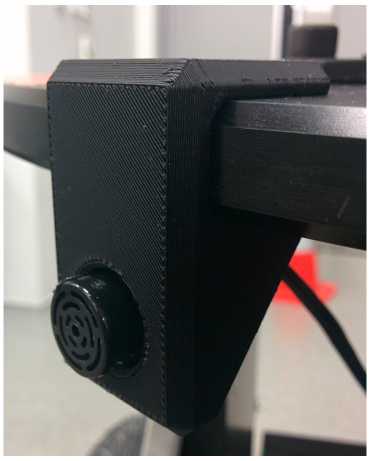
As discussed in section 2.2.1, laser rangefinders are effective sensors for localisation. The LMS100's long range, wide FOV, high accuracy and low errors allows it to reliably detect features in a large area with one sweep of its laser. This makes it a particularly effective sensor when combined with probabilistic localisation techniques, as it can compare a large number of accurate data points to a known map, resulting in probabilistically likely location estimates. These characteristics also make the SICK LMS100 a capable sensor for obstacle detection; however, it must be augmented with additional sensors to detect obstacles above and below its planar FOV. The data from this laser rangefinder is processed to account for the Segway's dynamic pitch (further explained in section 4.3) and then used by MARVIN's navigation stack for localisation (explained in section 5.1) and obstacle detection (explained in section 5.2).

Ultrasonic Sensor Network

As the laser rangefinder uses IR lasers to measure distances, it is unreliable at detecting glass. MARVIN's operating environment has a number of glass walls, as discussed in section 1.1.1. This requires that the laser rangefinder is augmented with additional senors for detecting glass. MARVIN has inherited an ultrasonic sensor network (USN) as an attempt to address this issue. Ultrasonic sensors measure distances by pulsing ultrasonic sound-waves and measuring their time-of-flight (the time it takes for the pulse to reach the measured surface, reflect off it and return to the ultrasonic receiver). Unlike the laser rangefinder, their measured distances are unaffected by the optical properties of the measured surfaces. This allows them to detect the glass panels. The USN comprises of a series of four ultrasonic sensors mounted around the front of the Segway's mounting plate, shown by figure 3.4. The USN is interfaced with ROS via a custom interface board based off the Arduino Mega 2560 [38].



(a) Full Network.



(b) Individual Sensor.

Figure 3.4: Ultrasonic Network.

While the USN can detect glass, when compared to the laser rangefinder it has a short range ($6\ m$), a narrow FOV ($160°$), lower range resolution ($25.4\ mm$) and much lower angular resolution ($40°$). For this reason, the USN is only used for obstacle detection as explained in section 5.2. As per the suggestions from [38], the original four wide beam ultrasonic sensors were replaced with five narrow beam sensors. This improves the angular resolution to $30°$ and improves the detection reliability.

Stairway Detectors

To detect obstacles above and below the laser rangefinder's planar FOV, MARVIN has inherited two stairway detectors mounted above MARVIN's left and right wheels, facing forwards. Each stairway detector consists of two SHARP position sensitive devices (PSDs). The PSDs contain an IR emitter and detector and use triangulation techniques to measure distances. The stairway detectors use the two PSDs to measure the angle of the ground in front of MARVIN (relative to the Segway pitch) and compare it to the predicted pitch (calculated from the Segway pitch assuming the ground plane is horizontal). If there is a discrepancy between the measured and predicted ground angle it can be assumed that there is an obstacle. This obstacle could be a low-lying object, an excessive slope or an upward or downward stairway. This process is visualised by figure 3.5.

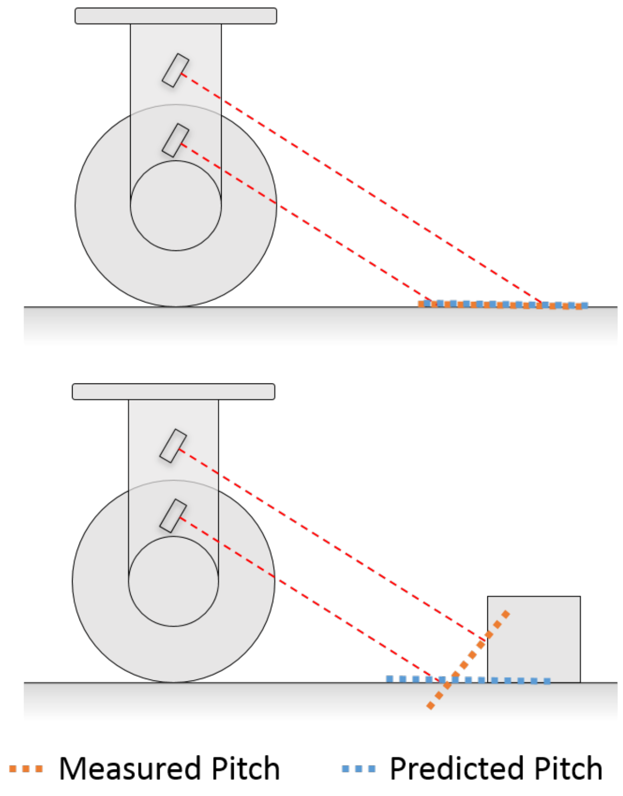


Figure 3.5: Stairway detectors [3].

The stairway detectors were shown to be effective in [3, 38]; however, due to the inclusion of the Kinect RGB-D camera (discussed in section 3.2.1) they have become redundant and are no longer used. However, their obstacle detection method is extended in the ground and celing removal algorithms used to process the Kinect's depth images, as discussed in section 4.4.

3.1.3 Inherited Control

MARVIN's 2014 redevelopment replaced its original robotic development environment (RDE), Microsoft's Robotics Developer Studio (MRDS), with Robot Operating System (ROS) [42]. ROS is an open-source RDE that runs on Linux. It uses a graph-based messaging network to communicate between subsystems (called nodes) [38]. Each node can subscribe and publish to message topics. This standardises and abstracts the inter-node communication making it transparent. This allows nodes to be written in different styles and languages without having to redevelop communication methods between them, which reduces development time. ROS handles the processing of each node, allowing multiple nodes to run simultaneously. This provides distributed processing for each of the robot's subsystems on the same machine, which leads to more efficient and robust control. ROS also has a highly active development community, which provides support and a vast library of packages for performing standard robotic tasks. This also dramatically reduces development times. For these reasons, ROS will continue to be used throughout this project.

MARVIN inherited a laptop as its control computer, the specifications of which are outlined in table 3.2. This laptop has neither the connection hardware nor processing power to interface with the depth camera (introduced in section 3.2.1). This inherited laptop has a number of redundant features which increase its size and weight, such as its screen, keyboard and DVD drive. The laptop also uses a mechanical hard disk drive (HDD) which is prone to vibration damage. As the Segway can cause large vibrations during motion, this is a potential point of failure that can be avoided through the use of solid state storage. These issues are resolved by replacing the inherited laptop with a new control computer, further explained in section 3.2.2.

Table 3.2: Key characteristics of the original control computer.

Parameter	**Value**
CPU Cores	2
CPU Threads	2
CPU Frequency	2.2 GHz
RAM	2.0 GB
Storage	80 GB (HDD)
Size ($w \times h \times d$)	$355\ mm \times 40\ mm \times 265\ mm$
USB Ports	$4\times$ USB2.0

MARVIN has no inherited control architecture and only a rudimentary prototype navigation stack designed to demonstrate its potential, as opposed to being a robust autonomous mobile platform. These systems are newly developed in Chapters 6 and 5 respectively.

3.1.4 Inherited Human-Robot Interaction

One of MARVIN's unique inherited features is its 7-DOF actuated torso. This torso was developed to enable MARVIN to emulate basic emotions. The torso can be controlled to: tilt the torso, extend the left and right shoulders, extend the neck, nod, tilt and shake the head and individually control the RGB LED eyes. This robotic torso can be used during human-robot interaction to personify MARVIN, similarly to the Jinny and Robox robots discussed in section 2.1. Figure 3.6 shows a photo of MARVIN's torso.

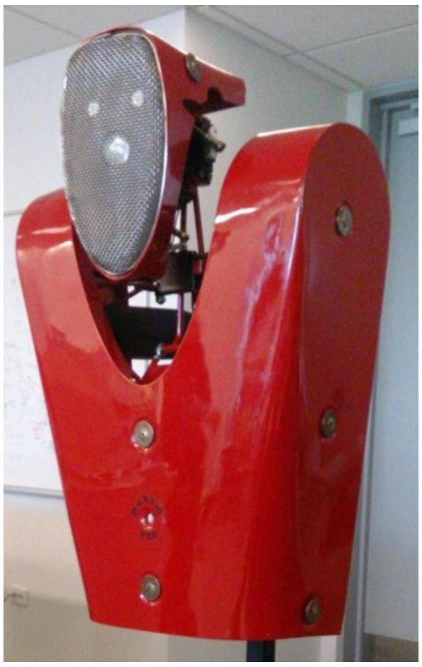


Figure 3.6: MARVIN's inherited actuated humanoid torso.

The torso was interfaced with ROS during the 2014 development, and is controlled via a the *torso_node* written in python. The control of this torso is extended in section 6.2.

MARVIN has inherited no other expressive elements, or any receptive elements for human-robot interaction. The development of these elements are discussed in section 3.2.3.

3.2 Augmented Platform

MARVIN's inherited systems must be augmented with new systems to better meet the objectives introduced in Chapter 1. MARVIN requires a sensing system to more reliably sense the 3D objects commonly found in office-like environments (discussed in section 3.2.1), more powerful control electronics (discussed in section 3.2.2) and additional receptive and expressive elements for human-robot interaction (discussed in section 3.2.3).

3.2.1 Augmented Sensors

As explained in section 3.1.2, MARVIN has inherited a laser rangefinder, ultrasonic network and stairway detectors. These sensors can be used for effective localisation (as discussed in section 5.1); however, they can only provide limited obstacle detection. They have dead-zones between their detection areas, where their FOVs don't overlap. Many objects inside the operating environment (explained in section 1.1.1) fall into these dead-zones, such as desk/bench tops, rubbish bins and chair legs. Additionally, these sensors provide no reliable methods for human detection and tracking. The laser rangefinder can be used for motion tracking to detect humans; however, this is typically prone to false positives (as explained in section 2.5.1). These limitations are addressed through the addition of a RGB-D camera and tactile sensors.

RGB-D Camera

Depth cameras provide both a vertical and horizontal FOV, as explained in section 2.2.4. RGB-D cameras combine a RGB camera and a depth camera into the same sensor. The RGB and depth cameras can be used separately, or together to produce a RGB point-cloud. A point cloud is a set of 3D points created from the depth image. Typically each point only contains three values (x, y and z), but a RGB point-cloud's contain six values (x, y,

z, red, green and blue). Point-clouds are not currently used by MARVIN due to their increased processing overhead (as discussed in section 4.4), but they may be used in future developments. Since Microsoft released a RGB-D camera for their Xbox 360 game console in 2010 (the Kinect), RGB-D cameras have dropped dramatically in cost. This has made them a popular sensor for mobile robots in recent years.

Currently there are three common manufacturers of consumer grade RGB-D cameras: Microsoft, ASUS and Intel. When Microsoft released their latest game console in 2013, the Xbox One, they also released the Kinect v2. The Kinect v2 moved from the intensity based depth mapping to a time-of-flight method, which improved its range and accuracy. ASUS manufactures the Xtion (re-branded from PrimeSense Carmine) RGB-D camera, which has similar specifications to the original Kinect. At the start of 2015 Intel announced the RealSense. The RealSense has similar specifications to the Kinect v2, however is physically much smaller and more power efficient. All three of these systems are compared in table 3.3.

Table 3.3: Comparison of RGB-D camera specifications.

Manufacturer	**RGB-D Camera**	**RGB Resolution**	**Depth Resolution**	**RGB FOV (H × V)**	**Depth FOV (H × V)**	**Max Depth Range**	**Power Consumption**
Microsoft	Kinect v2 [43]	1920×1080 ($60\ Hz$)	512×424 ($30\ Hz$)	$84° \times 54°$	$70.6° \times 60°$	$8\ m$	~ 15 W
ASUS	Xtion [44]	1280×1024 ($60\ Hz$)	640×480 ($30\ Hz$) or 320×240 ($60\ Hz$)	$58° \times 45°$	$58° \times 45°$	$3.4\ m$	2.5 W
Intel	RealSense [45]	1920×1080 ($60\ Hz$)	640×480 ($60\ Hz$)	$70° \times 43°$	$59° \times 46°$	$10\ m$	1.6 W [46]

Of the specifications outlined in table 3.3, FOV and maximum depth range are the most important for obstacle detection (assuming the other specifications are comparable). The area in which the sensor can detect obstacles is proportion to the FOV and range. The Kinect v2 has the largest FOV, while the RealSense has the longest range. The Xtion has the narrowest FOV and shortest range, therefore it is not considered for use with MARVIN. Multiple depth sensors could be used to increase their collective FOV. The RealSense would be preferable for this due to its lower power consumption, larger range and higher resolution (which would make it easier to stitch the multiple depth images together). Due to supply issues, RealSense sensors could not be obtained during this project. Instead, a single Kinect v2 was used; however, multiple RealSenses may be considered in future (further discussed in section 9.2).

The Kinect v2 has been demonstrated as an effective sensor for indoor mobile robots by [47, 48]; however, it is not as effective outdoors (it has higher noise when overcast and fails to produce valid data in direct sunlight). As MARVIN's operating environment consists of indoor office-like areas, this sensor is effective for obstacle detection. The steps taken to process the Kinect's data are discussed in section 4.4. MARVIN uses this processed Kinect data for obstacle detection (explained in section 5.2) and human tracking (explained in section 4.4.3).

For better obstacle detection performance, the Kinect should be mounted at an angle which can sense the ground in front of MARVIN ($\sim 5^\circ$ below horizontal). For better human detection performance the Kinect should be mounted at an angle which can view entire people in a single frame ($\sim 10^\circ$ above horizontal). To enable better performance for both obstacle detection and human detection, the Kinect is mounted on a pivot actuated by a servo. This mount is shown in figure 3.7.

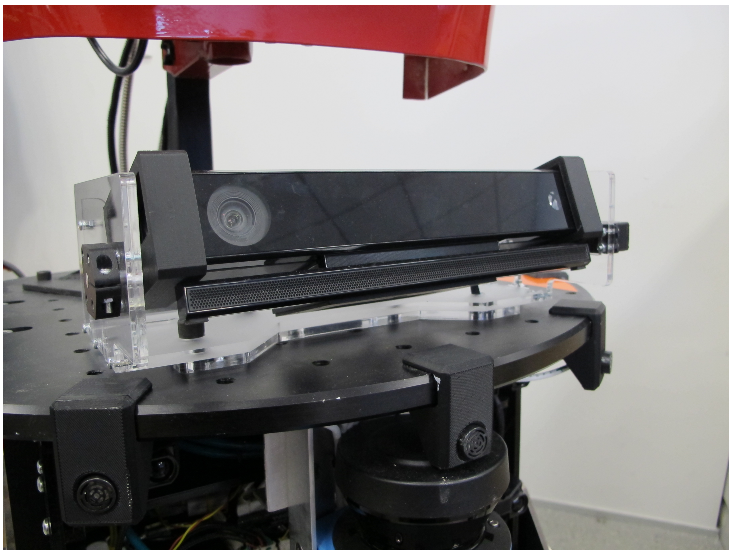


Figure 3.7: Microsoft Kinect v2 mounted on MARVIN.

This mount allows the Kinect to tilt from $-20°$ to $20°$ (where $0°$ is parallel with the Segway's mounting plate). The servo is controlled through a microcontroller connected to the main control computer, as explained in section 4.2.2.

The addition of the Kinect aligns with the 3D sensor objective 1b.

Tactile Sensors

Tactile sensors are useful for reactive control as discussed in section 2.2.5. They can be used to command the motion platform to perform an immediate stop in the event of a collision. When the Segway collides with an object, its internal control attempts to account for the additional force the object applies to the Segway. If the Segway gets stuck on the object, it continues to account for this force, which can lead to unstable control and unsafe operation. To avoid this the Segway needs to know that it has collided (or is about to collide) with an object. This is typically achieved with MARVIN's range sensors, but if an object is in the range sensors' dead-zones it will be missed. Tactile sensors can be used to help avoid this issue. Ideally any tactile sensors used on MARVIN would detect an obstacle before MARVIN collides with it (to prevent the collision force problem). Whisker sensors are tactile sensors that have a protruding trigger element (the whisker). They use the whisker to detect obstacles before the robot collides with them. By comparison, the bump senors introduced in section 2.2.5 only trigger when the robot collides with an object.

MARVIN makes use of four whisker sensors, each mounted to one of the Segway's tilt limiters. This is because the tilt limiters are the outermost parts of MARVIN, making up the four corners of its footprint. Figure 3.8 shows the two front whisker sensors.

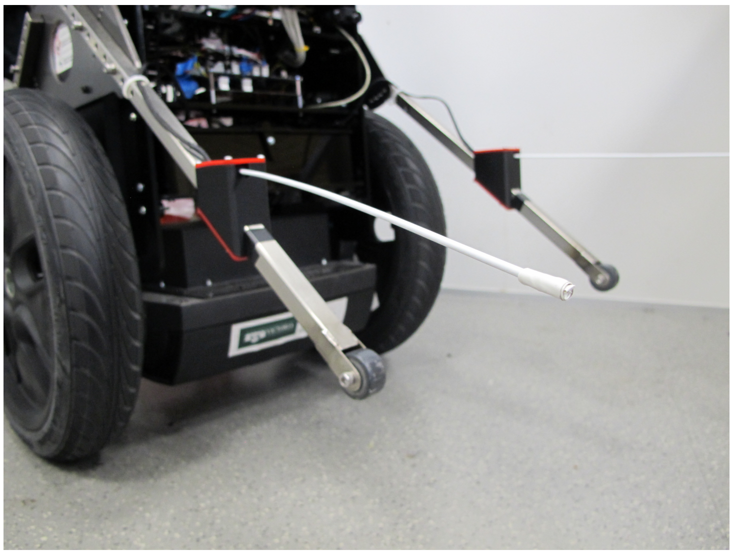


Figure 3.8: Whisker sensors mounted on MARVIN's front tilt limiters.

The whisker sensors consist of two tactile switches on either side of a whisker. If the whisker rotates or bends due to a collision it triggers one or both of the switches. The whiskers consist of a 1.5 mm stainless steel core (for rigidity) surrounded by a 3.0 mm styrene shell (for flexibility and to prevent the metal core from damaging the environment). They are 400 mm long to give the Segway time to react after an obstacle has been detected and are mounted at an outward angle of $20°$ to help detect objects to MARVIN's sides. This is shown by figure 3.9.

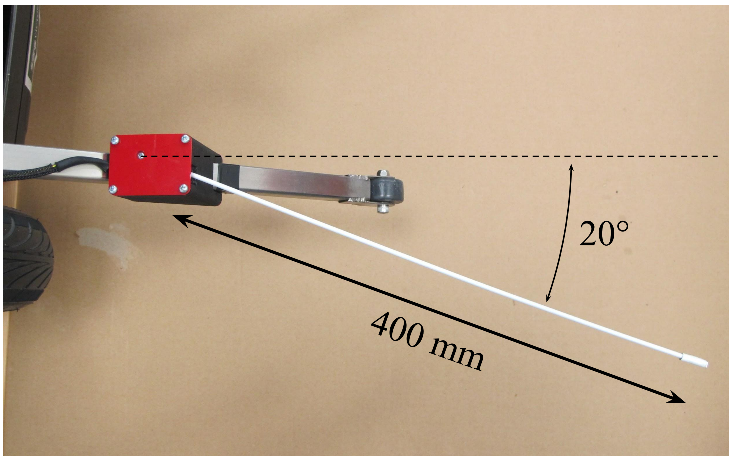


Figure 3.9: Whisker sensor dimensions.

The whisker sensors are interfaced directly into MARVIN's movement control (as described in section 6.1) and the navigation stack's obstacle detection (as described in section 5.2).

3.2.2 Augmented Control Hardware

The inherited control computer does not have the connection hardware to interface with the Kinect, nor the computational power to process its data (as mentioned in section 3.1.3). Microsoft recommends the following specifications for the Kinect's Windows Software Development Kit (SDK): dual core 64-bit 3.1 GHz processor, 4 GB RAM, USB 3.0 and a DX11 capable graphics adaptor [49]. As explained in section 3.1.3, MARVIN is developed in ROS (which runs on Linux), so the windows SDK cannot be used. However, these specifications give an indication for the requirements of MARVIN's new control computer. In addition to meeting the processing requirements for the Kinect, it is desired that the new control computer is physically small (to preserve space on MARVIN for future developments) and power efficient (to maximise MARVIN's operating time). The Intel Next Unit of Computing (NUC) is a compact computer which runs mobile grade hardware (which is more power efficient than desktop grade hardware). The Intel NUC D54250WYKH was available and is used as MARVIN's new control computer. Its specifications are shown in table 3.4.

Table 3.4: Key characteristics of the NUC [6].

Parameter	**Value**
CPU	Intel Core i5 4250U (64-bit)
CPU Cores	2
CPU Threads	4
CPU Frequency	1.3 GHz (2.6 GHz Boost)
RAM	8.0 GB
Graphics Processor	Intel HD Graphics 5000 (DX11)
Storage	120 GB (SSD)
Size ($w \times h \times d$)	$115\ mm \times 50\ mm \times 110\ mm$
USB Ports	$4\times$ USB3.0

From this table it can be seen that the Intel NUC D54250WYKH meets or exceeds all the recommended specifications for the Kinect's SDK, except the processor speed (2.6 GHz as opposed to 3.1 GHz). However, as shown in section 4.4, the Intel Core i5 4250U is capable of running the Kinect at 30 frames per second (FPS) at $\sim 45\%$ CPU load. This provides sufficient overhead for the rest of the MARVIN's processing requirements. However, if multiple RealSenses are used in future (as suggested in section 3.2.1) the NUC's processor may need to be upgraded (or augmented). This is further discussed in section 9.2.

To enable MARVIN's development via external computers and allow the control computer to communicate with the human-robot interface tablet (introduced in section 3.2.3) a wireless local network is required. This is implemented through the use of a TP-Link TL-WR702N Nano Router, which is a compact (57 mm $\times$ 18 mm $\times$ 57 mm) low power (powered by USB 2.0) 150 Mbps Wireless N router [50].

3.2.3 Augmented Human-Robot Interaction

MARVIN requires the addition of both expressive and receptive human-robot interaction elements. From Chapter 2 it was found that speech recognising and synthesis are required for human-interaction and that touch screens provide a versatile receptive and expressive element. These expressive and receptive elements can be introduced to MARVIN through the addition of a tablet computer.

As explained in Chapter 7, MARVIN uses Microsoft's Speech Platform SDK for voice synthesis and recognition. Microsoft's recommended system requirements include: dual core 2 GHz processor, 1 GB RAM, DX9 compatible graphics processor and 40 GB + storage [51]. As explained in section 9.2, the Microsoft Speech Platform might be replaced with Nuance's Dragon NaturallySpeaking to improve the speech recognition performance in the future. Nuance's recommended system requirements include: dual core 2.2 GHz processor, 4 GB RAM and 4 GB + storage [52]. Thus the hardware of the human-robot interface computer should meet both sets of specifications.

Both Microsoft's and Nuance's speech software only work on Microsoft's Windows OS. The Microsoft Surface 3 Pro is a Windows based tablet. It is selected as MARVIN's human-robot interface computer. The specifications of the selected model are shown in table 3.5.

Table 3.5: Key characteristics of the Surface 3 Pro.

Parameter	**Value**
CPU	Intel Core i5-4300U Processor
CPU Cores	2
CPU Threads	4
CPU Frequency	1.9 GHz (2.9 GHz Boost)
RAM	4.0 GB
Graphics Processor	Intel HD Graphics 4400 (DX11)
Storage	128 GB (SSD)
Size ($w \times h \times d$)	$292\ mm \times 201\ mm \times 9\ mm$
USB Ports	$1\times$ USB3.0
Display	12″ 2160×1440 pixels (Touch)

From table 3.5 it can be seen that the Surface 3 Pro meets both Microsoft's and Nuance's recommended system requirements. It also has a high-resolution 12″ touch screen that can be used for further human-robot interaction, as explained in Chapter 7. The Surface is mounted on the front of MARVIN's torso, as shown in figure 3.10.

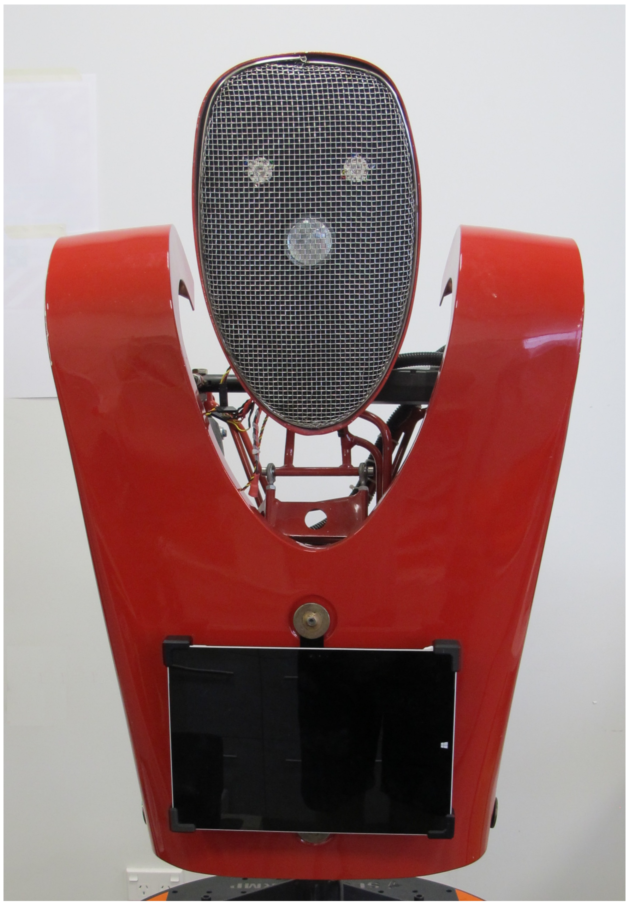


Figure 3.10: Surface 3 Pro, mounted on MARVIN's torso.

The addition of the Surface 3 Pro aligns with the receptive and expressive element objectives 2c and 2d.

3.3 Summary

This chapter introduced MARVIN's hardware platform. This included the inherited mobility platform, sensing systems, control hardware and humanoid torso; as well as the augmented sensing systems, control hardware, and human interface tablet. The structure of these hardware systems are visualised in figure 3.11.

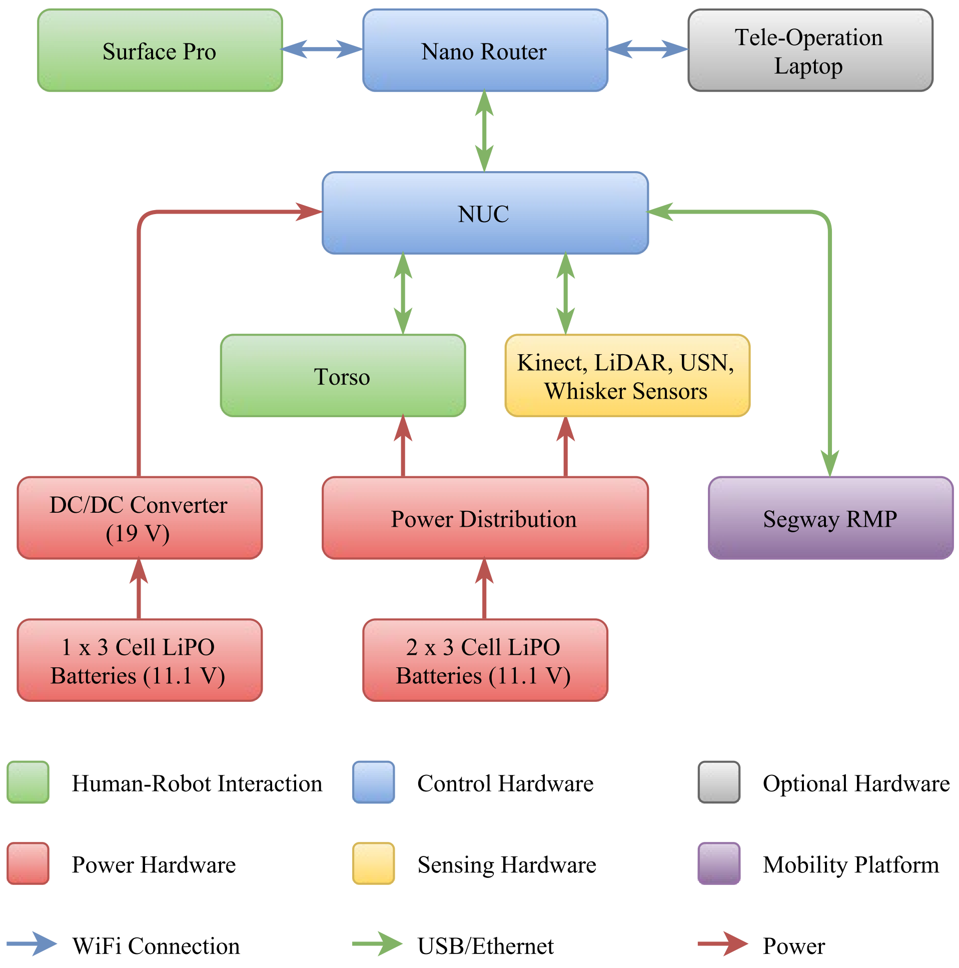


Figure 3.11: Overview of MARVIN's hardware platform.

The torso and sensing systems are powered by an inherited power distribution system. The NUC is powered by a 3 Cell LiPO regulated by a 19 V DC/DC converter. The Segway, Surface and Tele-Operation Laptop are powered by individual enclosed batteries. The NUC communicates to the Surface and the Tele-Operation laptop via the wireless local network provided by the nano router. The NUC communicates with the nano router via a physical Ethernet connection. The torso, sensors and Segway communicate directly with the NUC via USB connections.

The data from the sensors must be processed to account for the Segway's dynamic pitch before they can be used by the navigation stack and human-robot interaction. This is explained in Chapter 4.

Chapter 4

Sensor Processing

The data from the sensing systems outlined in Chapter 3 is used by MARVIN's navigation stack (further discussed Chapter 5) and human-robot interaction (detailed in Chapter 7) to perform localisation, obstacle detection and human tracking. To facilitate this, the sensor data must be processed to account for the Segway's dynamic pitch (previously explained in section 3.1.1) and converted into a format that the navigation stack and HRI can use. This chapter details the steps taken to process this sensor data.

Section 4.1 details the data format and coordinate systems used by the navigation stack. Section 4.2 explain the interfaces to the sensor hardware. Section 4.3 discusses the steps taken to process the laser rangefinder's and ultrasonic network's data. Section 4.4 discusses the novel algorithms developed to process the Kinect depth images.

4.1 Data Format

As explained in Chapter 5, MARVIN utilises a 2D navigation stack. Because of this, the measurements taken in the 3D world must be reduced to 2D and communicated to the navigation stack in a format it understands. This format is the ROS *LaserScan* message from the *sensor_msgs* package [53]. *LaserScan* messages store sensor data in polar coordinates, with a range and bearing. The range data is stored as a 1D array of 32-bit floats and the bearing data is communicated as a start angle (θ_{min}), end angle (θ_{max}) and the angular difference between two range measurements ($\theta_{increment}$). The size of the array should be equal to $\frac{\theta_{max}-\theta_{min}}{\theta_{increment}}$. In order to detect invalid measurements the minimum and maximum ranges are also stored (r_{min} and r_{max}). In addition to the array of ranges, an array of equal size can be used to store the measurement intensities. If the *LaserScan* message is generated by a laser rangefinder, this array stores the intensity of the reflected light. Figure 4.1 visualises the *LaserScan* message. The green regions represent valid measurements and the red regions represent invalid measurements (measurements with ranges outside of r_{min} and r_{max}).

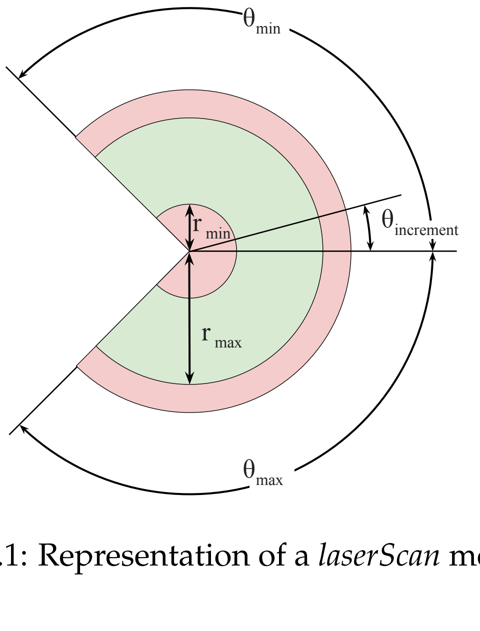


Figure 4.1: Representation of a *laserScan* message.

MARVIN uses a two primary Cartesian (x, y, z) coordinate frames, local and global. The local frame is relative to MARVIN's origin, while the global frame is relative to the world's origin. In addition to these primary coordinate frames, each sensor has their own frame of reference, referred to as secondary coordinate frames. When a sensor takes a measurement, it is relative to sensor's origin and on the sensor's individual coordinate frame. For the navigation stack to make use of this data, the measurement must be transformed from the secondary frame to the MARVIN's local frame. This is achieved in ROS using *tf* package [54]. As explained by [55], "the tf library was designed to provide a standard way to keep track of coordinate frames and transform data within an entire system".

The complexity of converting from the secondary frames to the primary frames is increased by the Segway's dynamic pitch. To avoid unnecessary complications, the local origin is chosen as the centre of the Segway's wheels as this is the only point that does not change with the Segway's pitch. This is visualised by figure 4.2.

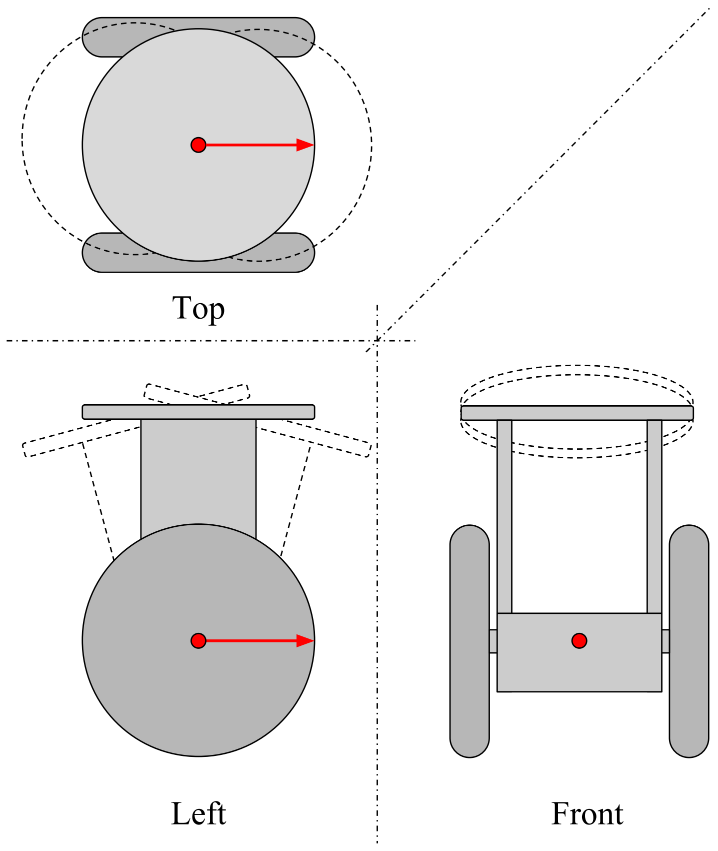


Figure 4.2: MARVIN's origin point, from which measurements are made. The red dot shows the origin point, the red line shows the origin vector.

As the sensors are mounted away from MARVIN's origin, their transforms must be recalculated every time the Segway's pitch is changed. Additionally the measurements must be projected down from three dimensions to two dimensions, while accounting for the Segway's pitch. This is detailed in section 4.3.

4.2 Sensor Interfaces

This section discusses the software interfaces between ROS and the sensing systems. Section 4.2.1 explains the Laser Range finder interface, section 4.2.2 discusses the Sensor Board interface and section 4.2.3 details the Kinect interface.

4.2.1 Laser Range Finder

As explained in section 3.1.2, MARVIN has inherited the Sick LMS100 laser rangefinder. This sensor communicates with with the NUC control computer using an Ethernet connection, via a USB to Ethernet adaptor. The *lms1xx* node [56] is utilised to interface the LMS100 with ROS to provide unprocessed *LaserScan* messages in the *laser/scan/raw* topic. This is shown by figure 4.3.

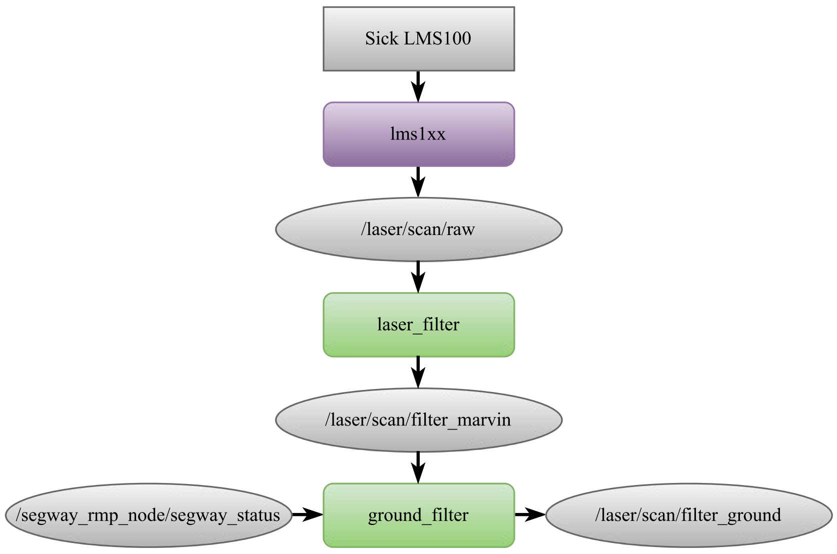

Sick LMS100
lms1xx
/laser/scan/raw
laser_filter
/laser/scan/filter_marvin
/segway_rmp_node/segway_status
ground_filter
/laser/scan/filter_ground

Figure 4.3: Block diagram of the LMS100 laser scanner's software interface.

Due to the mounting position of the LiDAR, it can detect parts of MARVIN's body. This results in false obstacles being detected, which prevents the navigation stack from operating correctly. To avoid this, the *laser/scan/raw* messages are first passed through a *laser_filter* [57]. The configuration file for this filter is shown in figure 4.4.

```
scan_filter_chain:
- name: laser_cutoff
  type: laser_filters/LaserScanAngularBoundsFilter
  params:
    lower_angle: -2.0
    upper_angle: 2.0
```

Figure 4.4: The configuration (.yaml) file for the laser filter.

In this configuration the *laser_filter* is set to a *LaserScanAngularBoundsFilter*, which removes all readings outside the angle range -2.0 to 2.0 radians. This removes the outer regions of the scan, which have detected MARVIN's body.

Finally the *laser/scan/filter_marvin* messages are passed through the *ground_filter* node, discussed in section 4.3, to account for the effect of the Segway's pitch. These processed *laser/scan/filter_ground* messages can then be used by the algorithms in the navigation stack, as explained in Chapter 5.

4.2.2 Sensor Interface Board

The whisker sensors, ultrasonic network and the Kinect's tilt servo are all interfaced with ROS via the interface board mentioned in section 3.1.2. This sensor interface board processes the sensor data and transmits it to control NUC via serial communication. This process is shown be figure 4.5.

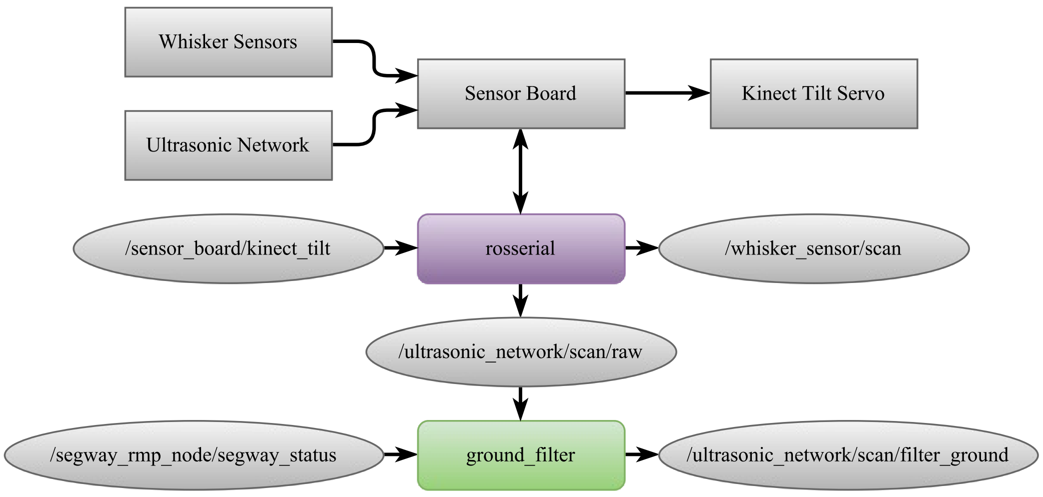

Whisker Sensors
Sensor Board
Kinect Tilt Servo
Ultrasonic Network
/sensor_board/kinect_tilt
rosserial
/whisker_sensor/scan
/ultrasonic_network/scan/raw
/segway_rmp_node/segway_status
ground_filter
/ultrasonic_network/scan/filter_ground

Figure 4.5: Block diagram of the sensor board's software interface.

The interface board communicates with ROS by utilising the *rosserial* library [58]. *rosserial* provides standardised methods for communicating through serial using ROS messages. The ultrasonic sensors' and whisker sensors' output pins are read by the Arduino, then processed with a rolling window filter to reduce noise. The processed data is converted to *LaserScan* messages and sent to ROS via the */ultrasonic_network/scan/raw* and */whisker_sensor/scan* topics. The */ultrasonic_network/scan/raw* messages are then further processed inside ROS by a *ground_filter* node, as explained in section 4.3. These processed messages are then accessed by the navigation stack via the */ultrasonic_network/scan/filter_ground* topic. The whisker sensor messages are used by both the *movement_control* node and navigation stack, as explained in section 6.1 and Chapter 5 respectively.

The Kinect's tilt servo is controlled using the Arduino *servo* library [59].This allows the Kinect to be tilted to 180 different positions and is set with 8-bit integers from the */sensor_board/kinect_tilt* topic.

4.2.3 Kinect

As explained in section 3.2.1, the Microsoft Kinect V2 RGBD camera is used to meet the 3D sensor objective (1b). As Microsoft developed the Kinect to run on Xbox One consoles and Windows computers, there is no native support for Linux or ROS. In order to interface the Kinect with Linux the open source *libfreenect2* driver [60] is utilised. This driver is bridged with ROS using the *iai_kinect2* package [61]. This package uses the OpenCL framework [62] to enable the connected computer's GPU to process the Kinect data. Without OpenCL support the Kinect data is processed with the CPU. This causes it to run at low frame rates (less than $1\ Hz$) which renders the sensor unusable for obstacle detection purposes. An open source implementation of the OpenCL specification, Beignet [63], was installed to enable the OpenCL libraries to run on the NUC's integrated Intel GPU.

The *kinect2_bridge* node from the *iai_kinect2* package communicates with the Kinect and uses ROS *image_transport* [64] messages to publish the depth and colour images. A depth image is a 2D grey scale image, where each pixel represents depth (the distance from the focal point of the camera to the measured point). The standard definition depth image (512$\times$424) is the primary image used by MARVIN and is published to the */kinect2/sd/image_-depth* topic. This topic is subscribed to by the *kinect_pipeline* node, as shown in figure 4.6.

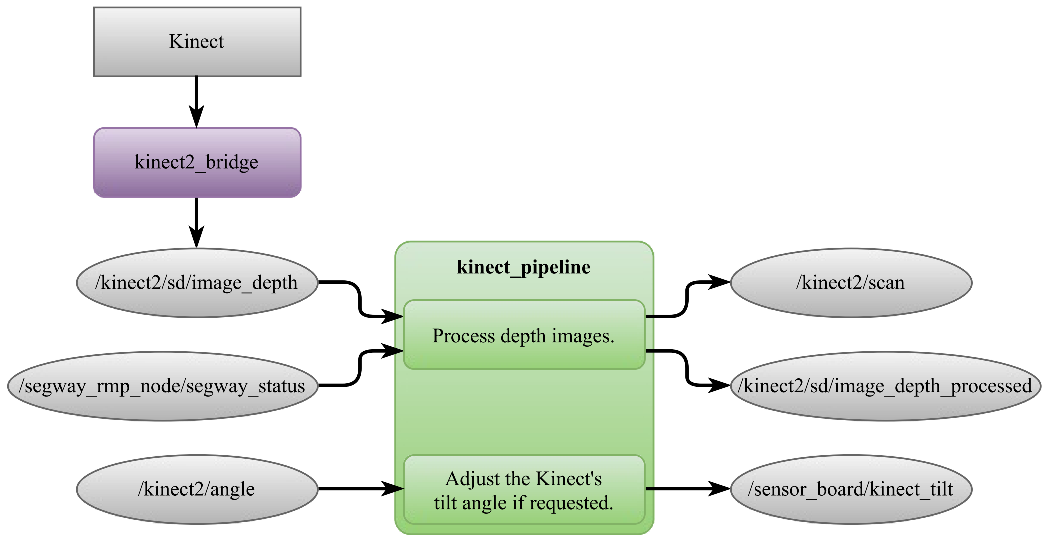

Kinect
kinect2_bridge
/kinect2/sd/image_depth
/segway_rmp_node/segway_status
/kinect2/angle
kinect_pipeline
Process depth images.
Adjust the Kinect's tilt angle if requested.
/kinect2/scan
/kinect2/sd/image_depth_processed
/sensor_board/kinect_tilt

Figure 4.6: Block diagram of the Kinects's software interface.

The *kinect_pipeline* node has three functions; it processes the depth images (explained in section 4.4.1); produces *LaserScan* messages for the navigation stack (also explained in section 4.4.1); and sends commands to the Sensor Interface Board to control the Kinect tilt servo.

As explained in section 4.2.2, the Kinect's tilt servo is set with 8-bit integers. The *kinect_pipeline* node receives angle requests from the */kinect2/angle* topic, which are in degrees, and converts them into 8-bit tilt value using equation 4.1.

$$T = m_{angle}\theta + c_{angle} \tag{4.1}$$

where T is the tilt value and θ is the requested angle. The constants m_{angle} and c_{angle} are calculated during the auto calibration process explained in section 4.4.2.

4.3 Laser Scan Processing

This section explains how the *laser_scan* messages produced by the LiDAR and the Ultrasonic Network are processed. Because these sensors have planar measurement areas, as shown in section 3.1.2, the changing Segway pitch can result in large measurement errors if not handled correctly. These errors are caused by two factors; the origin (mounting location) of the sensors; and the measured distance when projected to two dimensions. This is shown in figure 4.7.

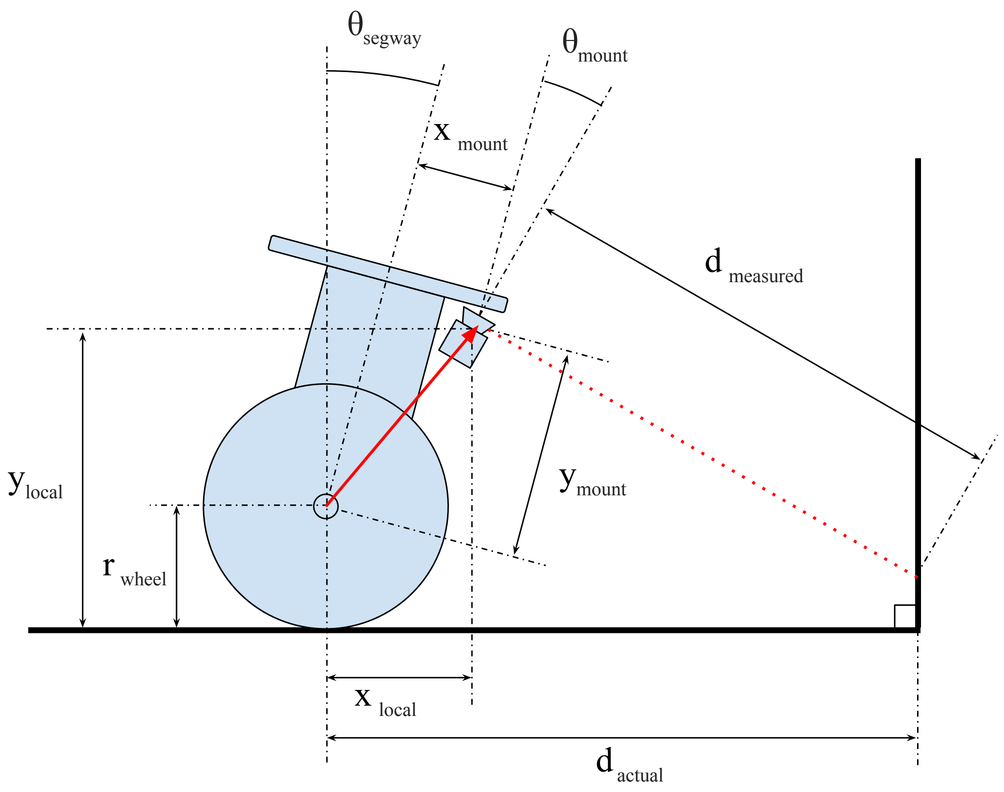


Figure 4.7: Diagram of the pitch filtering algorithm, using the LiDAR as an example.

The red arrow shows the vector from MARVIN's origin to the LiDAR's origin. The red dotted line represents the LiDAR measurement. As the Segway's pitch increases in the forward direction, figure 4.7 shows two issues; firstly, the origin of the sensor relative to MARVIN's origin moves forward, which reduces the measured distance; secondly, as the sensor is angled downwards it measures a longer distance (assuming it is measuring a surface which is perpendicular to the ground). The process used to correct these errors out is shown by figure 4.8.

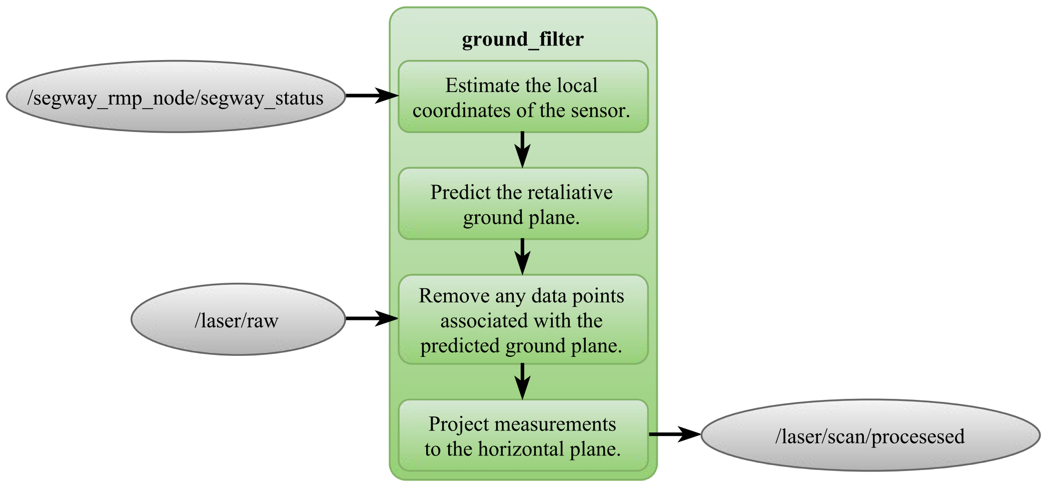

ground_filter
/segway_rmp_node/segway_status
Estimate the local coordinates of the sensor.
Predict the retaliative ground plane.
/laser/raw
Remove any data points associated with the predicted ground plane.
Project measurements to the horizontal plane.
/laser/scan/procesesed

Figure 4.8: Block diagram of the sensor's ground and pitch correcting algorithm.

The first step is to calculate the sensor transform, explained in section 4.3.1. The next step predicts the position of the ground plane relative to the sensor's frame of reference and removes any measurement points associated with that plane (explained further in section 4.3.2). The final step is to account for the Segway pitch by projecting the measurements to the global horizontal plane, as explained in section 4.3.3.

The *ground_filter* node's code can be found in appendix C.

4.3.1 Calculating the Sensor Transform

Calculating the transform from MARVIN's origin to the sensor's origin is performed in two steps. Firstly, when the node is initialised, the polar coordinates of the sensor (represented by the red arrow in figure 4.7) are calculated from the mounting variables, x_{mount} and y_{mount}, using equation 4.2. Secondly, these polar coordinates, r_{mount} and b_{mount}, are used in conjunction with the Segway's pitch θ_{segway} to calculate the local coordinates, x_{local} and y_{local}, using equation 4.3. It should be noted that the y_{local} coordinate also needs to account for the radius of the Segway's wheel, r_{wheel}.

$$\begin{aligned} r_{mount} &= \sqrt{(x_{mount})^2 + (y_{mount})^2} \\ b_{mount} &= \tan^{-1}\left(\frac{x_{mount}}{y_{mount}}\right) \end{aligned} \tag{4.2}$$

$$\begin{aligned} x_{local} &= \sin\left(b_{mount} + \theta_{segway}\right) \times r_{mount} \\ y_{local} &= \cos\left(b_{mount} + \theta_{segway}\right) \times r_{mount} + r_{wheel} \end{aligned} \tag{4.3}$$

4.3.2 Predicting the Relative Ground Plane

At full acceleration, the Segway's pitch can tilt up to $\approx 18°$, which causes the sensors to detect the ground plane. This causes the navigation stack to detect false obstacles, which results in jerky movement (as the motion planner tries to avoid the false obstacles). This issue can be avoided by filtering out any measured points which are likely to be part of the ground plane. This is accomplished in three steps:

First, the distance from the local coordinates of the sensor to the ground plane is calculated using equation 4.4. If this distance is outside the valid range of the sensor, it can be assumed that the sensor is unable to detect the ground plane at this pitch and the following steps are ignored.

$$d_{ground} = \frac{y_{local}}{\sin(\theta_{mount} + \theta_{segway})} \tag{4.4}$$

Second, as explained in section 4.1, the laser rangefinder and ultrasonic network use *LaserScan* messages to return their measured ranges. Because these measurements are polar and the ground has a flat surface, the estimated ground distance has to be adjusted for each n range measurement in the array as shown in figure 4.9. This is achieved through equation 4.5.

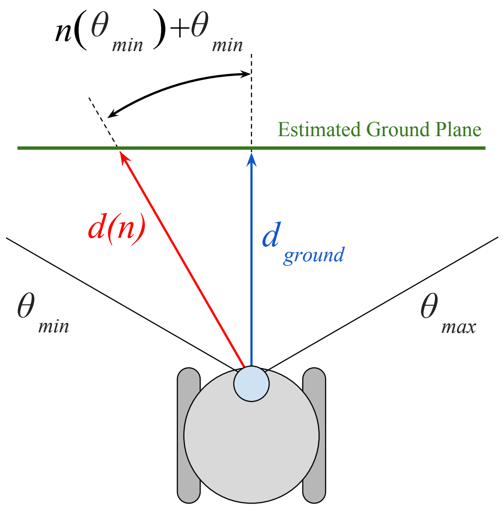


Figure 4.9: Accounting for the polar distortion of the ground plane estimates.

$$d(n) = \frac{d_{ground}}{\cos(n \times \theta_{increment} + \theta_{min})} \tag{4.5}$$

Finally, any points that are within a threshold, $d_{threshold}$, of the ground estimate, $d(n)$, are made invalid by setting them to r_{min}, as shown in 4.6.

$$r(n) = \begin{cases} r_{min} & \text{if } |r(n) - d(n)| \leq d_{threshold} \\ r(n) & \text{otherwise} \end{cases} \tag{4.6}$$

4.3.3 Project Measurements to the Horizontal Plane

Equation 4.7 is used to project the ground filtered range measurements to the horizontal plan, effectively removing the measurement error caused by the Segway pitch.

$$r(n) = \cos(\theta_{segway}) \times r(n) \tag{4.7}$$

It should be noted that equation 4.7 relies on the assumption that all objects have surfaces that are perpendicular to the ground (parallel with the global vertical axis). This is a reasonable assumption to make for the LiDAR and Ultrasonic sensors as their primary purpose is to measure the walls and glass panels, which are vertical surfaces. For general obstacle measurements, this assumption can become inaccurate. The Kinect is the primary obstacle detection sensor and is processed using the algorithms explained in section 4.4, which do not rely on this assumption.

This projection step could have also been achieved with a mechanical gimbal for the laser rangefinder, however this would have increased the complexity and price of the system, as well as introducing mechanical lag. For these reasons it was decided that the software solution was more appropriate. Due to the way the ultrasonic network is mounted it would be infeasible to use a mechanical gimbal (this can be seen in figure 3.4a). It should also be noted that this final step could be removed if 3D data was desired in the future, if for example a 3D navigation stack was used.

The full code for the *ground_filter* node can be found in Appendix C.

4.4 Kinect Pipeline

This section explains the steps required to take the Kinect's unprocessed depth images and produce the data required for the navigation and interaction stacks. This includes; noise filtering, ground and ceiling removal, calibration algorithms, and human detection. The Kinect's driver (explained in section 4.2.3) has a large computational overhead, so it is desired that these processing steps are computationally efficient to reduce the overall impact to the control system.

4.4.1 Processing Steps

The primary goal of processing the Kinect depth images is to detect the location of obstacles. To facilitate this, any pixels that don't contain obstacles or humans must be removed. This includes any noise, the ground plane and the celling plane. Ground removal from depth data is a common problem and is typically solved by converting the depth images into a point-cloud. Once the data is in point-cloud form the ground plane is usually detected using a robust RANSAC (random sample consensus) plane detection algorithm. This approach has been used by many robotic systems ranging from humanoid robots [65] to UAVs (unmanned flying vehicles) [66]. This method can be implemented easily using the open source PCL (point cloud library) [67]. However, this approach has a number of issues. The conversion from the depth image to a point cloud increases the computational overhead. This overhead would be acceptable if the point-clouds were useful, but none of MARVIN's other systems use point-cloud data. Due to point clouds' inherent computational cost they are often reduced down and stored in voxel grids [25, 68], but this loses detail. RANSAC based algorithms tend to require large numbers of data points fitting the linear function they are trying to detect. However, this is not always possible, for example if most of the Kinect's FOV is taken up by an obstacle and there is ground visible around the edges. The edge ground points may not be properly detected and therefore not removed. RANSAC algorithms are also not guaranteed to find accurate solutions. To reduce computation time, RANSAC selects random points until a "good enough" solution is found. The more iterations it runs through, the better the solution, meaning the effectiveness of RANSAC based algorithms is inversely proportional to their computational efficiency.

To avoid these issues and to reduce the computational overhead, all of the Kinect's data is processed using 2D image processing techniques. Additionally, rather than detecting planes in every frame, the Kinect's mounting position (in local coordinates) is used to predict where the ground plane should be. This is more robust than frame by frame approaches and much less computationally expensive. These processing steps are shown in figure 4.10.

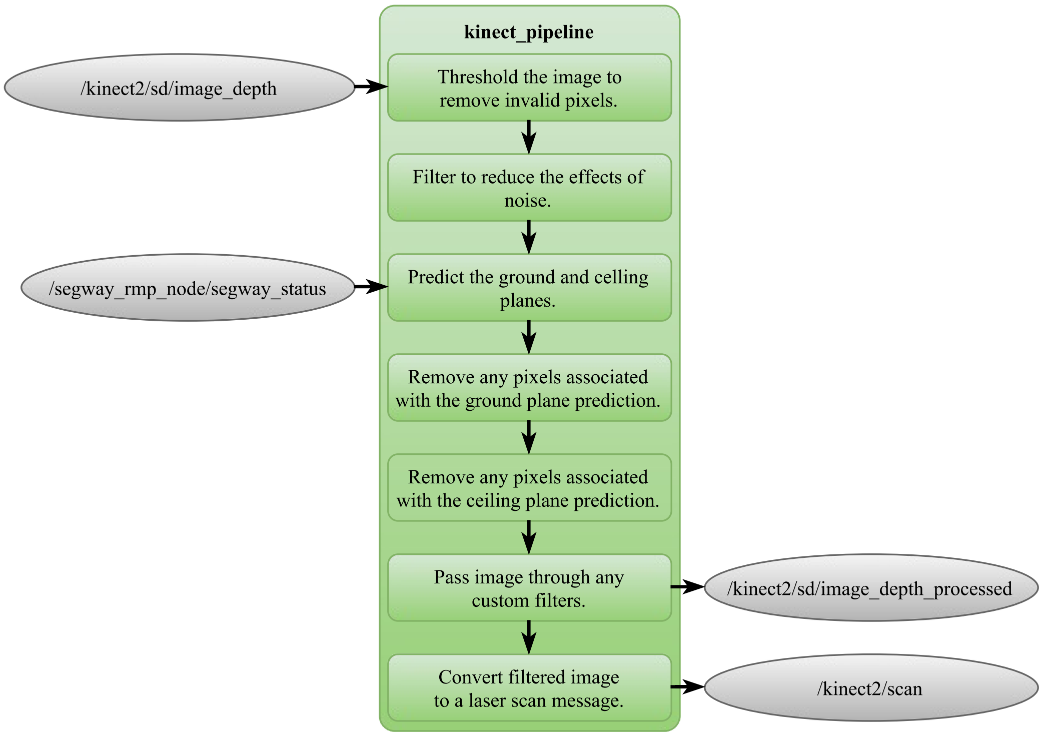

kinect_pipeline
/kinect2/sd/image_depth
Threshold the image to remove invalid pixels.
Filter to reduce the effects of noise.
/segway_rmp_node/segway_status
Predict the ground and celling planes.
Remove any pixels associated with the ground plane prediction.
Remove any pixels associated with the ceiling plane prediction.
Pass image through any custom filters.
/kinect2/sd/image_depth_processed
Convert filtered image to a laser scan message.
/kinect2/scan

Figure 4.10: Kinect depth image processing steps.

The majority of these steps make use of the open source computer vision library, *OpenCV* [69], which provides highly efficient implementations of computer vision algorithms. The *kinect_pipeline* node publishes two output topics, */kinect2/scan* used for the navigation stack's obstacle avoidance explained in section 5.2 and */kinect2/sd/image_depth/processed* used by the human detection algorithm explained in section 4.4.3.

Input Depth Image

As explained in section 4.2.3, the *kinect_pipeline* node subscribes to the standard definition depth image produced by the Kinect, an example of which is shown in figure 4.11.

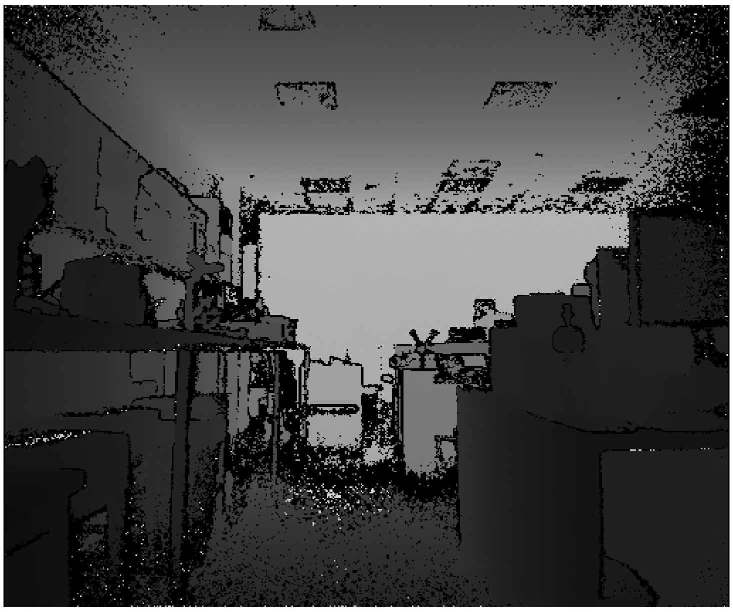


Figure 4.11: Raw Depth Image.

This depth image is configured so that the pixel values are proportional to the measured depth. The pixels are 8-bit grey scale values ranging from black (0) being the minimum depth ($0\ m$) to white (255) being the maximum depth ($12\ m$).

Prepossessing the Depth Image

The Kinect works by projecting infrared (IR) rays in a specific pattern. It measures the time-of-flight of the IR light using a IR camera. The longer the light takes to illuminate a pixel, the further away the measured surface and the larger the pixel value. This method works well for flat surfaces, but information can be lost if the IR rays reflect away from the camera. The Kinect returns these invalid measurements as 0 valued pixels. These pixels are removed by passing the image through the threshold described by equation 4.8, where v is the pixel value.

$$v = \begin{cases} 255 & \text{if } v < 2 \\ v & \text{otherwise} \end{cases} \tag{4.8}$$

The depth image also needs to be filtered to reduce the effects of the measurement noise. This is achieved using the OpenCV *morphologyEx* function [70] using the *MORPH_OPEN* kernel. This function applies morphological transforms to images. Morphological transforms are simple operations based on the image shape through convolution. The type of transform is determined by the kernel which is the structuring element that is applied to the image. The *MORPH_OPEN* kernel performs the opening transform, which is equivalent to performing an erosion transform followed by a dilation transform. Erosion "erodes away the boundaries of the foreground object" [70]. This essentially reduces the size of the foreground regions in the image. Dilation is essentially the opposite of erosion, it makes the foreground regions larger. Noise can be considered as small regions of foreground surrounded by background, or small regions of background surrounded by foreground. By eroding the image the first type of noise can be removed, but this also removes some of the valid data. This lost data can be recovered by dilating the image, which returns the foreground regions back to their original size, but as the noise has been completely removed those regions cannot be returned. The dilation step also simultaneously removes the second type of noise by expanding the foreground regions and "filling in" the holes left by the noise. The result of the threshold and noise reduction is shown in figure 4.12.

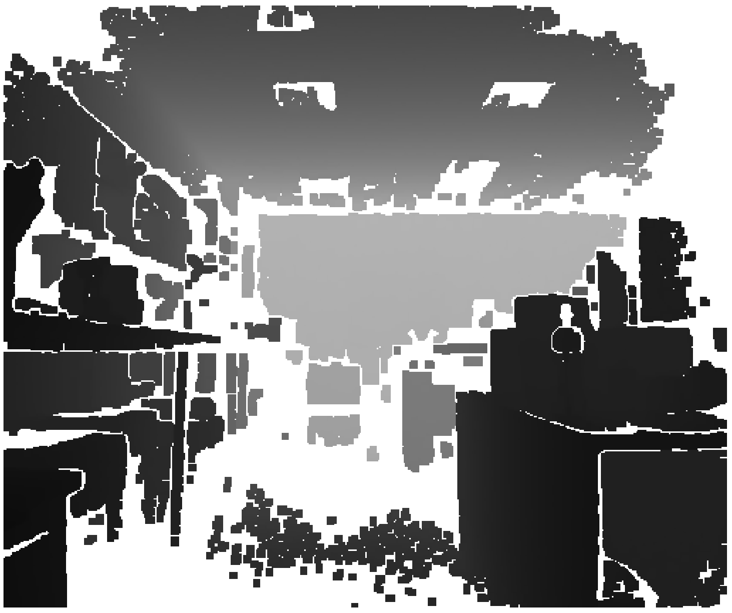


Figure 4.12: Preprocessed Depth Image.

Predicting the Relative Ground and Ceiling Planes

The next processing step is to determine which pixels are relevant to the navigation stack and human detection algorithm. The navigation stack needs to know which pixels are obstacles that MARVIN might collide with and the human detection algorithm needs to know which pixels might contain humans. Due to the Kinect's vertical FOV it can detect the ground and the region of space above the top of MARVIN (set at $2\ m$), which will be referred to as the ceiling plane. Neither region can contain obstacles nor humans, so any pixels in these regions should be removed.

Before removing these pixels, they must associated with either the ground or ceiling plane. To do this we must predict what each pixel value in the image would be if it was detecting the ground plane and the ceiling plane.

Any pixels within a threshold of the predictions can be removed. To make these predictions we need to know what the depth image represents in real world values.

The depth image produced by the *kinect_bridge* node is a normalised 8-bit depth image. This means that the values of each pixel represents the distance from the focal plane to the measured point, not the distance from the focal point to the measured point. This is shown in figure 4.13.

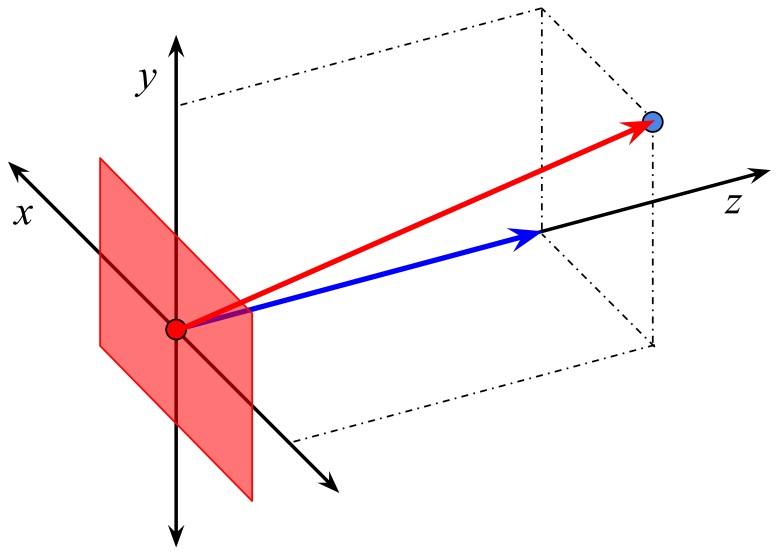


Figure 4.13: Normalised depth image explanation.

The red dot represents the focal point (the Kinect's origin), the red plane represents the focal plane and the blue dot represents the measured point. The red vector is the distance between the measured point and the focal point. Rather than the pixels of the image representing the red vector, they represent the blue arrow. The real-world value (in metres) of the blue vector can be calculated using equation 4.9, where d_z is the distance along the z axis, v is the pixel value, v_{max} is the maximum pixel value (255) and z_{max} is the maximum distance ($12\ m$).

$$d_z = \frac{v}{v_{max}} \times d_{max} \tag{4.9}$$

Because the pixel values represent d_z, the distortion normally created by the pin-hole camera model can be ignored. This is best visualised by aiming the Kinect at a flat surface, parallel to the focal plane. If the pixel values returned the measured distance, the centre of the plane would look closer than the edges because the distance to the focal point is longer. But because the depth image is normalised, the same pixel value is returned for the whole surface.

For this reason, and because the Kinect and Segway only rotate in pitch and yaw (not roll) axes, the ground and ceiling planes return the same values for all the columns of the image, but different values for the rows of the image. This means that we only need to predict the ground and ceiling planes for one column of the image, then apply it to all the rows.

Using the information shown in figure 4.14 and the Kinect's FOV we can predict equations that represent the ground and ceiling planes.

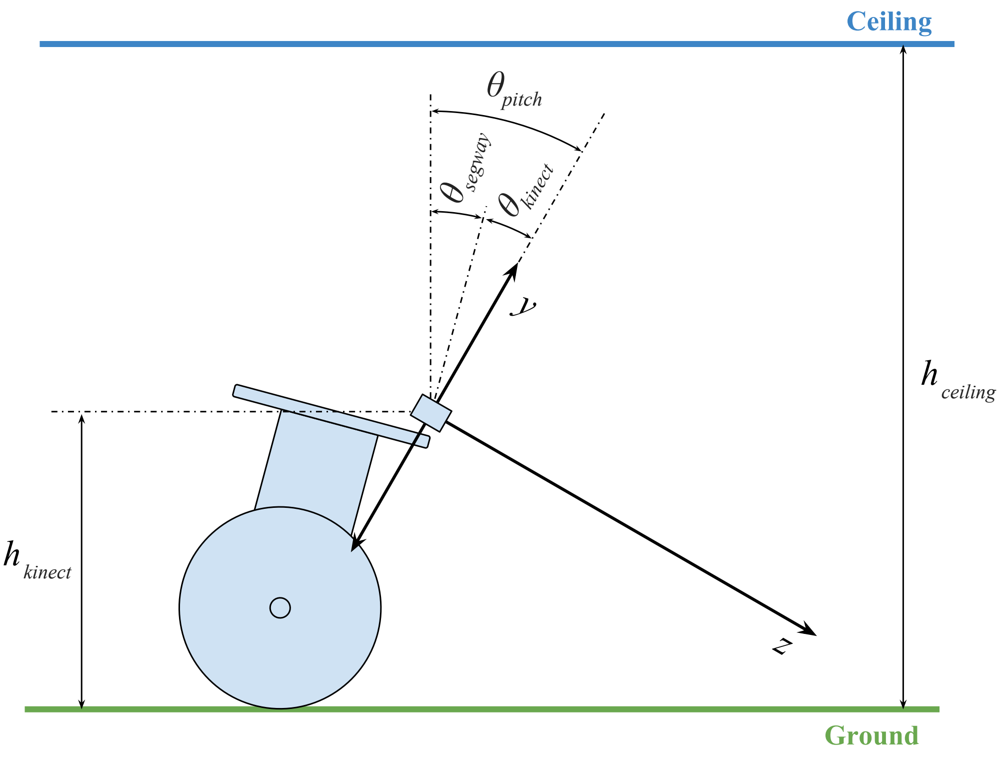


Figure 4.14: Kinect's frame of reference.

θ_{pitch} is the angle between the Kinect's frame of reference and the global frame of reference. It is the combination of the Segway's pitch, θ_{segway}, and the Kinect's angle, θ_{kinect}. h_{kinect} is the distance between the ground plane and the Kinect, it is calculated in the same way as the other sensor transforms, discussed in section 4.3.1. $h_{ceiling}$ is the distance between the ground plane and the ceiling plane. This distance is user defined, and is not necessarily the same height as the actual ceiling. Currently it is set to $2\ m$ as MARVIN is $\approx\ 1.8\ m$ tall. Figure 4.15 shows the ground and ceiling planes relative to the Kinect's reference frame. The y and z axes are as labelled in figure 4.14.

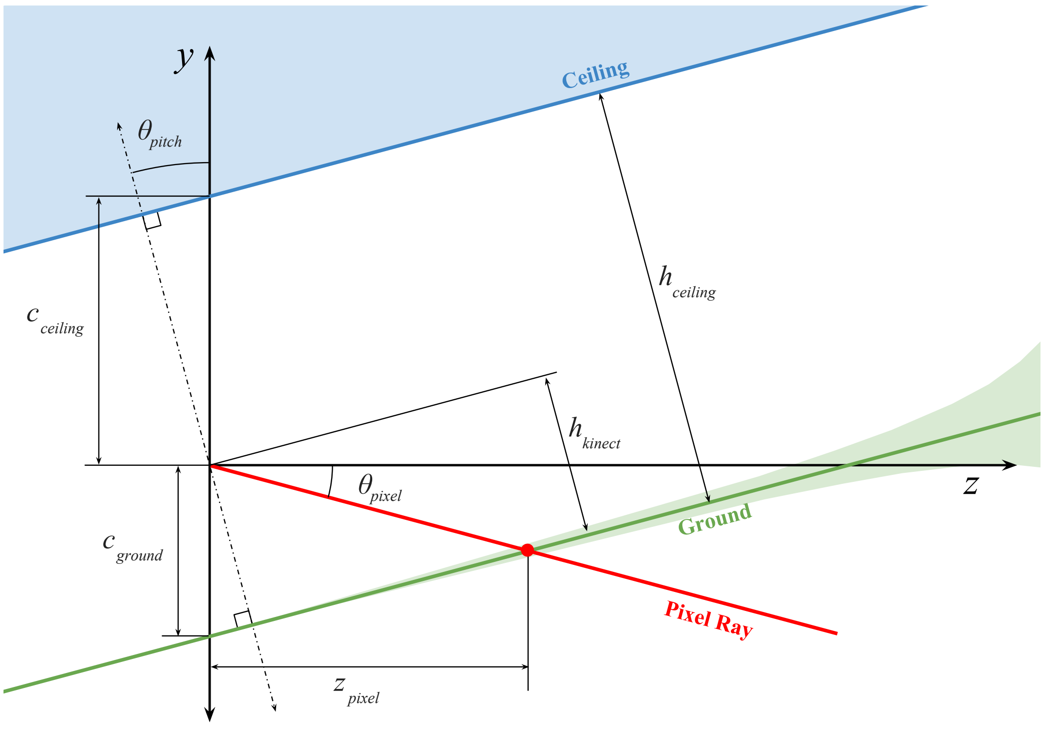


Figure 4.15: Explanation of the Kinect ground removal equations.

The ground and ceiling planes are represented by the blue and green lines. The light blue area is the region associated with the ceiling, and the light green area is the region associated with the ground. The red line represents a IR ray that is reaching one pixel row. The angle of this ray can be determined from the pixel row and the Kinect's FOV using equation 4.10, where θ_{vfov} is the Kinect's vertical FOV ($60°$), h_{image} is the image height (424 pixels) and r is the pixel row.

$$\theta_{pixel} = \frac{r - \frac{h_{image}}{2}}{h_{image}} \times \theta_{vfov} \tag{4.10}$$

To predict what the pixel values will be for the ground and ceiling planes, we need to know the z value of the interception between the pixel ray line and the ground and ceiling lines. This is shown as z_{pixel} and the red dot in figure 4.15.

These lines can be expressed using the standard linear equation 4.11 and the intercept between two lines can be found with equation 4.12.

$$y = mx + c \tag{4.11}$$

$$x = \frac{c_1 - c_2}{m_2 - m_1} \tag{4.12}$$

Because the ground and ceiling planes are parallel to each other the equations share the same m constant, calculated with equation 4.13.

$$m = \tan(\theta_{pitch}) \tag{4.13}$$

The c constant is separate for each plane and is determined by h_{kinect}, $h_{ceiling}$ and θ_{pitch} using equation 4.14.

$$\begin{aligned} c_{ground} &= \cos(\theta_{pitch}) \times h_{kinect} \\ c_{ceiling} &= \cos(\theta_{pitch}) \times h_{ceiling} \end{aligned} \tag{4.14}$$

In summary, the ground, ceiling and ray lines can be expressed using equations 4.15, 4.16 and 4.17 respectively.

$$y = mz + c_{ground} \tag{4.15}$$

$$y = mz + c_{ceiling} \tag{4.16}$$

$$y = \tan(\theta_{pixel})z \tag{4.17}$$

From equation 4.12, the predicted z values for the ground and ceiling planes can be found using equation 4.18.

$$\begin{aligned} z_{ground} &= \frac{c_{ground}}{\tan(\theta_{pixel}) - m} \\ z_{ceiling} &= \frac{c_{ceiling}}{\tan(\theta_{pixel}) - m} \end{aligned} \tag{4.18}$$

The z_{ground} and $z_{ceiling}$ values are then converted to pixel values using equation 4.19 and saved in an array so that they can be compared to all the columns of the depth image.

$$v = \frac{z}{d_{max}} \times v_{max} \tag{4.19}$$

Removing the Ground and Ceiling Planes

Using the arrays of predicted ground and ceiling values, the pixels can now be associated and removed. This is done using the threshold shown in equation 4.20, where v is the current pixel value, p_{ground} is the predicted ground value, $p_{ceiling}$ is the predicted ceiling value and t is the threshold.

$$v = \begin{cases} 255 & \text{if } |v - p_{ground}| < t(v) \\ 255 & \text{if } v > p_{ceiling} \\ v & \text{otherwise} \end{cases} \tag{4.20}$$

The measurement error in the depth image is typically proportional to the depth of the pixel. The further away the measured point, the more noise. This is accounted for by using a user-defined non-linear threshold that is dependent on the value of the pixel, $t(v)$. The currently defined threshold function is shown in figure 4.16.

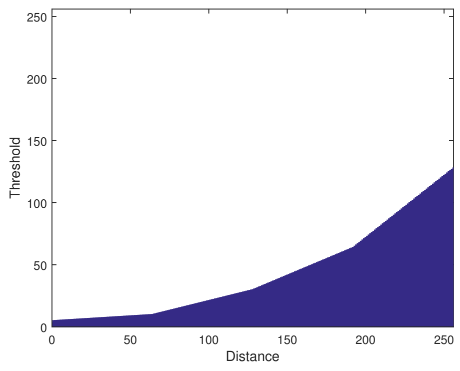
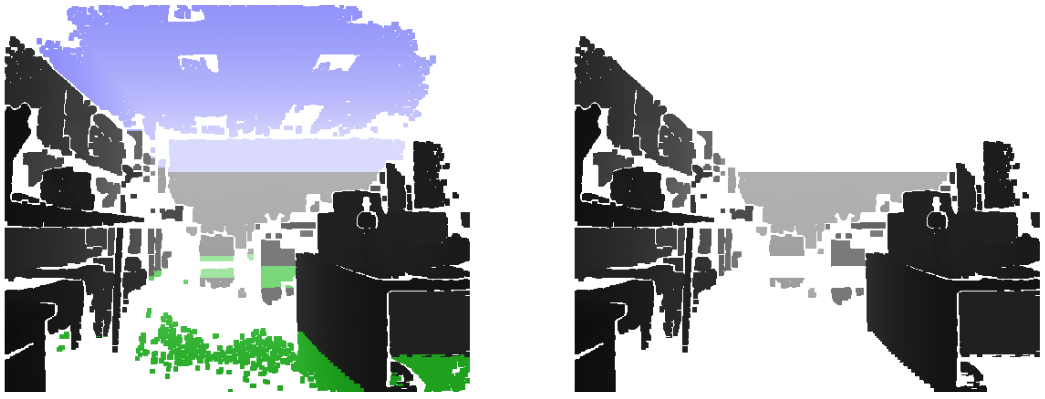


Figure 4.16: Distance dependent threshold for ground filtering.

Figure 4.16 shows that as the pixel value increases, so does the threshold. The effective areas removed by this threshold are shown in the green and blue regions in figure 4.15. Figure 4.17a shows the pixels associated with the ground and ceiling planes. Green represents the ground and blue represents the ceiling. Figure 4.17b shows the depth image after the planes have been filtered.



(a) Ground and ceiling plane associations.

(b) Ground and ceiling planes removed.

Figure 4.17: Ground and ceiling plane filtering.

It should be noted that the figures used to explain the Kinect's processing steps are taken in a vinyl floored room. The vinyl can cause reflections in the IR projections, which can make some vertical surfaces appear to extend through the ground plane. This can be observed as the isolated pixel regions in the centre of figure 4.17b. This is not an issue, as the conversion to *laser_scan* messages only takes into account the nearest pixel value of each column. This can be seen in figure 4.20b.

To improve the performance of these processing steps, all three of the threshold processes (invalid pixel removal, ground removal and ceiling removal) are conducted in a single paralleled *for* loop. By using the *#pragma omp parallel for* command, the *for* loop is processed on multiple threads further improving the performance. Figure 4.18 presents the code to implement this.

```
 1 void removePlanes(const cv::Mat &in, cv::Mat &out){
 2   cv::Mat tmp = cv::Mat(in.rows, in.cols, CV_8U);
 3   #pragma omp parallel for
 4   for(int r = 0; r < in.rows; ++r) {
 5     const uint8_t *itI = in.ptr<uint8_t>(r);
 6     uint8_t *itO = tmp.ptr<uint8_t>(r);
 7     for(int c = 0; c < in.cols; ++c, ++itI, ++itO) {
 8       if ((*itI < 2) || ((*itI > (ground_plane_est[r] - threshold[*itI])) && *itI <
   (ground_plane_est[r] + threshold[*itI]))) || (*itI > celling_plane_est[r])) {
 9         *itO = 255;
10       }else{
11         *itO = *itI;
12       }
13     }
14   }
15   tmp.copyTo(out);
16 }
```

Figure 4.18: Function for removing invalid, ground plane and ceiling planes pixels.

Line 1 declares the *removePlanes* function. Its parameters takes two pointers to *OpenCV* matrices. One pointer is to the input image and the other is to an output image. Line 2 initialises a temporary matrix for making dynamic changes. Line 3 sets the *for* loop to run in parallel. Lines 5 and 6 initialise pointers for accessing the input and temporary matrices. Lines 8 to 12 perform the invalid pixel removal, ground removal and ceiling removal operations. Line 15 copies the temporary matrix to the output matrix.

Creating a Laser Scan Message

For the navigation stack to use the depth image data, it must be converted into a *laser_scan* message. The smallest pixel value (nearest measurment) is selected from each column of the image. Each of these values are converted into distances and saved in a *laser_scan* message (with the same number of points as the image width) and published to the */kinect2/scan* topic. Because the *laser_scan* message contains its depth data in a polar form, the Kinect's horizontal FOV must be taken into account, as shown in figure 4.19

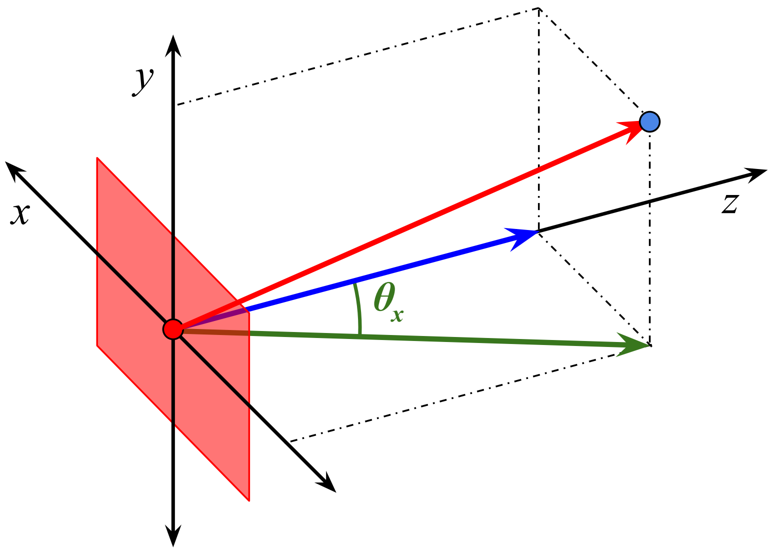


Figure 4.19: Converting the depth image to a laser scan message.

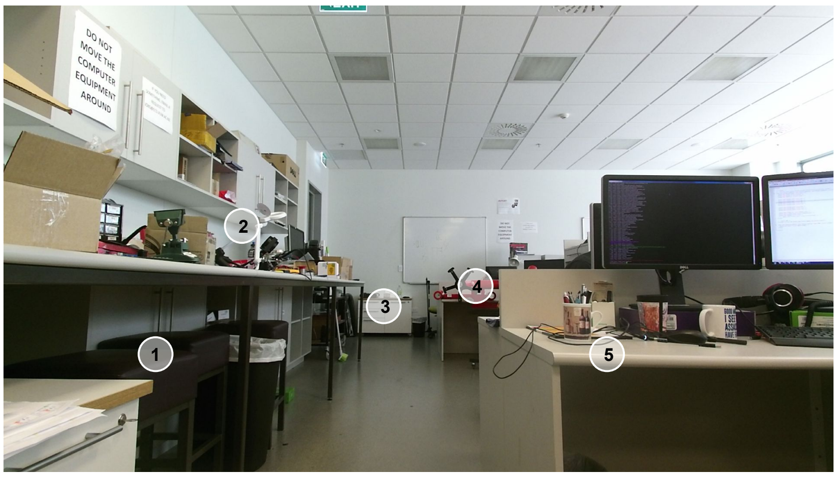
The pixel values represent the blue vector, but the *laser_scan* data points represent the green vector. θ_x is dependent on the pixel row and is calculated using equation 4.21, where c is the pixel column, w_{image} is the image width (512) and θ_{hfov} is the horizontal FOV ($70.6°$).

$$\theta_x = \frac{c - \frac{w_{image}}{2}}{w_{image}} \times \theta_{hfov} \tag{4.21}$$

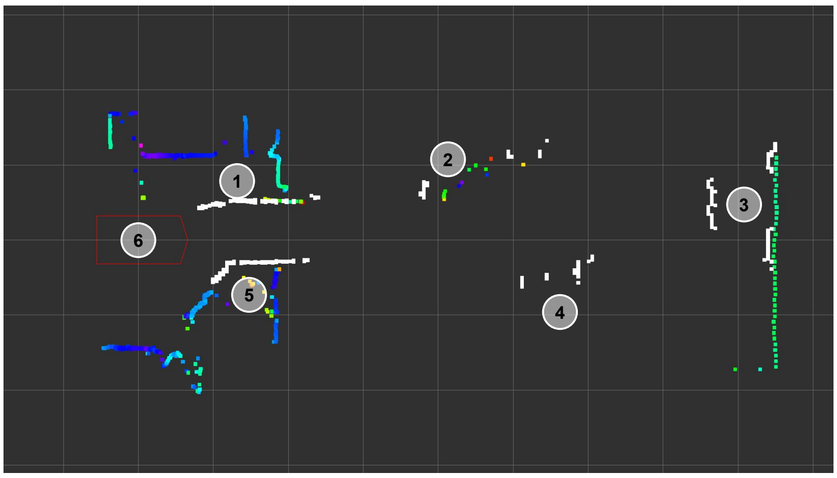
The length of the blue vector, d_z, and be calculated as before using equation 4.9. The length of the green vector, d_x, can be calculated from d_z using equation 4.22.

$$d_x = \frac{d_z}{\cos(\theta_x)} \tag{4.22}$$

Figure 4.20 shows the output laser scan, compared to the Kinect's RGB image. Points of interest have been marked so the laser scan can be more easily compared to the image and are labelled in table 4.1. The white points are the Kinect's laser scan, the blue-green points are the LiDAR's laser scan. The grid in figure 4.20b shows 1 m increments.



(a) RGB Image



(b) Laser Scan

Figure 4.20: Kinect Pipeline Output.

Table 4.1: Description of points of interest.

1	Cabinet and bench stools under the side bench.
2	Equipment on side bench.
3	Cabinet at the far side of the room.
4	Robot behind front desk.
5	Front desk.
6	MARVIN's location.

Figure 4.20b was created by taking a screen-shot from *rviz* [71], which a ROS program used to visually display the robot's sensor data. By comparing the LiDAR and Kinect laser scans, it can be seen that the Kinect often returns closer measurements. This is because it has a vertical FOV which allows it to detect objects that are above or below the LiDAR's planar detection region. Figure 4.20b also shows the advantage of the LiDAR's larger horizontal FOV, allowing it to detect obstacles beside and slightly behind MARVIN, which the Kinect cannot detect.

As demonstrated in section 8.1, noise can be introduced into the Kinect's depth image if the Segway's pitch suddenly changes. This is shown by figure 4.21a. This noise is not guarantied to be removed by the previous processing steps and can be incorporated into the *LaserScan* messages. This issue is accounted for by monitoring the confidence of each measurement. To confidence is calculated by calculating the rolling window average μ_d of each distance measurement $d(n)$. The intensity $i(n)$ is calculated using equation 4.23, where $d_{threshold}$ is a user defined value which can be tuned to optimise performance.

$$i(n) = 1 - \frac{d(n) - \mu_d}{d_{threshold}} \tag{4.23}$$

The produced *LaserScan* message can then be filtered using a similar method to section 4.2.1, using a *laser_filter* node. However, for this purpose the node is configured as a *LaserIntensityFilter* which removes any data points with an intensity less than 0.2. The output of this is shown in figure 4.21b.

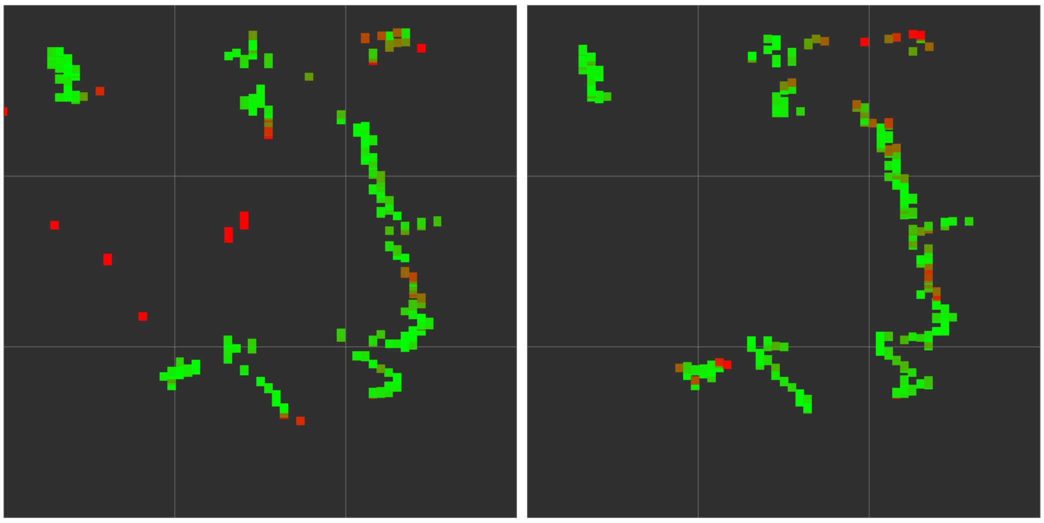

(a) Without Intensity Filter (b) With Intensity Filter

Figure 4.21: Kinect intensity filter.

The full code for the *kinect_pipeline* node can be found in Appendix C.

4.4.2 Auto Calibration

The processing steps discussed in section 4.4.1 requires the accurate local coordinates of the Kinect. The Segway returns accurate pitch measurements through the */segway_rmp_node/status* topic and the local coordinates can be calculated using the method explained in section 4.3.1. However, this requires accurate mounting position measurements, x_{mount}, y_{mount} and θ_{kinect} (see figure 4.23). If these measurements are inaccurate, the ground and ceiling plane predictions do not work effectively and the *kinect_pipeline* produces unusable outputs. It is difficult and time consuming to measure these mounting variables with sufficient accuracy to produce viable results. To avoid this issue an auto calibration algorithm is developed. The steps of the calibration process are shown in figure 4.22.

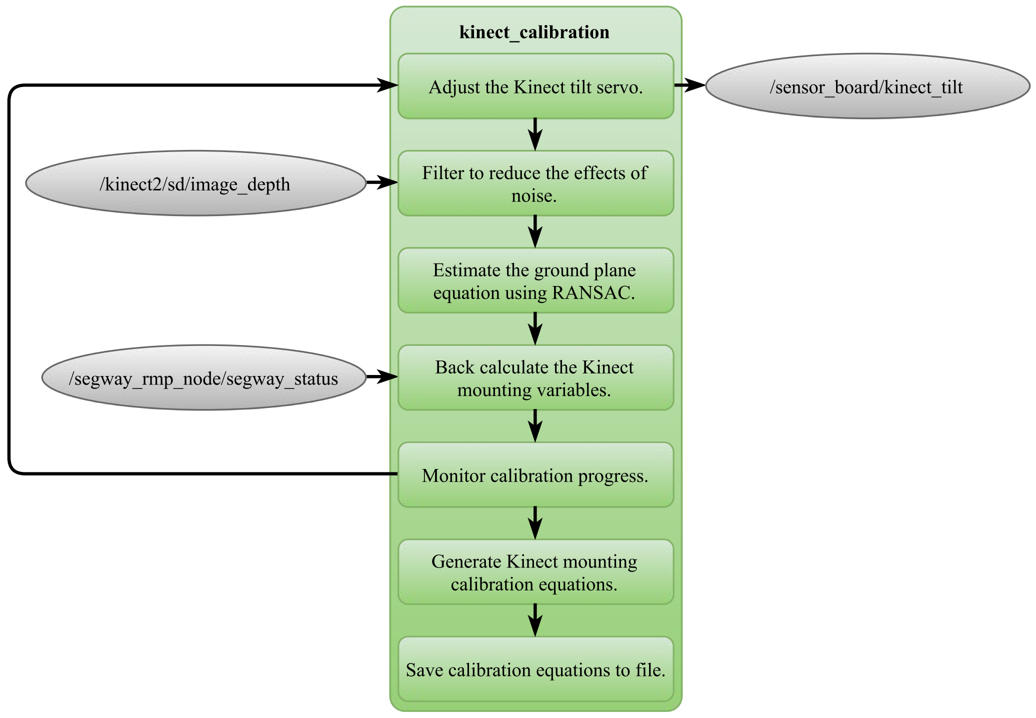

kinect_calibration
Adjust the Kinect tilt servo.
/sensor_board/kinect_tilt
/kinect2/sd/image_depth
Filter to reduce the effects of noise.
Estimate the ground plane equation using RANSAC.
/segway_rmp_node/segway_status
Back calculate the Kinect mounting variables.
Monitor calibration progress.
Generate Kinect mounting calibration equations.
Save calibration equations to file.

Figure 4.22: Block diagram of the Kinect auto calibration algorithm.

Because the Kinect's angle can be changed with the servo, the mounting variable θ_{kinect} must be calibrated over the range of possible Kinect tilt positions. This is to generate the m_{angle} and c_{angle} constants for the conversion from angle requests to servo values, outlined in section 4.2.3 (equation 4.1).

The first step requests the servo to move to required tilt position by publishing to the */sensor_board/kinect_tilt* topic. The depth data is then read from the *kinect2/sd/image_depth* topic and filtered to reduce noise, using the same process described by section 4.4.1.

The centre column of the processed depth image is taken and RANSAC is applied to it to estimate an equation for the ground plane relative to the Kinect's frame of reference, in the form of $y = mx + c$. θ_{pitch} and h_{kinect} are then calculated using the ground estimate and equation 4.24. This is the reverse of the steps in section 4.4.1 and can be seen in figure 4.15.

$$\begin{aligned} \theta_{pitch} &= \tan^{-1}(m) \\ h_{kinect} &= \cos(\theta_{pitch}) \times c \end{aligned} \tag{4.24}$$

The mounting variables (x_{mount}, y_{mount} and θ_{kinect}) are then calculated from θ_{pitch} and h_{kinect}, as shown in figure 4.23.

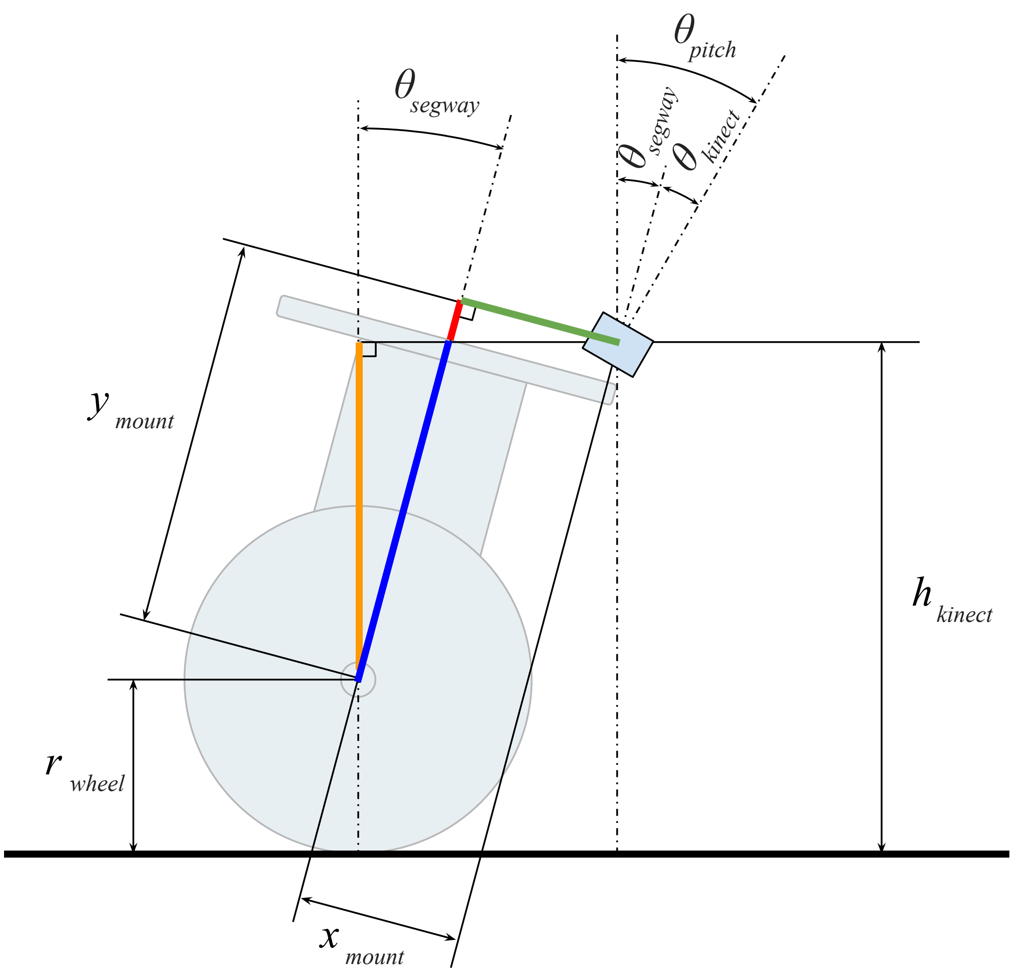


Figure 4.23: Kinect auto calibration algorithm variables.

There is not enough information to back calculate both x_{mount} and y_{mount}, but x_{mount} can be manually measured with sufficient accuracy for this application. This is because it can be easily measured along the top of the Segway's mounting plate, unlike y_{mount} and θ_{kinect} which are both difficult to measure accurately. θ_{kinect} is calculated with equation 4.25

$$\theta_{kinect} = \theta_{pitch} - \theta_{segway} \tag{4.25}$$

From figure 4.23, it can be seen that y_{mount} is equal to the sum of the blue and red lines. The red portion, y_1, can be calculated from x_{mount} and θ_{segway} using equation 4.26. The blue portion, y_2, can be calculated from h_{kinect}, r_{wheel} and θ_{segway} using equation 4.27.

$$\begin{aligned} y_1 &= \tan(\theta_{segway}) \times x_{mount} \qquad (4.26)\\ y_2 &= \frac{h_{kinect} - r_{wheel}}{\cos(\theta_{segway})} \qquad (4.27)\\ y_{mount} &= y_1 + y_2 \end{aligned}$$

Now that the mounting variables y_{mount} and θ_{kinect} are known for this particular Kinect tilt position, they are stored and the next position is requested. Once all the tilt positions have been calibrated, RANSAC is used to find the line of best fit for both the y_{mount} and θ_{kinect} sets of data. The equations for the lines of best fit are saved to the calibration file. This file is loaded by the *kinect_pipeline* node when it is launched.

4.4.3 Human Detection

One of the key objectives of this project is to enhance MARVIN's ability to interact with humans, see section 1.2. To enable this, MARVIN must be capable of detecting and tracking humans. This is achieved by processing the Kinect's depth images, as shown in figure 4.24.

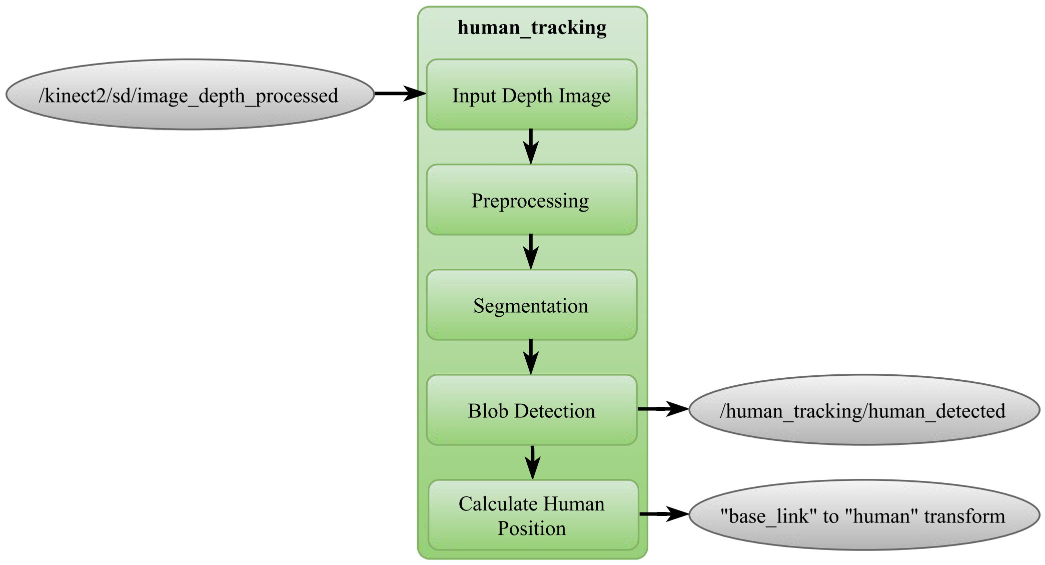
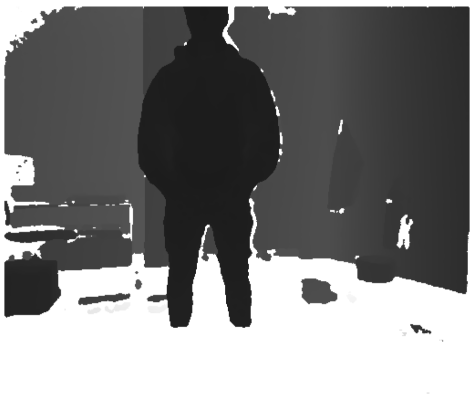

human_tracking
/kinect2/sd/image_depth_processed
Input Depth Image
Preprocessing
Segmentation
Blob Detection
/human_tracking/human_detected
Calculate Human Position
"base_link" to "human" transform

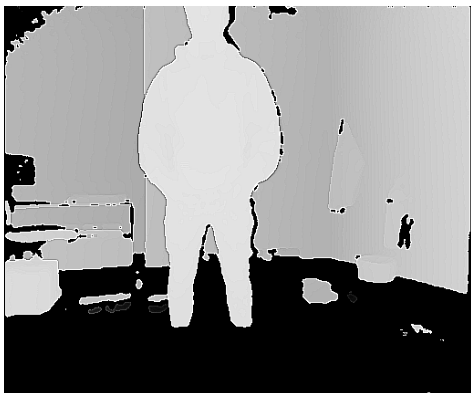
Figure 4.24: Block diagram of the human detection algorithm.

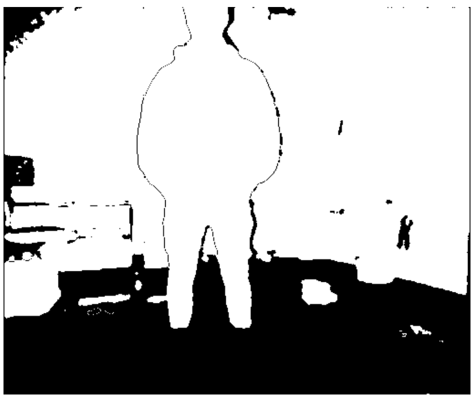
To determine if and where humans exist inside the depth image, it undergoes a number of processing steps. The pixels associated with the ground plane are first filtered out, see section 4.4.1. The image is inverted and sharpened to enhance the edges for the segmentation stage. The output of this stage is shown in figure 4.25b.

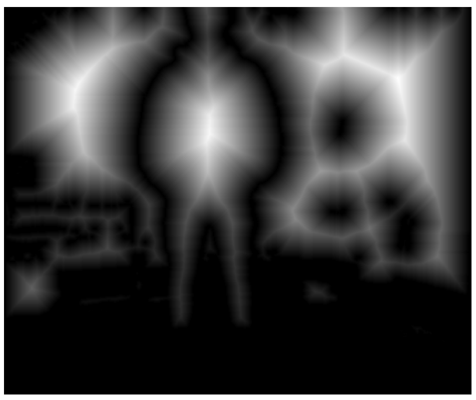
Segmentation separates the foreground objects so they don't touch each other, which is required for the blob detection stage. The image is thresholded so that only the background pixels remain. These background pixels are the pixels that have been removed from the image by the *kinect_pipeline*. This is stored as a separate binary image, shown in figure 4.25c. A distance transform is applied to the binary image resulting in figure 4.25d. This image is thresholded once again, resulting in figure 4.25e. This has effectively expanded the background regions of the image. The original input depth image is multiplied by the expanded background binary image, resulting in figure 4.25f. This resulting image has been segmented, with large separations between the foreground objects.

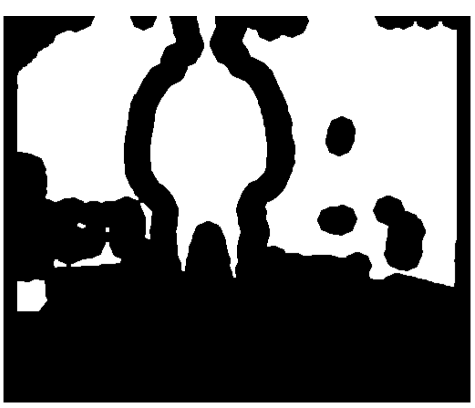
The *OpenCV* blob detection algorithm [72] is applied to the segmented image. The parameters of the blob detector have been set to look for human-like blobs. This includes; blobs that have an area within a particular range, are less than $2\ m$ away and have an inertia less than 0.3 (where an inertia of 1.0 is a circle and 0.0 is a straight line). The blobs that are considered human by this algorithm are shown as red circles in figure 4.25f. The value from the centre pixel of detected human blob is converted to a range and bearing using the same technique for converting the depth image into a *laser_scan* message, as described in section 4.4.1. The range and bearing are published as a transform from MARVIN's origin to the centre of the detected human.


(a) Input


(b) Inverted and Sharpened


(c) Binary


(d) Distance Transform


(e) Threshold

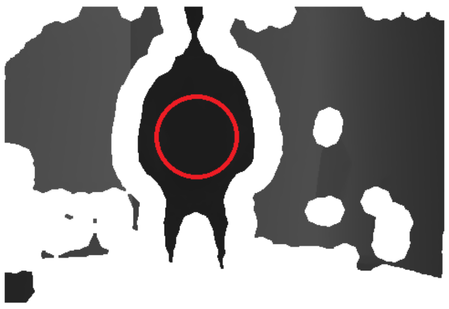

(f) Blob Detection

Figure 4.25: Human Detection Process

This section shows a viable technique for human detection from still frames using only 2D image processing techniques. It has a relatively low computation overhead resulting in good real time performance. However, there are a number of areas where this method could be extended. This technique is reliable at detecting a single human, but can struggle when multiple humans are in frame. The algorithm currently handles this problem by only returning the position of the closest human blob. It also has a relatively high false positive rate when looking at vertical surfaces that result in vaguely human shaped blobs. The multiple human issue can be resolved by tracking the humans, rather than only performing frame by frame detection. This is typically achived using some form of Kalman filter, such as the uncentred Kalman filter approach described by [68].

The full code for the *human_detection* node can be found in Appendix C.

4.5 Summary

This chapter discussed the algorithms used to process the sensor data, accounting for errors introduced by the Segway's dynamic pitch and converting it to a format the navigation stack can use. It covered the interfaces between the sensor's hardware and ROS, the steps taken to process the laser rangefinder and ultrasonic network, the novel approach used to remove the ground and ceiling planes from the Kinect's dept images and the process developed to detect humans.

The performance of the Kinect pipeline is evaluated by measuring its frame rate as additional layers of processing are applied. This is shown in table 4.2.

Table 4.2: Kinect Pipeline Performance in Frames per Second (FPS)

Processing	**Ave.**	**Min**	**Max**	**Std. Dev.**
No Filtering	30.00	23.72	40.92	1.453
Depth Removal	30.00	21.74	46.50	3.020
Depth Removal & Human Detection	30.00	19.13	58.23	3.617

From this table it can be seen that the additional processing layers do not impact the average performance of the Kinect; however, it can increase the variation of the frame rate. If the average frame rate drops, it would suggest that the Kinect pipeline requires more processing power than is available. This demonstrates that the code is sufficiently transparent and has no negative affect on the performance of MARVIN's other systems.

This processed data is used by the navigation stack for localisation and obstacle detection (as discussed in Chapter 5) and for human-robot interaction (detailed in Chapter 7).

Chapter 5

Navigation Stack

As discussed in section 1.2, MARVIN must be capable of autonomously operating in an environment designed for, and shared with, humans. To achieve this, a robust navigation stack is required. Rather than redeveloping existing solutions, the ROS *navigation* package is implemented. This is a "2D navigation stack that takes information from odometry, sensor streams and a goal pose, and outputs safe velocity commands that are sent to a mobile base" [73]. Safe velocity commands are defined as commands that will not result in the robot colliding with an obstacle. This *navigation* package has been shown to work effectively (for example) in Willow Garage's paper [25]. Figure 5.1 shows how the *navigation* package is implemented in MARVIN.

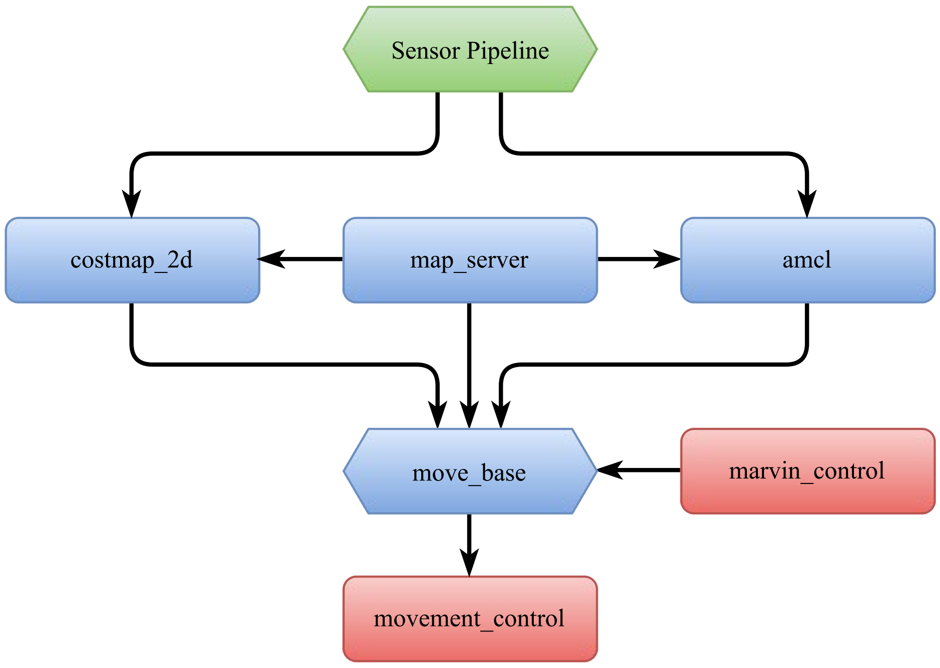


Figure 5.1: Block diagram of the Navigation Stack.

The navigation stack must be capable of three major tasks; Localisation, Obstacle Detection and Motion Planning. Localisation allows MARVIN to estimate where it is in a known map. This is achieved by the *amcl* node, explained in section 5.1. Obstacle detection informs MARVIN about the obstacles immediately around MARVIN, allowing it to navigate without colliding with them. This is achieved by the *costmap_2d* node, explained in section 5.2. Motion planning calculates a path from the position estimate to a goal location taking into account any detected obstacles and the known map. This is achieved by the *move_base* group of nodes, explained in section 5.3. The generation of the known map is achieved using simultaneous localisation and mapping (SLAM) techniques and is supplied to the other nodes using the *map_server* node, explained in section 5.4.

5.1 Localisation

MARVIN needs to know its current pose (position and rotation) to allow it to navigate. The Segway measures the amount each of its wheels rotates over time. From these measurements we can estimate the change in MARVIN's pose relative to its starting pose. This is known as the robot's odometry. Odometry can be used to estimate MARVIN's pose, but it prone to errors. These errors can be caused by the wheel radius changing (due to flat tires), by the wheels slipping or by stochastic errors in the wheel rotation measurements. Over time these errors accumulate providing a less certain estimate. This is known as odometry drift. Because of this, MARVIN needs additional information to correctly estimate its pose. MARVIN's operating environment is confined to known areas, as discussed in section 1.1.1, so a pre-built map can be used. We can compare what MARVIN is currently sensing to the map to narrow down the list of possible poses. If we then combine this list of possible poses with the odometry and any previous pose estimates, a much more certain pose estimate can be made. This process is known as probabilistic localisation.

A well known and effective probabilistic localisation method is Monte Carlo Localisation (MCL). MCL uses a particle filter to localise, where the particles represent pose estimates. It works by incrementally updating the weights of each particle based on the sensor and odometry data, then redistributing the particle pose estimates based on those weights. Over time the distribution of the particles should converge on the robot's actual pose. The returned pose estimate from MCL is the pose at the peak of the probability density function (PDF) created by the particles. MCL becomes more effective (more likely to produce a correct pose estimate in fewer steps) the larger the number of particles, however this also makes it more computationally intensive. MCL's effectiveness is therefore inversely proportional to its computational efficiency. One method to reduce this trade-off is to adaptively change the number of particles. For example, when the pose estimate is less certain it may be desirable to have more particles to increase the chance of generating a more accurate estimate. When the pose is more certain, the number of particles can be reduced to improve the computational efficiency. This is known as Adaptive Monte Carlo Localisation (AMCL). AMCL is implemented in MARVIN using the ROS *amcl* package [74] which implements the approach as described by Dieter Fox in [75]. Figure 5.2 shows how the *amcl* package is integrated in MARVIN.

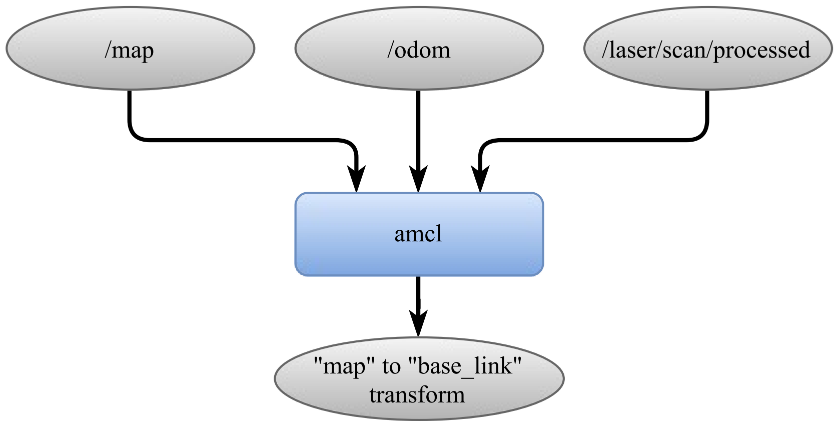


Figure 5.2: Block diagram of the *amcl* node.

The *amcl* uses the */map* topic as the known map, the */odom* topic as odometry and the */laser/scan/processed* topic as the sensor update topic. The laser scanner topic is the only sensor topic the node is subscribed to as it is the same sensor used to generate the map, and therefore should produce the most consistent results. The primary output of the *amcl* node is a transform from the *map* frame to the *base_link* frame (where the *base_link* frame is MARVIN's origin, as explained in Chapter 4). This transform is used to link the global (map) coordinates to the local (robot) coordinates.

5.2 Obstacle Detection and Sensor Fusion

As MARVIN operates in an indoor human environment, it is imperative that it has a robust method for detecting and avoiding obstacles. As explained in section 3.2.1, MARVIN utilises four sensing systems to detect obstacles; the Kinect, the LiDAR, whisker sensors and the ultrasonic network. The sensor data from these four sources have been processed into *LaserScan* messages, as explained in Chapter 4, but all have different ranges and FOVs. This sensor data must be fused so that the motion planner knows about all the sensor sources, while taking into account their differing parameters. This is typically achieved using an occupancy grid. An occupancy grid is essentially a local map centred at the robot's origin. The cells of the map are either classed as occupied or free space and are set by converting the sensor data from polar coordinates into the local Cartesian coordinates. This is achieved in MARVIN using the ROS *costmap_2d* package [4].

The *costmap_2d* package extends the simple occupancy grid in a number of ways. The grid cells can be in one of three states; occupied, free or unknown. This lets the motion planner treat unknown space differently from free space, allowing for safer navigation. The cells in the grid are not only set by the sensors, they are also cleared by the sensors. This means that obstacles are only removed from the occupancy grid if it has been observed that they are no longer present. This is further explained in section 5.2.2. The obstacles in the occupancy grid are also inflated to take into account the robot's footprint, allowing the planner to safely navigate the shortest path while ensuring that the robot's edges won't collide with the known obstacles. This is further explained in section 5.2.4. Figure 5.3 shows how the *costmap_2d* package is integrated with MARVIN.

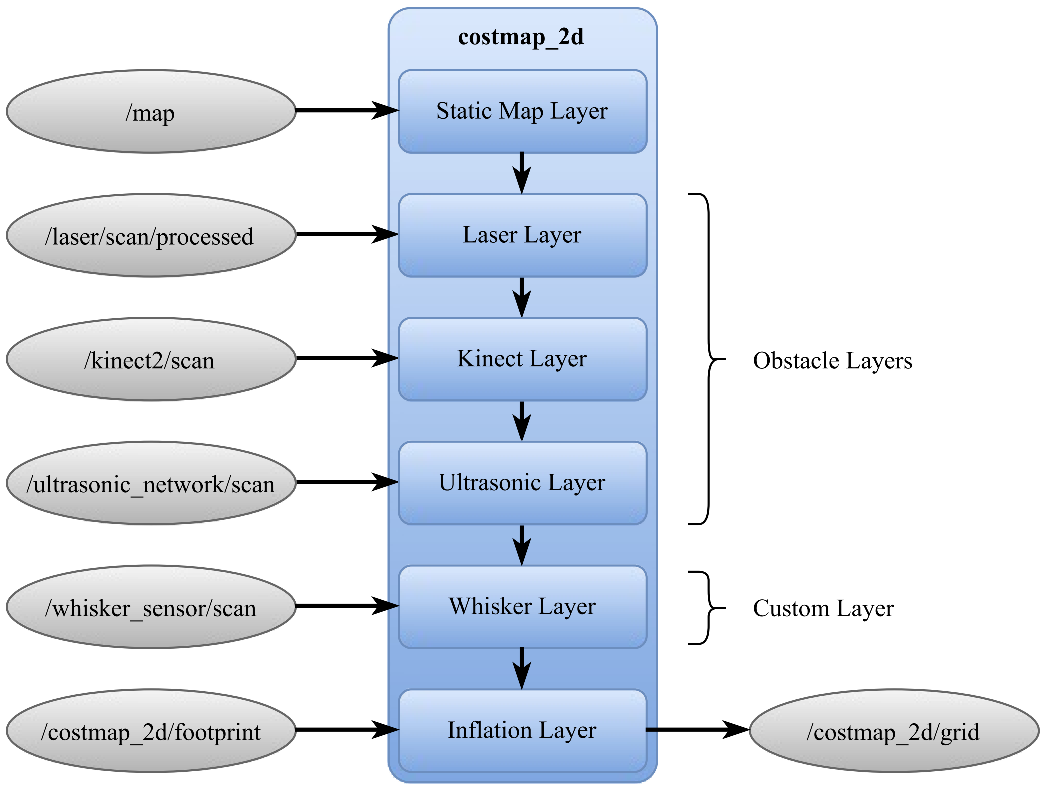


Figure 5.3: Block diagram of the *costmap_2d* node.

5.2.1 Static Map Layer

The static map layer incorporates the known map into the occupancy grid. If there are no observations in a region of the occupancy grid, the static map is used as the observation source rather than the sensors. This allows the planner to predict what is in unknown regions of the map before it reaches them, using prior knowledge. This results in safer and more reliable navigation.

5.2.2 Obstacle Layers

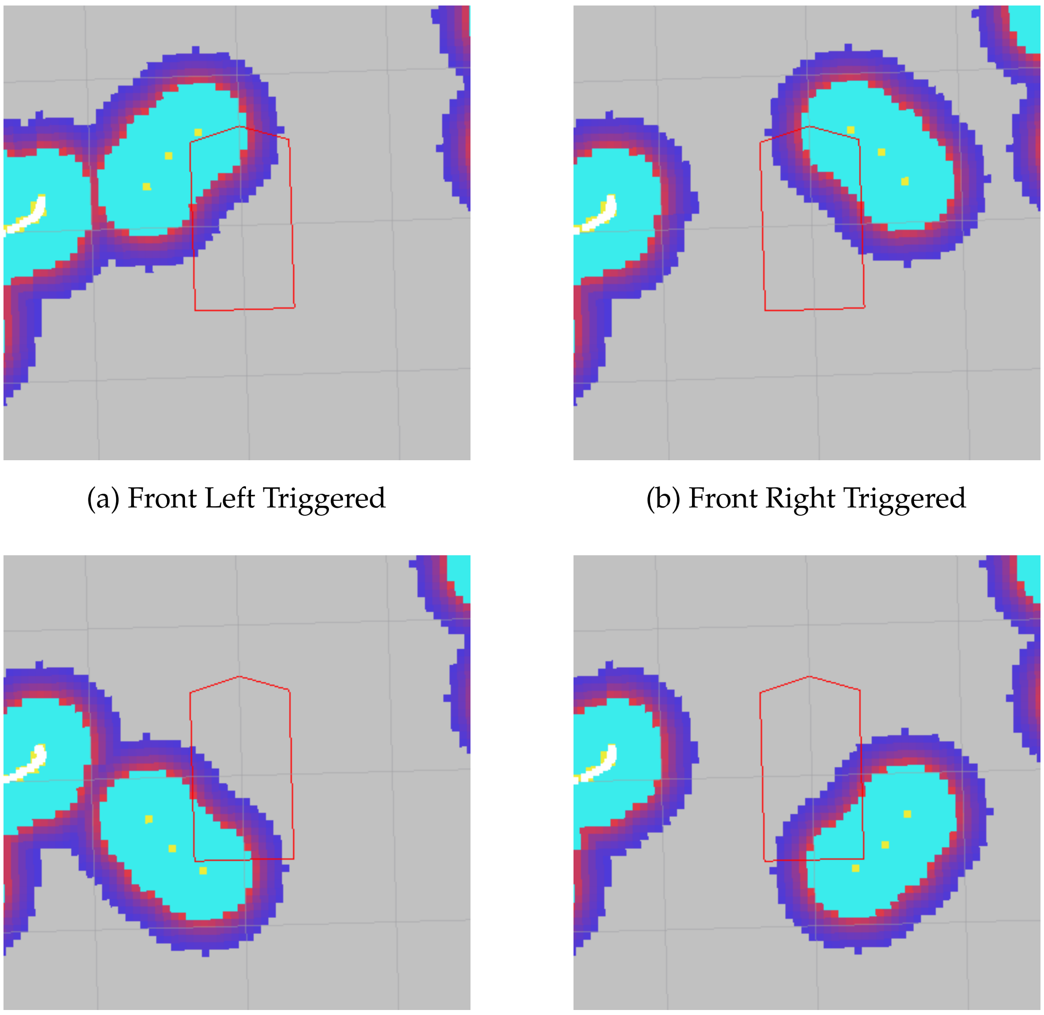
The obstacle layers handle the sensor sources and the setting and clearing of the occupancy cells. Cells are set by converting the sensors' polar measurements into the grid's Cartesian coordinates. Cells are cleared by ray tracing along the sensors' angle increments. If a sensor detects an obstacle behind and in-line with an existing obstacle, it is assumed that the existing obstacle no longer exists (because the senor saw through it). A separate obstacle layer is used for each of the *LaserScan* based sensors; the LiDAR, Kinect and ultrasonic network. This is because each source can sense different obstacles. The LiDAR has a much larger horizontal FOV than the Kinect, but the Kinect can sense obstacles above and below the LiDAR's planar sensing region. The ultrasonic network can sense in much lower detail than the other two sensors, but it can sense glass. If all the sensor sources are integrated into one obstacle layer they clear each other's measurements. This leads to the motion planner incorrectly believing that obstacles no longer exist, which can cause unsafe paths to be generated.

This is best explained by imagining MARVIN moving beside a table with legs. The LiDAR can only sense the legs but not the table's surface. The Kinect can sense the whole table. While the table is within the Kinect's limited horizontal FOV the moiton planner knows it cannot travel through the table. However, as MARVIN moves along the table, the rear of the table's surface is no longer in the Kinect's FOV. As the LiDAR can detect obstacles behind the table, it clears the Kinect's measurements, resulting in the planner believing that there is now a free path through the table. This causes MARVIN to turn towards the table, putting it back in the Kinect's FOV. Now MARVIN can sense the table's surface again and turns away from the table. This means the table is no longer in the Kinect's FOV and the process repeats indefinitely. This problem is avoided by having a separate obstacle layer for each of the sensor sources. This means that each sensor only sets and clears its own measurements.

5.2.3 Whisker Layer

In addition to the reactive control discussed in section 6.1, the high level control also needs to know if the whisker sensors have been triggered. This means the whisker sensors need to be integrated in the occupancy grid.

A standard obstacle layer cannot be used for the whisker sensors. This is because the whisker sensors only produce four data points, which makes them unreliable at clearing their obstacles through ray tracing. This is caused by the resolution of the occupancy grid; if MARVIN rotates slightly the ray traced will be slightly off the previous measurement which prevents it from being cleared. To avoid this issue a custom layer is developed. The custom layer checks the */whisker_sensor/scan* topic and sets the cells at each corner of MARVIN's footprint to occupied if the corresponding whisker sensor is triggered. When a new whisker sensor message is received the entire whisker layer is cleared and then the triggered corners are re-applied. An example of this is shown in figure 5.4.


(a) Front Left Triggered (b) Front Right Triggered

(c) Back Left Triggered (d) Back Right Triggered

Figure 5.4: Whisker layer example.

5.2.4 Inflation Layer

The inflation layer inflates the regions around the occupied space in the grid to take into account the physical size (footprint) of MARVIN. This process converts the three state occupancy grid into an 8-bit costmap. Each cell in the costmap is given a value between 0 and 254, depending on how likely a collision is if the origin of the robot was in that cell. This can be seen in figure 5.5.

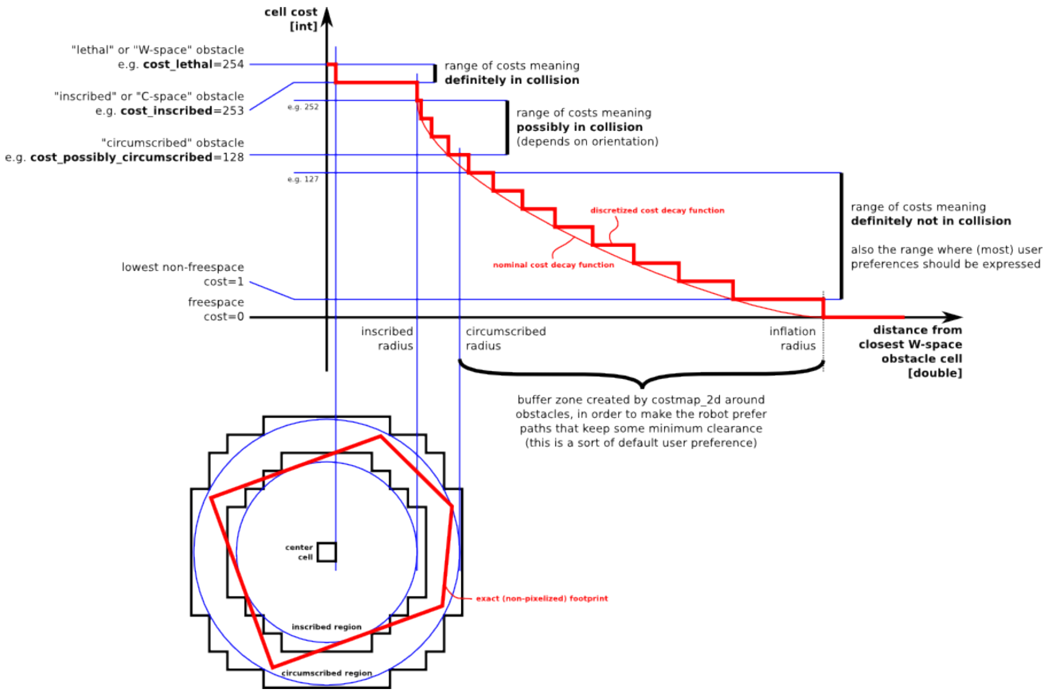

cell cost [int]
"lethal" or "W-space" obstacle e.g. cost_lethal=254
"inscribed" or "C-space" obstacle e.g. cost_inscribed=253
"circumscribed" obstacle e.g. cost_possibly_circumscribed=128
lowest non-freespace cost=1
freespace cost=0
e.g. 252
e.g. 127
range of costs meaning definitely in collision
range of costs meaning possibly in collision (depends on orientation)
range of costs meaning definitely not in collision
also the range where (most) user preferences should be expressed
discretized cost decay function
nominal cost decay function
inscribed radius
circumscribed radius
inflation radius
distance from closest W-space obstacle cell [double]
buffer zone created by costmap_2d around obstacles, in order to make the robot prefer paths that keep some minimum clearance (this is a sort of default user preference)
center cell
inscribed region
circumscribed region
exact (non-pixelized) footprint

Figure 5.5: Costmap inflation layer [4].

From [4], the cell values are split into five specific symbols. A value of 254 is ”lethal”, meaning that the robot would be inside the obstacle. Values from 253 to 128 are ”inscribed”, meaning that these cells will result in a collision as they are within the robot’s inscribed radius of an obstacle. Values from 127 to 1 are ”possibly circumscribed”, meaning that these cells might result in a collision, depending on the orientation of the robot. A value of 0 is ”freespace”, meaning that this cell will not result in a collision. Any ”unknown” cells are treated as ”freespace” until they are observed. The motion planner can now use this 8-bit costmap to generate safe paths by optimising for the lowest cost path.

5.2.5 Costmap Example

Figure 5.6 shows how the static map, obstacle, whisker and inflation layers work together to produce a functional costmap. The yellow points are obstacles and the blue regions are the inflated regions. In figure 5.6a a human is to the front right of MARVIN. In figure 5.6b the human moves left to be in front of MARVIN. In figure 5.6c the human moves towards MARVIN. The cells behind the human are not cleared because the human is blocking the sensor's view of those cells, preventing them from being cleared. As the human moves to the left in figure 5.6d, those cells are cleared.

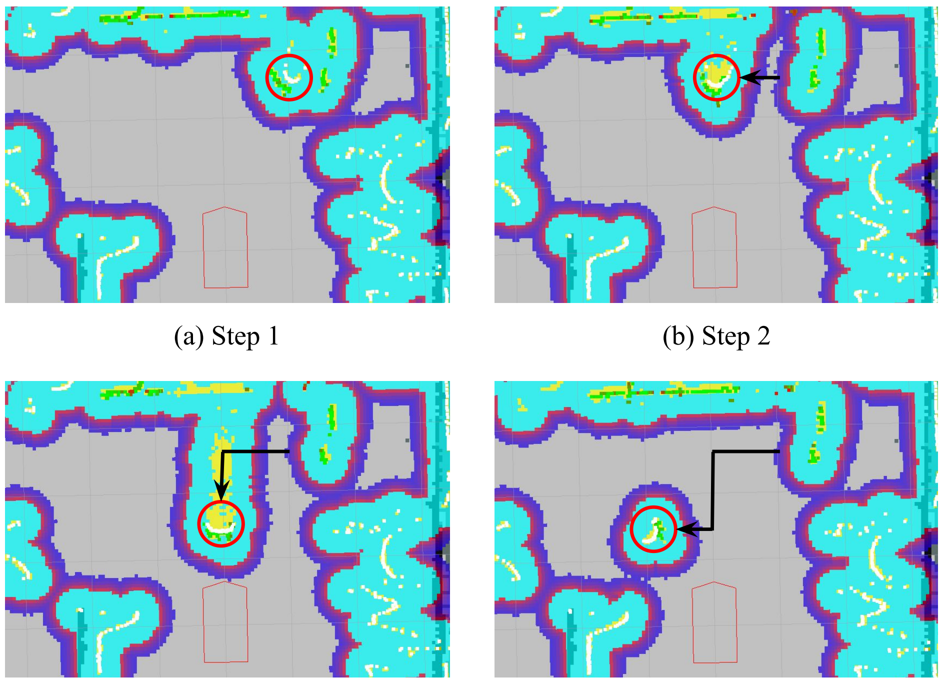

(a) Step 1
(b) Step 2
(c) Step 3
(d) Step 4

Figure 5.6: Dynamic costmap example over four time steps. The red circle represents the human's current position and the black arrow shows their travelled path.

5.3 Motion Planning

Motion planning is handled by the *move_base* package [76], which takes commands from the control stack and interfaces with separate global and local planners. The global planner used is the *global_planner* package [5], which generates a high level path from MARVIN's current pose to a goal pose. This is further explained in section 5.3.1. The local planner used is the *base_loacal_planner* package [77], which generates a low level obstacle avoidance path. This further explained is section 5.3.2. Figure 5.7 shows how these packages are integrated in MARVIN.

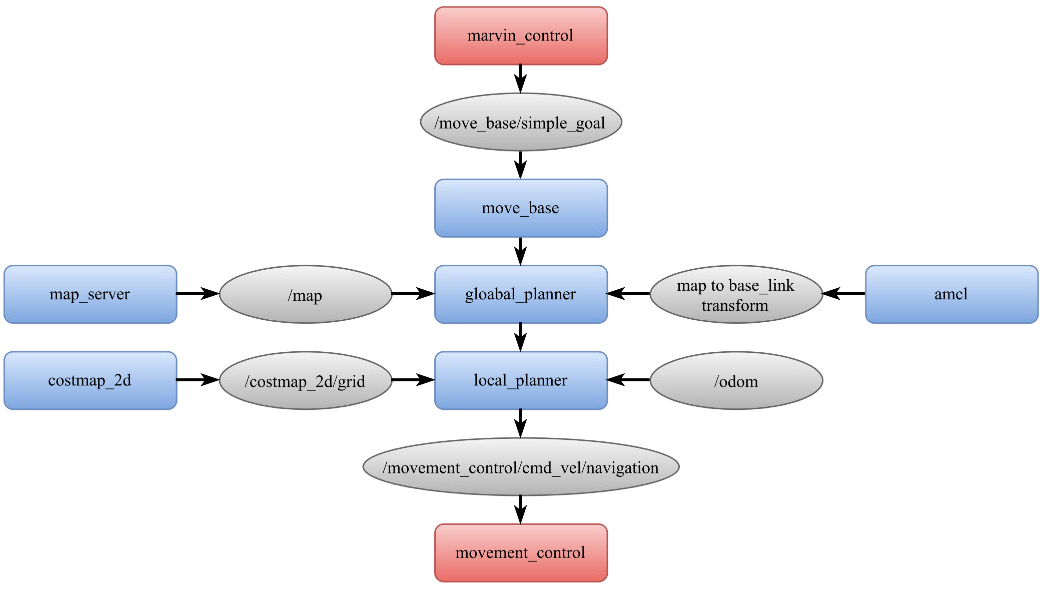


Figure 5.7: Block diagram of the *move_base* node.

5.3.1 Global Planner

The global planner gets a goal pose from the *move_base* node and MARVIN's pose from the *amcl* node. It then uses a search algorithm to find a path from

MARVIN's location to the goal inside of the static map. The *global_planner* package has two search algorithms available; A* search and Dijkstra's search. A* can find a path in fewer computation steps than Dijkstra's, but by doing so typically produces less optimised paths. This is shown by figure 5.8. Due to the Kinect's processing requirements, the NUC contains high performance components (as explained in section 3.2.2). This makes the performance loss of Dijkstra's algorithm acceptable for the production of more optimised paths. As such, Dijkstra's is used over A*.

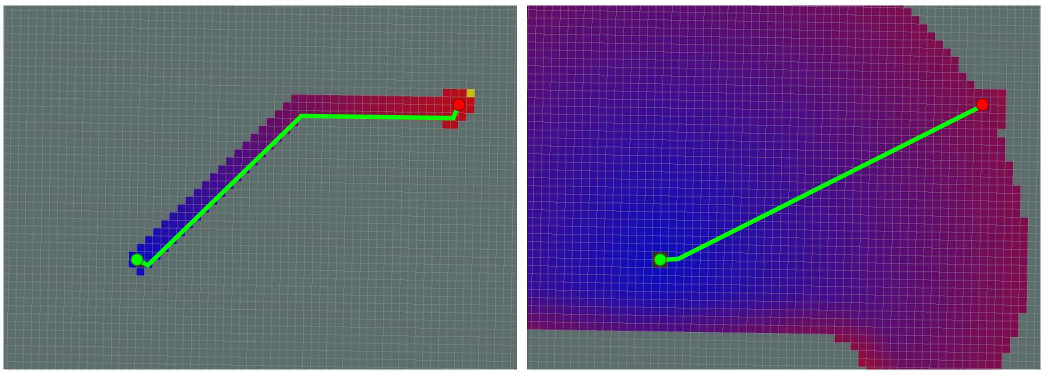

(a) A* search (b) Dijkstra's

Figure 5.8: Comparison of the *global_planner* search algorithm implementations [5].

5.3.2 Local Planner

The *base_local_planner* is built from the ideas explained in [78, 34, 79]. It is a hybrid controller, meaning it combines the deliberative plan from the global planner with the more reactive obstacle information from the costmap. It allows the use of two different approaches; the trajectory roll-out algorithm [78], or the dynamic window approach (DWA) [34]. Both approaches follow the same basic idea. First they sample the robot's current pose and velocity (both angular and linear). Then they simulate a number of trajectories (with slight variations in angle and speed) from the robot's current pose, taking into account its velocity. The simulated trajectories are then evaluated against their distance from the global path, distance from the goal, speed of the trajectory and eliminating any invalid paths that would cause collisions. The trajectory with the highest score is then sent to the robot's mobility platform and the process is repeated.

Trajectory roll-out simulates the predicted trajectories for multiple steps into the future, where as DWA only simulates one step into the future. This makes the DWA more computationally efficient than trajectory roll-out, but trajectory roll-out can produce better local plans, especially when the robot is moving with low acceleration. However, according to [77], both approaches produce similar results in practice, so MARVIN makes use of the DWA for the reduced computational cost.

5.4 Mapping

As explained in sections 5.2.1 and 5.3.1 a static map is required for both the costmap and the global planner. To produce a static map, SLAM techniques are used. The *gmapping* package [80] contains a ROS wrapper for OpenSLAM's [81] GMapping algorithm, which is described by the University of Freiburg's papers; [82] and [83]. GMapping uses a "highly efficient Rao-Blackwellized particle filer to learn grid maps from laser range data". Figure 5.9 shows the method used to generate the map of the environment.

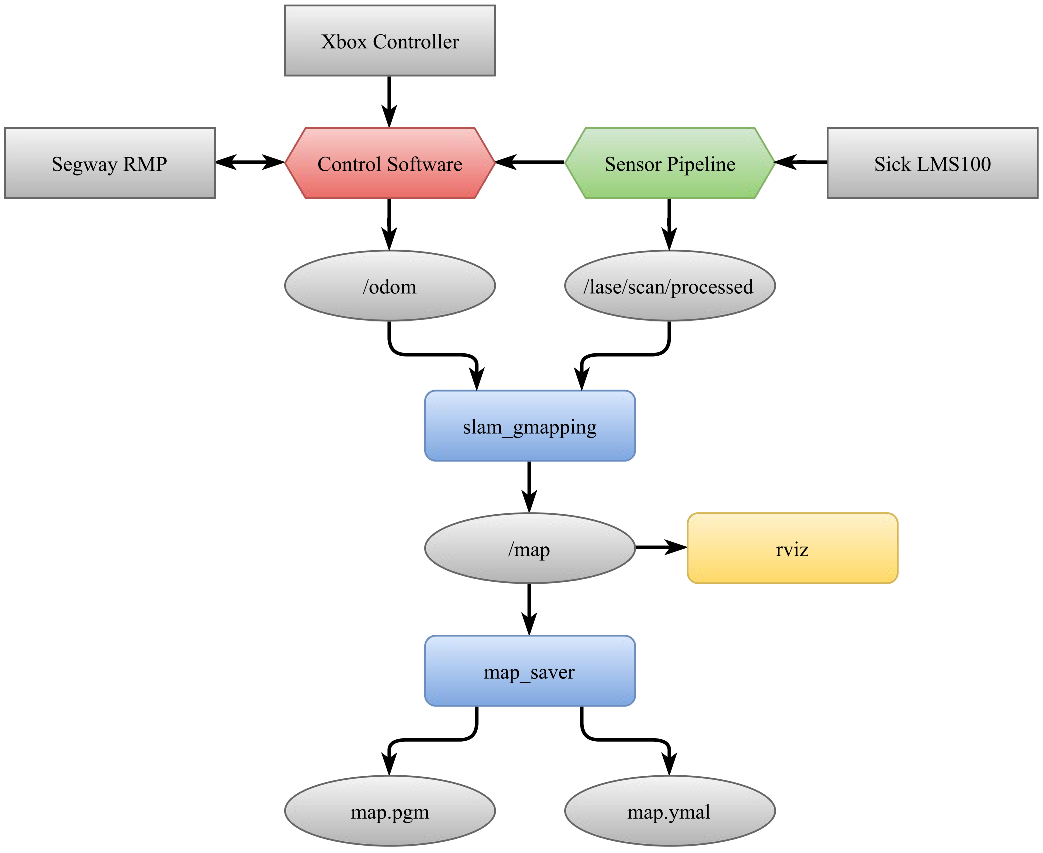


Figure 5.9: Map generation method.

MARVIN is manually controlled by the XBox controller through the control stack, discussed in section 6.3. The LiDAR's data is processed as explained by section 4.3 and passed to the *slam_gmapping* node along with the Segway's odometry data. This is used to produce the map which is published to the */map* topic. This topic can then be subscribed to by *rviz* [71] so that the user can monitor the generation of the map. Once the map is complete, the *map_saver* node from the *map_server* package [84] can be used to save the generated map to disk.

During run time the *map_server* node loads the saved map and publishes it to the */map* topic. The *amcl*, *costmsp_2d* and *global_planner* nodes can then subscribe to the */map* topic to make use of the known map. This is shown in figure 5.10.

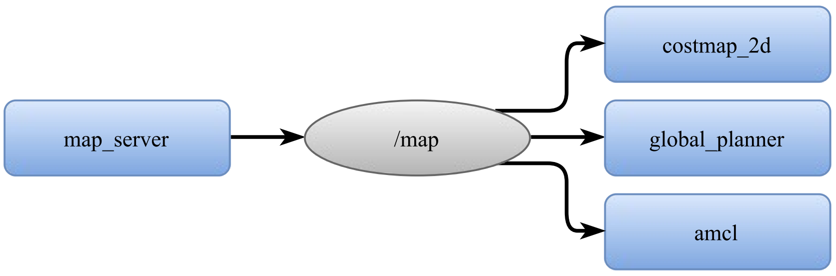
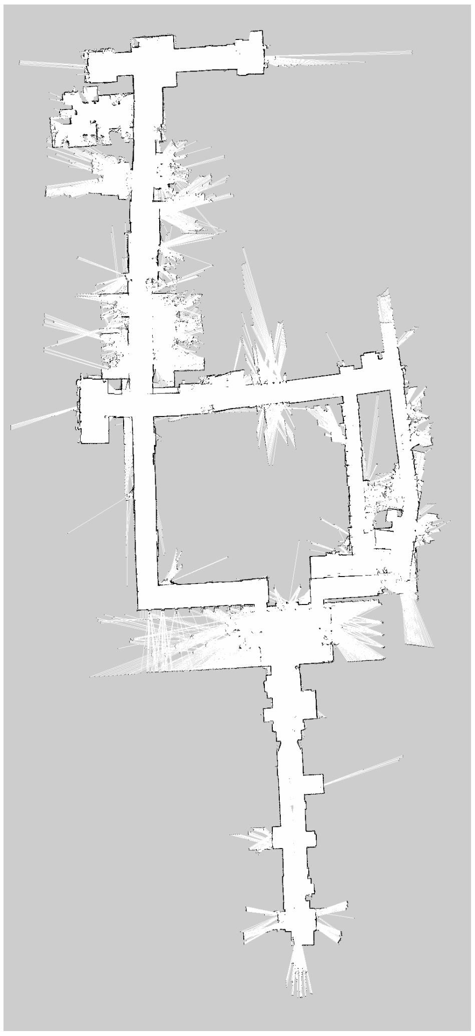


Figure 5.10: Block diagram of the map server node.

An example of a generated map is shown in figure 5.11a. This is a map of the second floor corridors of the Cotton and Alan MacDiarmid buildings of VUW's Kelburn campus. This map was cleaned up manually using the open source photo editing program, Gimp, shown in figure 5.11b. This manual processing step is unnecessary for the navigation stack, but was performed to better visualise the produced map.

From figure 5.11 it can be seen that the GMapping algorithm can produce detailed maps, with good local accuracy. However, there are some areas where sections of the map are stitched together at incorrect angles. This problem would likely be improved by performing more loops around the affected areas. However, as the localisation algorithm compares the local details to the sensor data, localisation is still effective in this map. Also, as long as the goals are inside the generated map, the global planner has no issue generating paths in the slightly distorted map.


(a) Raw Map

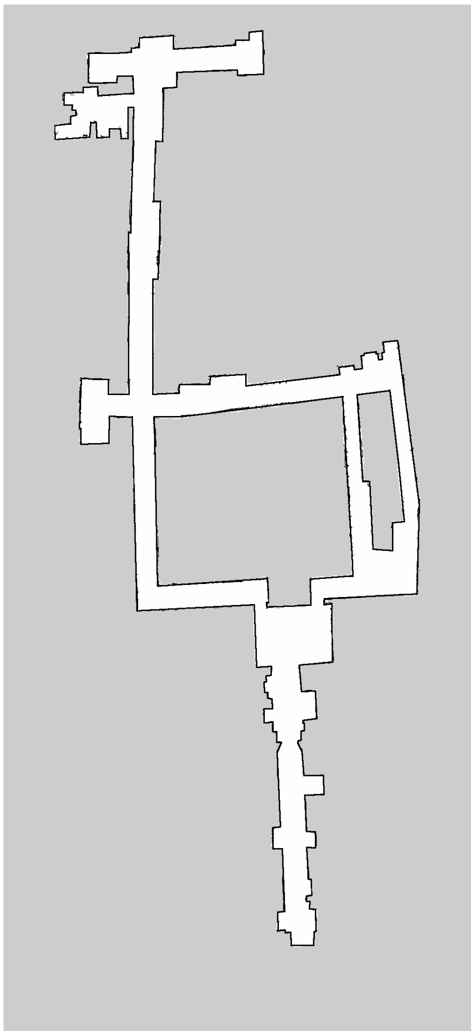

(b) Processed Map

Figure 5.11: Map of level 2 AM and CO corridors, generated with GMapping.

5.5 Summary

This chapter discussed MARVIN's navigation stack, including the algorithms and techniques used for localisation, motion planning and mapping. This process takes the sensor data and generates safe velocity commands for the Segway RMP. These commands attempt to move MARVIN towards a high level goal while preventing any collisions with the detected obstacles. The process also attempts to maintain an accurate global position estimate thought the AMCL algorithm. The sucessful implementation of these systems meet the autonomous navigation objectives outlined in section 1.2, including objectives: 1c (sensor fusion), 1d (localisation) and 1e (motion planning). The navigation stack is integrated with the rest of MARVIN's systems through a multilayer hierarchical control system, as discussed in Chapter 6.

Chapter 6

Control Architecture

This chapter explains the design and implementation of MARVIN's control architecture. Control software is required to manage MARVIN's subsystems, including the navigation stack explained in Chapter 5, the interaction stack explained in Chapter 7 and the Segaway and torso interfaces explained in sections 6.1 and 6.2. This is important to regulate the behaviour of the system as a whole, ensuring safe and robust operation.

MARVIN's control architecture has five layers; high level control, user control, mid level control, low level control and the hardware interface. This control hierarchy is shown in figure 6.1.

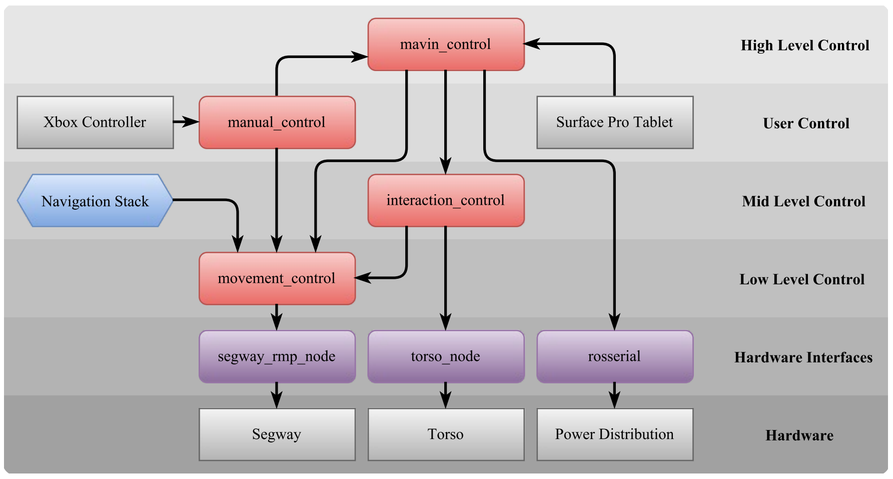


Figure 6.1: Block diagram of the control software hierarchy.

The high level control layer consists of the central control node *marvin_-control*, which manages and monitors all the lower control levels. This is further explained in section 6.4. The user control level contains the user interfaces, including the Microsoft Surface and the Xbox controller. The XBox controller is interfaced with the *manual_control* node, explained in section 6.3. The use of the Surface is further explained in Chapter 7. The user control layer can make control requests to the other control layers, but does not have direct control over them. The mid level control layer contains the intermediary nodes that are controlled by *marvin_control*, but also have control of the lower level layers. This includes the *interaction_-control* node (explained in section 6.2) and the navigation stack (explained in Chapter 5). The low level control layer contains the *movement_control* node, which communicates with the Segway's interface node (explained in section 6.1). The hardware interface layer contains the interfaces between the hardware and the control layers, including: the Segway's interface (*segway_rmp_node*), the torso's interface (*torso_node*) and the inherited power distribution board's interface (implemented with a *rosserial* node).

6.1 Movement Control

The movement control node has two primary tasks: combine the different movement control sources and safely command the Segway to move. As one of the main objectives of this thesis is human interaction, it is imperative that MARVIN is capable of moving safely in close proximity with humans. No matter which source the movement commands originate from, they must be processed and monitored to ensure safe operation (as defined in section 1.2). The movement control node is a low level control node that processes the mid level movement commands and then feeds them to the Segway RMP interface. The process of combining and monitoring the movement commands is shown in figure 6.2.

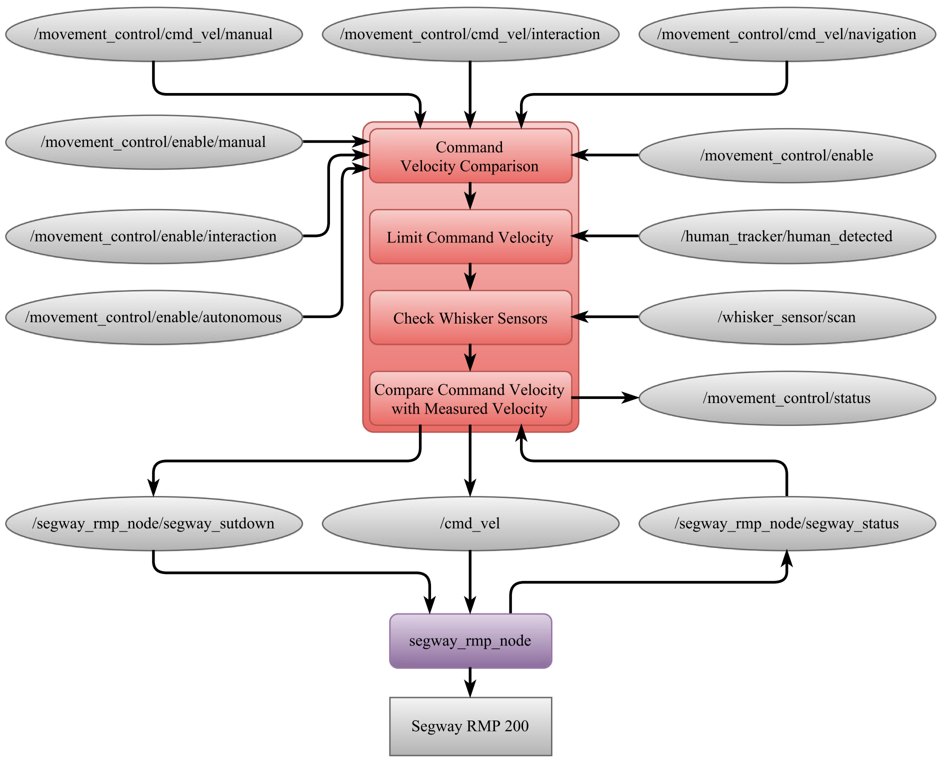

/movement_control/cmd_vel/manual
/movement_control/cmd_vel/interaction
/movement_control/cmd_vel/navigation
/movement_control/enable/manual
Command Velocity Comparison
/movement_control/enable
/movement_control/enable/interaction
Limit Command Velocity
/human_tracker/human_detected
/movement_control/enable/autonomous
Check Whisker Sensors
/whisker_sensor/scan
Compare Command Velocity with Measured Velocity
/movement_control/status
/segway_rmp_node/segway_sutdown
/cmd_vel
/segway_rmp_node/segway_status
segway_rmp_node
Segway RMP 200

Figure 6.2: Block diagram of the Movement Control node.

The first step is to combine the control sources. Each of these sources sends movement commands using the *geometry_msgs/Twist* messages from the ROS geometry messages library [85]. *geometry_msgs/Twist* messages express velocity with two 3D vectors, one for the linear component and the other for the angular component. There are three movement control sources; manual control, interaction control and the navigation stack (as illustrated on the top of figure 6.2). Manual control allows a user to operate MARVIN using an Xbox 360 controller, as explained in section 6.3. Interaction control changes the pose of MARVIN when interacting with humans, as discussed section 6.2. The navigation stack allows for the autonomous operation of MARVIN, previously explained in Chapter 5. If improperly managed, the combination of these control sources can cause the Segway to become unpredictable and dangerous. As such, only one source should be passed to the Segway at a time. To decide which source to use, they are prioritised:

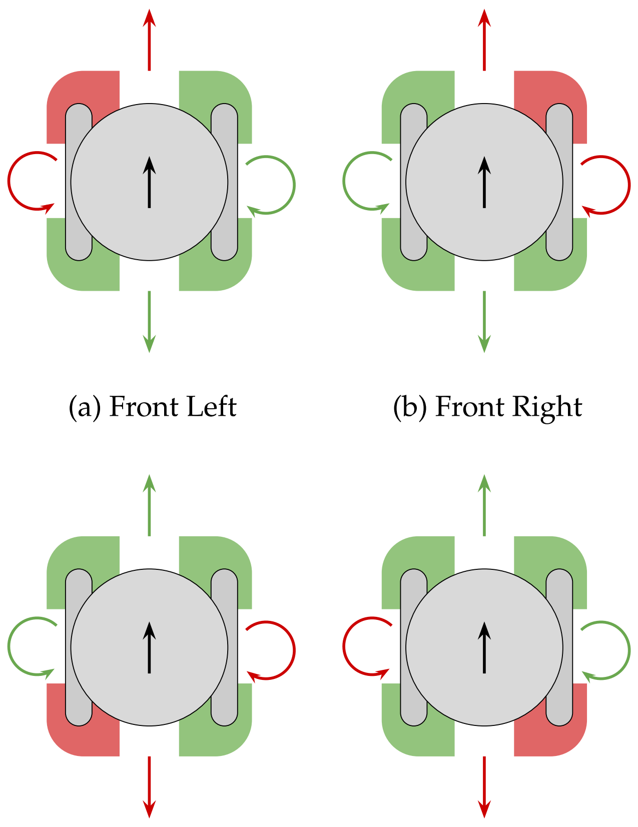
1. Manual Control
2. Interaction Control
3. Navigation Stack

Manual control has the highest priority and overrides any messages from the other two sources. If manual control is enabled, it is assumed that a trained operator is controlling MARVIN. Because of this, it is considered the most reliable command source. This is required to improve safety during testing procedures as the operator can override any actions deemed to be unsafe. The second highest priority is the interaction control, which can override commands from the navigation stack. If MARVIN is interacting with a user it should only adjust its pose to look at that user. As such, any navigation commands will be invalid and should be ignored. The lowest priority is the navigation stack, which does not override any other commands. The navigation stack should only control MARVIN if there are no manual or interaction commands.

In addition to their priorities, each control source can be enabled or disabled. This allows the *marvin_control* node to change which source to use depending on the current operating conditions (further explained in section 6.4). The entire *movement_control* node can also be disabled, which prevents any movement controls from passing to the Segway. This can be used to stop or pause MARVIN's motion. It is important that the control sources can be enabled or disabled without affecting the source itself. This allows the navigation stack to operate in the background, which is necessary for maintaining MARVIN's location estimate.

The next step is to limit the velocity of the movement command to a safe level. The maximum linear velocity is set to 1.41 ms^{-1}, as this is the average humans' comfortable walking speed [17]. This enables the user to follow behind MARVIN at a comfortable pace while being guided. The linear velocity is then further limited to 1.0 ms^{-1} if a human is detected in front of MARVIN. This reduces the stopping distance of MARVIN (further explained in section 8.1), which reduces the chance of causing harm if the human unpredictably moves into MARVIN's path. The angular velocity is limited to 1.0 rads^{-1} if no human is detected and 0.5 rads^{-1} otherwise. This helps prevent the Segway's tilt limiters from causing damage as MARVIN turns.

The whisker sensors are monitored to help prevent MARVIN from turning or driving into obstacles. There are four whisker sensors mounted on MARVIN's four outermost corners, as discussed in section 3.2.1. As these sensors are contact sensors, they are useful as reactive sensors in case any obstacles are missed by the range senors. Figure 6.3 shows how the whisker sensors' reactive control is configured. Red indicates that that linear motion or angular direction has been disabled and green indicates that they are enabled.



(a) Front Left (b) Front Right

(c) Back Left (d) Back Right

Figure 6.3: Whisker sensor reactive control.

The final step of the *movement_control* node is to monitor the output command velocity topic. This allows the *movement_control* node to enforce its monopoly on the control of the Segway. This is important to ensure the safe operation of the Segway. As previously stated, whenever multiple sources control the Segway directly it can lead to unpredicted and dangerous operation. If the *movement_control* node detects unauthorised control messages (messages that differ from its own output) the Segway is shutdown.

To allow the *marvin_control* node to monitor the *movement_control* node, it publishes *MovmentControlStatus* messages to the *movement_control/status* topic. These status messages can be seen in appendix A. The code for the *movement_control* node can be seen in appendix C.

6.2 Interaction Control

Interaction control provides an interface between the interaction stack (explained in Chapter 7) and MARVIN's actuation hardware. This abstracts the hardware interface layer, allowing future researchers to control MARVIN's pose without an in-depth knowledge of the low level systems.

This abstraction is achieved using configurable zones. The *interaction_control* node subscribes to the *base_link* to *human* transforms, provided by the *human_detection* node (explained in section 4.4.3). These transforms are used to determine if a human is inside of one of the configured zones. If this is the case, MARVIN's torso and the Segway are requested to move to match the pose set for that particular zone. Figure 6.4 shows the current zone configuration.

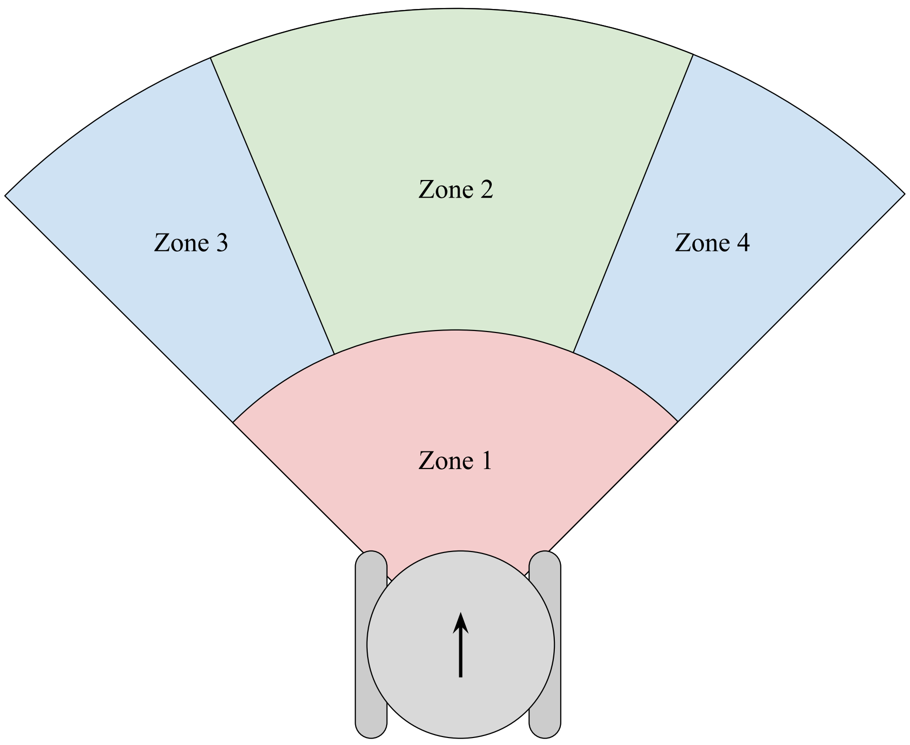


Figure 6.4: Current configuration of the interaction zones.

Any number of zones can be configured, allowing researchers to change the way MARVIN reacts to humans quickly and easily. Each zone contains range and bearing limits, a torso frame and a Segway tracking enable flag. The current zone configuration is explained in Chapter 7.

Range and bearing limits set the area of the zone. The torso frame stores the values for each of the torso's changeable elements. This includes; the left and right shoulders, torso tilt, head rise, tilt, nod and shake, as well as the RGB values for each eye. The eyes can also be set to modulate with an input audio signal. This is typically used by inputting MARVIN's synthesised voice (further explained in Chapter 7). When the detected human enters a zone, its torso frame is sent to the *torso_node* via the */torso/request* topic.

If the Segway tracking flag is enabled, the *interaction_control* requests that MARVIN faces the detected human directly.

Figure 6.5 shows how these zones are used to control the Segway and torso poses.

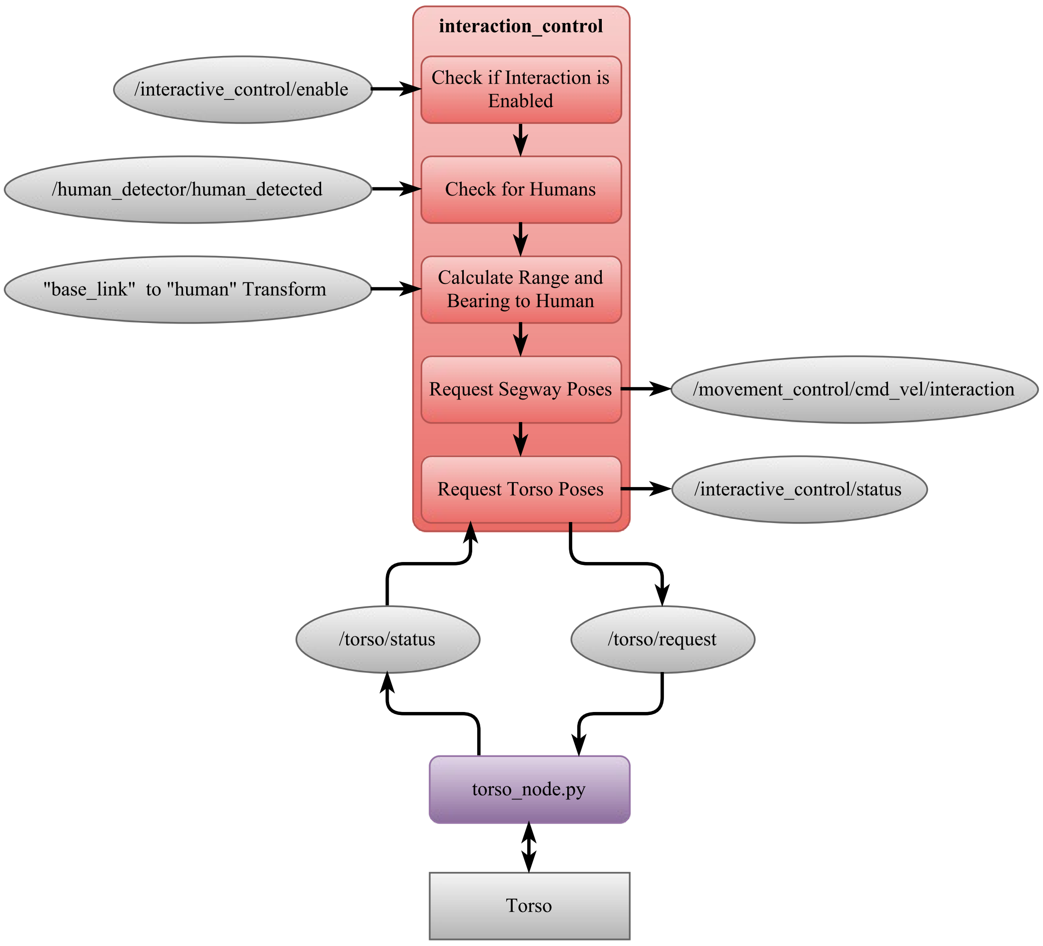

interaction_control
/interactive_control/enable
Check if Interaction is Enabled
/human_detector/human_detected
Check for Humans
"base_link" to "human" Transform
Calculate Range and Bearing to Human
Request Segway Poses
/movement_control/cmd_vel/interaction
Request Torso Poses
/interactive_control/status
/torso/status
/torso/request
torso_node.py
Torso

Figure 6.5: Block diagram of the Interaction Control node.

The first step is to check if the *interaction_control* node is enabled. This is controlled by the *marvin_control* node, see section 6.4. If the *interact_control* node is disabled, no further actions are taken. The next step is to check if a human has been detected. If a human has been detected the range and bearing to them is calculated, otherwise no further actions are taken.

The range and bearing from MARVIN's origin to the detected human is calculated from the *base_link* to *human* transform using equation 6.1, where r_{human} is the range to the human, b_{human} is the bearing to the human, t_x is the x value of the transform's origin and t_y is the y value of the transform's origin.

$$\begin{aligned} r_{human} &= \sqrt{{t_x}^2 + {t_y}^2} \\ b_{human} &= \tan^{-1}\left(\frac{t_y}{t_x}\right) \end{aligned} \tag{6.1}$$

The range and bearing is used to determine which zone the detected human is in. This is achieved by comparing them against each of the configured zones' limits until a match is found. If no match is found it is assumed the human is outside the interaction area and no further actions are taken.

If Segway tracking is enabled, a movement request is sent to the *movement_-control* node via the */movement_control/cmd_vel/interaction* topic. This request has a linear velocity of $0.0\ m^{-1}$ and an angular velocity of b_{human}, which turns MARVIN to face the human.

If the zone has a configured torso frame, the relevant commands are sent to the *torso_node* via the */torso/request* topic. The *torso_node* is the inherited software used to interface the torso's control boards with ROS.

The *interaction_control* node outputs status messages that are used for debugging purposes and allows the *marvin_control* node to monitor the interaction subsystems. The status messages can be seen in appendix A. The code for the *interaction_control* node can be seen in appendix C.

6.3 Manual Control

Manual control of MARVIN is necessary for mapping new environments, transporting MARVIN between test environments and as a manual override during autonomous tests. Manual control is achieved using a wireless XBox 360 controller and is interfaced with ROS using the *joy* node [86]. The */joy* messages are processed by the *manual_control* node and published as *geometry_msgs/Twist* and *std_msgs/Bool* messages. This communication structure is shown by figure 6.6.

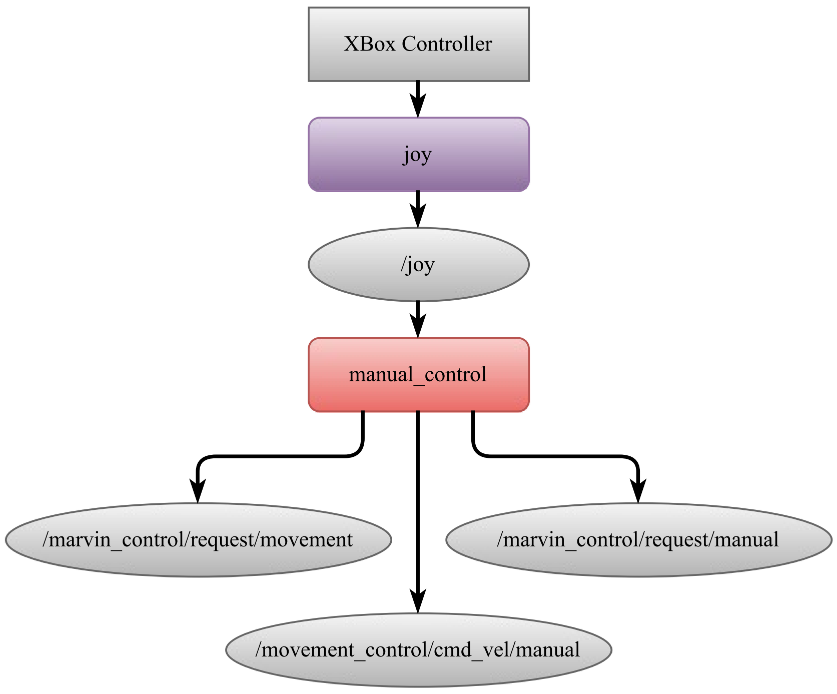

XBox Controller
joy
/joy
manual_control
/marvin_control/request/movement
/marvin_control/request/manual
/movement_control/cmd_vel/manual

Figure 6.6: Block diagram of the Manual Control node.

The y axis of the left analog stick and the x axis of the right analog stick are used to produce the linear and angular velocity components of the *geometry_msgs/Twist* message. This velocity command is published to */movement_control/cmd_vel/manual* topic. The A and B buttons are used to request that movement is enabled and disabled by publishing *std_msgs/Bool* messages to the */marvin_control/request/movement* topic. The X and Y buttons are used to request that manual control is enabled and disabled by publishing *std_msgs/Bool* messages to the */marvin_control/request/manual* topic. The control layout is visualised by figure 6.7.

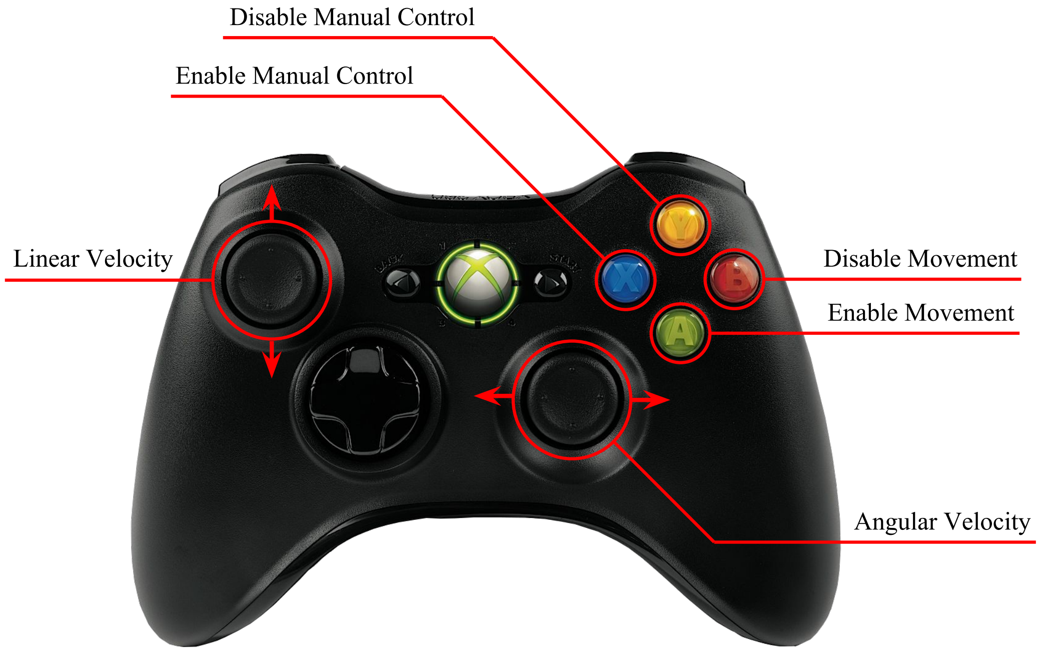


Figure 6.7: Control layout for the Xbox 360 controller.

The code for the *manual_control* node can be seen in appendix C.

6.4 MARVIN Control

The high level control node, *marvin_control*, monitors and manages MARVIN's other control nodes and subsystems. It can be used to control MARVIN's overall behaviour by processing high level goals and passing them down to lower level control nodes. It has three major tasks; monitoring and managing the other control nodes, interfacing with MARVIN's GUI (explained in Chapter 7) and interfacing with the navigation stack (explained in Chapter 5).

To improve the performance and robustness of *marvin_control*, it operates over three separate threads. The first thread manages the other control nodes and controls MARVIN's overall behaviour, further explained in section 6.4.1. The second thread interfaces with the GUI, discussed in section 6.4.2. The third thread interfaces with the navigation stack, detailed in section 6.4.3.

6.4.1 Main Thread

The purpose of this thread is to control MARVIN's overall behaviour and ensure robust operation. This is achieved by monitoring the other control nodes' statuses and controlling them when appropriate. This is shown in figure 6.8.

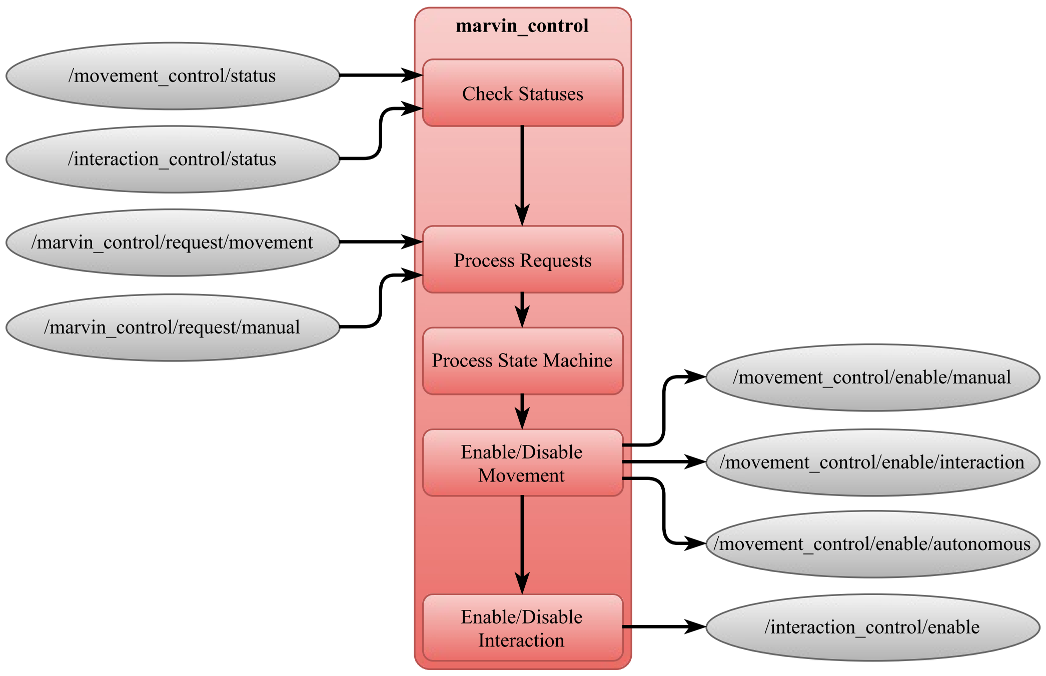

marvin_control
Check Statuses
Process Requests
Process State Machine
Enable/Disable Movement
Enable/Disable Interaction
/movement_control/status
/interaction_control/status
/marvin_control/request/movement
/marvin_control/request/manual
/movement_control/enable/manual
/movement_control/enable/interaction
/movement_control/enable/autonomous
/interaction_control/enable

Figure 6.8: Block diagram of the MARVIN Control node.

First, the *movement_control* and *interaction_control* statuses are checked for faults. If faults are detected they can be shut down in a controlled manner. If status messages have not been received for longer than a 2 s threshold it is assumed that the control node has disconnected. In this case MARVIN is commanded to become inactive until the control nodes have reconnected. Second, any requests from the *manual_control* are stored from the */marvin_-control/request/movement* and */marvin_control/request/manual* topics. Third, the state-machine shown in figure 6.9 is processed. This will be further explained in the next paragraph. The information from the processed requests and state machine are then used to enable and disable the various movement control sources and the *interaction_control* node.

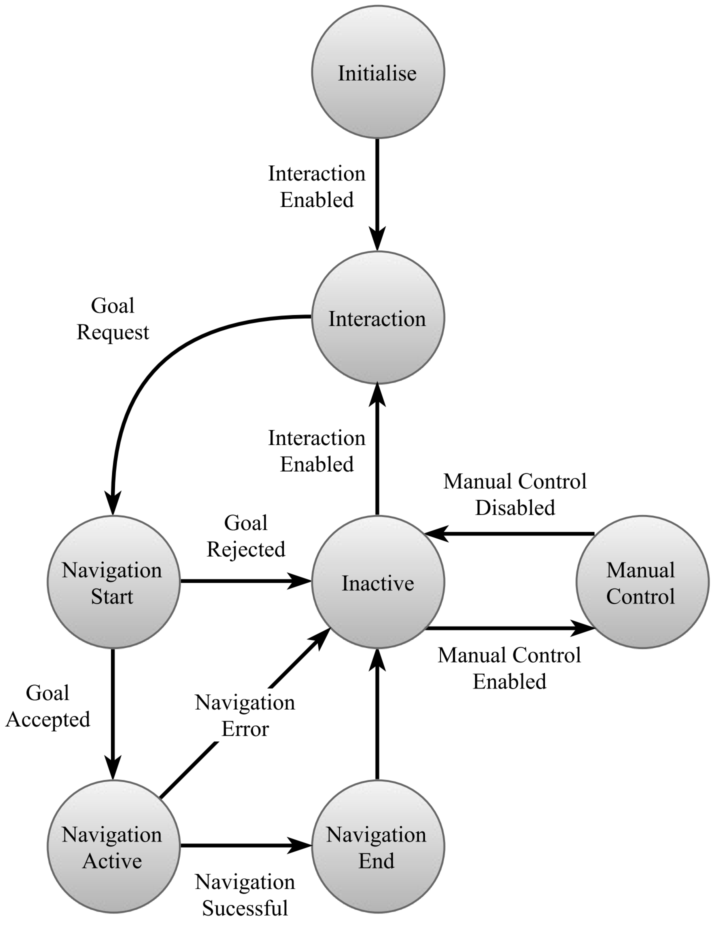


Figure 6.9: The MARVIN control internal State Diagram.

Figure 6.9 shows the state machine used to control MARVIN's behaviour. Each state enables and disables the *movement_control* and *interaction_control* nodes as well as the movement command sources according to table 6.1. The *Initialise* state is used to initialise the *marvin_control* node at launch. The *Interaction* state is the default state in normal operation. In this state a user can interact with MARVIN's GUI, and MARVIN's pose changes according to the user's position (as discussed in section 6.2). The *Navigation Start* state is used to communicate the goal request to the navigation stack (as discussed in Chapter 5) and wait for the user to inform MARVIN that they are ready to be guided (as discussed in Chapter 7). The *Navigation Active* state is used to ensure the other control nodes and GUI are aware that the navigation stack has control of MARVIN. The *Navigation End* state is used to inform the other control nodes and GUI that the navigation stack is no longer in control of MARVIN. The *Inactive* state is used as a intermediary state when going from one from one behaviour to another. It clears any existing commands and ensures the next behaviour has uncontested control of MARVIN. The *Manual Control* state informs the other control nodes that MARVIN is being controlled manually and prevents the navigation and interaction stacks from taking control.

Table 6.1: MARVIN Control State Overview

State	Control	Enabled/Disabled
Initialised **Inactive** **Navigation Start** **Navigation End**	Movement Control	Disabled
	Movement (Manual)	Disabled
	Movement (Interaction)	Disabled
	Movement (Navigating)	Disabled
	Interaction Control	Disabled
Interaction	Movement Control	Enabled
	Movement (Manual)	Disabled
	Movement (Interaction)	Enabled
	Movement (Navigating)	Disabled
	Interaction Control	Enabled
Navigation Active	Movement Control	Enabled
	Movement (Manual)	Disabled
	Movement (Interaction)	Disabled
	Movement (Navigating)	Enabled
	Interaction Control	Disabled
Manual Control	Movement Control	Enabled
	Movement (Manual)	Enabled
	Movement (Interaction)	Disabled
	Movement (Navigating)	Disabled
	Interaction Control	Disabled

6.4.2 GUI Thread

The GUI is used to allow humans to interact with MARVIN, as discussed in Chapter 7. To enable this interaction, the GUI must be able to communicate with the *marvin_control* node. As the GUI runs on a tablet running the Windows OS, standard ROS messages cannot be used for communication.

Instead, Transmission Control Protocol (TCP) serial communication is used. This method of communication is visualised in figure 6.10.

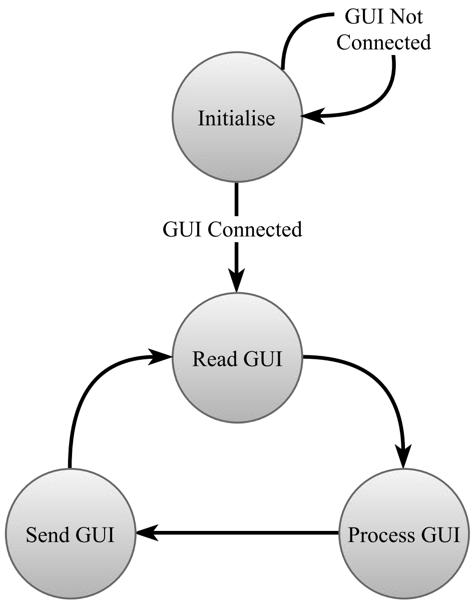


Figure 6.10: MARVIN control GUI interface state diagram.

First, the communication is initialised. This involves creating a TCP socket with the NUC as host, then waiting for a client to connect. Once a client has connected, a handshaking process is conducted to ensure the client is MARVIN's GUI. The *Initialise* state is maintained until a client has connected with a successful handshake. Next, the TCP socket is read. This process can detect if the client has disconnected. If this is the case *Initialise* state is reinstated. Otherwise, the read data is compared to the list of known requests and then processed. Once processed, the result is sent to the GUI via the TCP socket. Once again, this process can detect if the client has disconnected. Both the *Read GUI* and *Send GUI* steps use blocking processes. This requires that the GUI interface runs on a separate thread from the rest of the *marvin_control* node's processes.

There are four types of requests the GUI can make; *update*, *stop*, *start* and *guide*. The *update* request sends updated data without changing MARVIN's behaviour. The *stop* and *start* requests are used when MARVIN is in the *Navigation Active* state to allow the user to enable and disable MARVIN's movement. The *guide* request enables the GUI to request navigation goals. The goal is a *string* that contains the name of the location. This means that the location must be known by both the GUI and the control stack. Locations are stored in the *locations* file with values for *name, x, y* and *yaw*. The *marvin_control* node responds to each request using the same message format shown in table 6.2.

Table 6.2: Format of the *marvin_control* node's response to GUI requests.

MARVIN State	Navigation Status	Human Detected	Location $(x,\ y,\ yaw)$	Command	Result

The same message structure is used for all responses so that the GUI maintains up to date information from the *marvin_control* node. The *MARVIN State* informs the GUI which state the main thread is in. This is used to change the GUI's display accordingly (further explained in Chapter 7). The *Navigation Status* informs the GUI what state the navigation thread is in. This is used so that the GUI knows when the request goal has been met or has failed. *Human Detected* informs the GUI if a human is currently detected, which is used to determine when a new user has arrived or when the current user has left. The *Location* is used to inform the user of their location by drawing it on a map. The *Command* contains the command that *marvin_control* is responding to and is used to detect if the communication has become out of sync. The *Result* informs the GUI of the outcome of the command that *marvin_control* is responding to.

6.4.3 Navigation Thread

The *marvin_contol* node must be interfaced with the navigation stack to request goals and detect when those goals are met. This interface is visualised by figure 6.11.

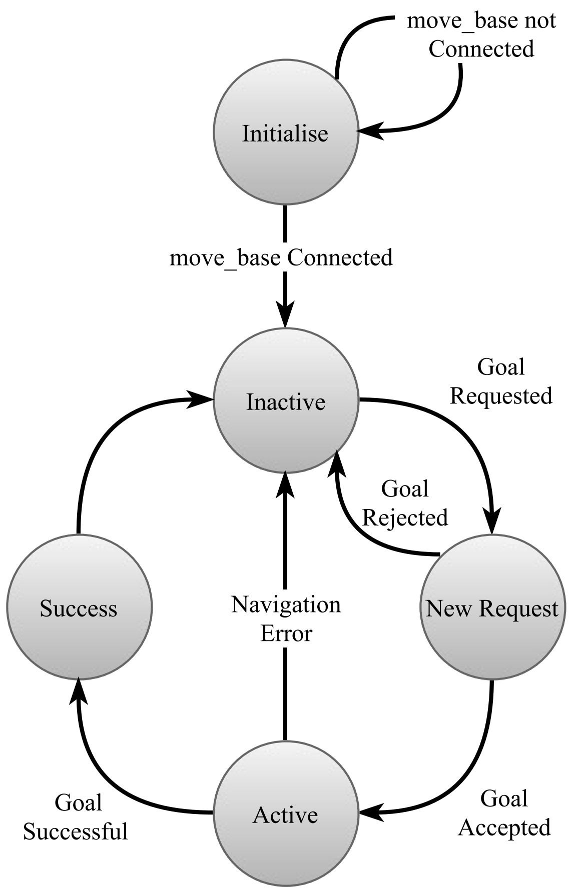


Figure 6.11: MARVIN control navigation stack interface state diagram.

The *Initialise* state waits until the *move_base* action server has connected (further explained in section 5.3). The *Inactive* state waits for a new goal request from the GUI. During this state, the navigation stack is running in the background to maintain MARVIN's location estimates. Once a goal request has been received, the *New Request* state is entered. The goal request is checked against the list of known locations. If it is valid, it is sent to the *move_base* action server. The *Active* state waits until the navigation either returns a success or a failure. If it results in failure, it informs the main and GUI threads, then moves to the *Inactive* state. Otherwise it moves to the *Success* state, which informs the main and GUI threads and proceeds to the *Inactive* state.

The code for the *marvin_control* node can be seen in appendix C.

Chapter 7

Human-Robot Interaction

As introduced in section 1.2, the aim of this project is to develop MARVIN into a research platform for human-robot interaction. To demonstrate MARVIN's effectiveness at meeting this aim, MARVIN is implemented as a robotic guide. To enable MARVIN's performance of this HRI task, all its subsystems must be fully integrated, including: locomotion, sensing, localisation, obstacle detection, motion planning and human-robot interaction. The successful completion of this HRI task helps evaluate MARVIN's suitability as a HRI research platform.

This chapter details the development of this HRI task. Section 7.1 outlines its aims and objectives, section 7.2 discusses the development of its user interface (UI), section 7.3 explains how MARVIN's torso is used to engage users and section 7.4 evaluates the success of the HRI task through a usability study.

7.1 HRI Task's Aims and Objectives

The aim of the HRI task is to guide users through an unmodified office-like environment. To achieve this aim MARVIN must be capable of meeting the following objectives:

1. **Detect a new user and introduce MARVIN.** Demonstrates the integration of MARVIN's human detection capabilities and HRI elements.

2. **Display the user's location.** Demonstrates the integration of MARVIN's localisation and HRI elements.

3. **Guide the user to a selected location.** Demonstrates the integration of obstacle detection, motion planning and HRI elements.

4. **Clearly communicate through voice synthesis and recognising.** Evaluates the effectiveness of MARVIN's vocal communication.

5. **Provide an intuitive interface through the tablet's touchscreen.** Evaluates the effectiveness of MARVIN's touchscreen interface.

6. **Engage the user by actuating the humanoid torso.** Demonstrates the integration of the robotic torso with the other HRI elements.

Objectives 1 to 5 are implemented through the development of the task's user interface, explained in section 7.2. Objective 6 is discussed in section 7.3. The task's implementation is evaluated against these objectives with a usability study, discussed in section 7.4.

7.2 User Interface

The user interface allows the user to make requests to MARVIN, whilst MARVIN feeds information back to the user. This is achieved through a graphical user interface (GUI) and a vocal interface. The GUI makes use of the tablet's touch screen to display information to, and receive requests from, the user (in the form of button presses). As explained in section 2.5.1, touch screens provide a versatile receptive and expressive element. The development of this GUI is explained in section 7.2.1. The vocal interface utilises the tablet's speakers and microphone with voice synthesis and voice recognition to communicate with the user through audio. Voice synthesis and recognition provide an interface that does not require line-of-sight (unlike the GUI), as discussed in sections 2.5.1 and 2.5.2. The implementation of the vocal interface is discussed in section 7.2.2.

The GUI and vocal interfaces are both controlled through the same state machine. This improves the coherency between both forms of interface. The state machine is designed to lead the interaction between the user and MARVIN. This limits the number of likely responses from the user, improving the robustness of the UI. It also avoids inflating the user's expectations of MARVIN. If the communication was led by the user, it could give them the belief that they can have an unlimited conversation with MARVIN. This invalid notion frustrates the user when the system fails to understand their requests. Having MARVIN drive the interaction helps avoid the illusion that MARVIN can have an unlimited conversation with the user, which better aligns the user's expectations with MARVIN's functionality.

The HRI application's state machine is shown in figure 7.1. The letters in each state correspond to the GUI screens shown in figure 7.3.

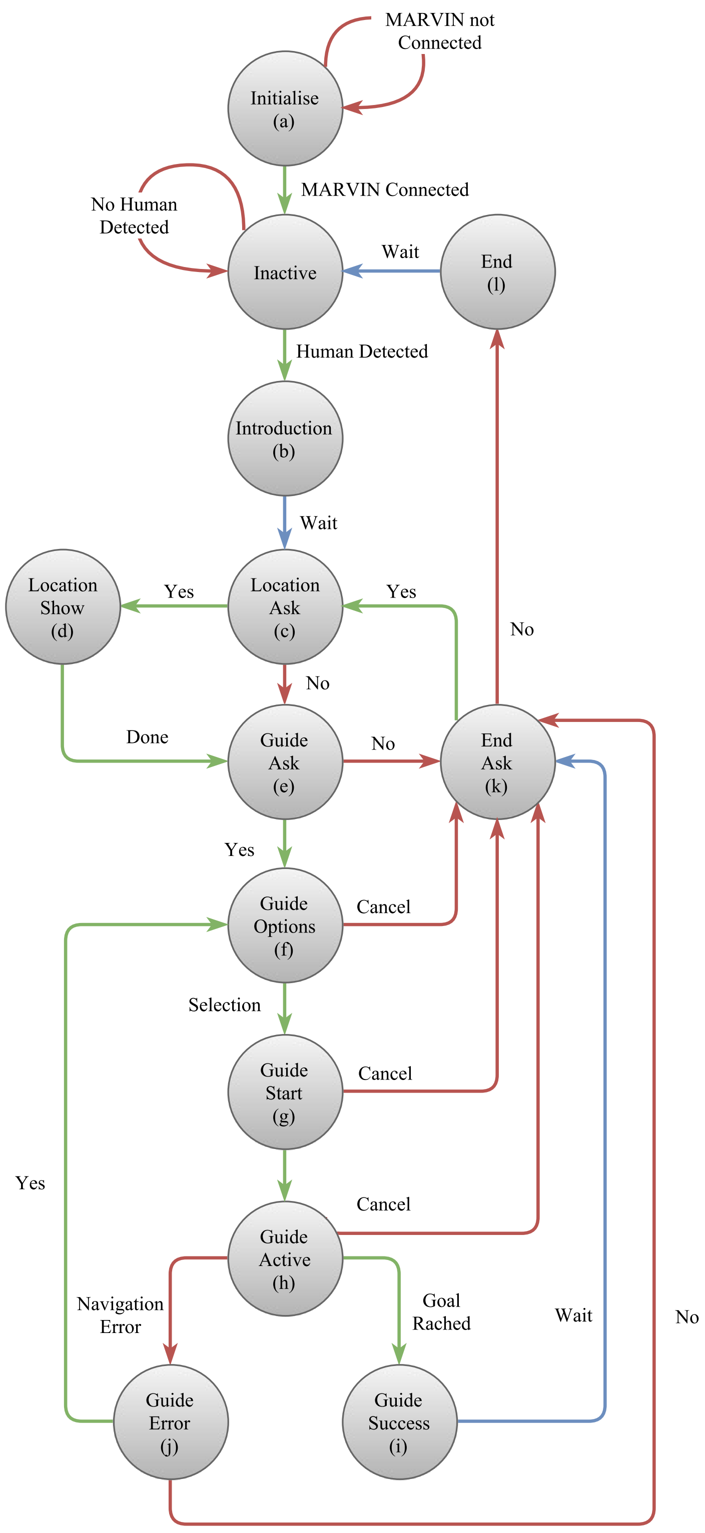


Figure 7.1: HRI application's state machine. Green lines represents positive responses, red represents negative responses and blue represents time delays.

The *Initialise* state waits for a successful connection with MARVIN's central control node *marvin_control* (explained in section 6.4). The communication between the HRI application layer and control layer uses TCP, as explained in section 6.4.2. After a successful handshake with the *marvin_control* node, the state machine moves to the *Inactive* state.

During the *Inactive* state, MARVIN actively seeks out a user by turning slowly on the spot, scanning its surrounding area for humans. Once a user has been detected, the *Introduction* state is set. The *Introduction* state introduces MARVIN to the newly detected user, saying "Hello, I am MARVIN". This meets HRI objective 1, as explained in section 7.1. After a short time delay, the *Location Ask* state is set.

The *Location Ask* state asks the user "Would you like to see our location?". If the the user replies in the affirmative, the *Location Show* state is set, otherwise the *Guide Ask* state is set.

The *Location Show* state displays a map of the environment with MARVIN's location marked on it and says "Here is our location, tell me when you are done". The location is estimated from the *amcl* node, as explained in section 5.1. This meets the HRI objective 2, as explained in section 7.1. The *Location Show* state is maintained until the user confirms that they have finished observing the location map, which sets the *Guide Ask* state.

The *Guide Ask* state asks the user "Would you like me to guide you somewhere?". A positive response sets the the *Guide Options* state, while a negative response sets the *End Ask* state.

The *Guide Options* presents the user with the set of possible goal locations, asking "Where would you like to go?". As the aim of the HRI application is to guide users, it is assumed that the user does not know where their goal location is, but they do know the name of it (for example, a particular room or office; such as CO239 or AM218). Because of this, they are provided with a list of known locations that they can select from. Once they select a goal location, the name of the location is sent to *marvin_control* and the *Guide Start* state is set.

The *Guide Start* state requests that the user "Tell me when you are ready to go.". This allows the user to get in a safe position behind MARVIN, before MARVIN starts moving. It also prevents the user from getting startled by MARVIN moving unexpectedly. When the user confirms that they are ready, the *Guide Active* state is set.

The *Guide Active* state requests to enable MARVIN's movement through the *marvin_control* node and informs the user "All right, let's go. Tell me if we need to stop." During this state the user can request MARVIN to stop or continue at any point, which can be used in fail conditions, or if the user wants to temporarily stop during the guiding process. This state is maintained until the *marvin_control* node has reported that MARVIN has reached the goal location, which sets the *Guide Success* state, or that the navigation stack has encountered an error (for example if there is no path found to the goal), which sets the *Guide Error* state.

The *Guide Error* state asks "Sorry, I was unable to guide you. Would you like to try again?". If a positive response is received, the *Guide Options* state is set, allowing the user to change the guide location goal if necessary. Otherwise, the *End Ask* state is set.

The *Guide Success* state informs the user that "We have arrived.", then moves to the *End Ask* state after a short delay (which gives MARVIN enough time to turn and face the user). The user can use the "cancel" response during the *Guide Options*, *Guide Start* and *Guide Active* states to cancel MARVIN's movement and move to the *End Ask* state.

The *End Ask* state asks "Can I do anything else for you?". A positive response sets the *Location Ask* state, repeating the interaction process. A negative response sets the *End* state.

The *End* state says "OK, See you later.", informing the user that the interaction has finished. After no human has been detected for a specified time, the *Inactive* state is set. This waits for a new user and then repeats the whole interaction process.

The user interface has been developed with the Windows Presentation Foundation (WPF). WPF is Microsoft's latest approach to a GUI framework and makes use of the .NET framework [87]. A GUI framework provides all the necessary elements required for developing a GUI, such as: labels, textboxes, buttons and event handlers. Having the underlying code provided for these elements simplifies the development process. WPF was specifically selected for three reasons. Firstly, MARVIN's tablet computer runs the Windows operating system for better voice synthesis and recognition support, as explained in section 3.2.3. This necessitates a GUI framework that supports Windows. By using one of Microsoft's frameworks (such as WPF) it is guaranteed to support Windows. Secondly, WPF is designed to work on touch screen devices. This makes it more effective for developing on MARVIN's tablet than the older GUI frameworks that are designed primarily for keyboards and mice. Thirdly, WPF is developed to make use of the .NET framework. The .NET framework is a group of libraries that provides a large range of functionality, including (most notably for MARVIN) voice synthesis and voice recognising. WPF applications are written in a combination of XAML and C#.

7.2.1 Graphical User Interface

The GUI is the primary source of information for the user. As such, it is required to be clear and understandable. To achieve this, considerations must be made for all the GUI elements, including: size, layout and colour.

These considerations will be discussed throughout this section. The GUI has two functions, present information to the user, and receive responses from the user. Information is provided to the user through text and imagery, while responses are received from the user through buttons. This GUI is developed to meet the HRI objective 5, as explained in section 7.1.

The GUI is split into four primary areas: the *Frame Title*, *Primary Button Panel*, *Status* and *Secondary Button Panel*. These areas are shown in figure 7.2.

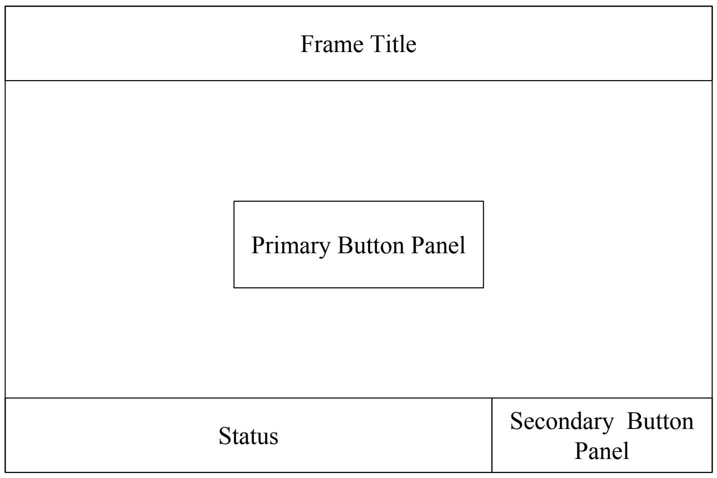


Figure 7.2: Basic layout of the HRI guide application's GUI.

The *Frame Title* is used to inform the user what the current frame is for. It typically displays what MARVIN has said. It is located at the top of the screen to encourage the user to read it first, providing context for the rest of the elements in the frame. The text in the *Frame Title* is large (72 pt) and white, which contrasts against the dark grey background improving readability and making it stand out.

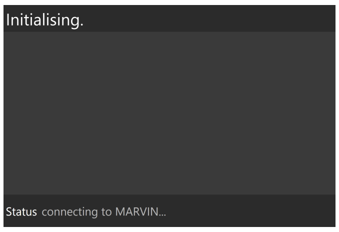
The *Primary Button Panel* houses the majority of the response buttons. It is placed in the centre of the screen, making it the second element the user should notice as they observe the frame. Its central location also emphasises that it is an interactive element that the user can press, which helps improve the intuitiveness of the GUI.

The *Status* element displays the UI's current status. It provides the user with additional information, such as: ″connecting to MARVIN″ in the *Initialising* state, ″waiting for human″ in the *Inactive* state and ″listening″ when the GUI is waiting for a response from the user. As the *Status* is a less important element than the *Frame Title* and *Primary Button Panel* elements, it is located at the bottom of the screen. It also uses a smaller font than the *Frame Title* (50 pt) and a light grey colour with less contrast against the dark grey background.

The *Secondary Button Panel* is used to house buttons if there is other information displayed in the centre of the screen (such as in the *Location Show* state), or to house secondary buttons (such as ″cancel″). It is located at the bottom right of the screen, making it the last element the user should notice. This is because these buttons only have relevance once the rest of the frame has been observed.

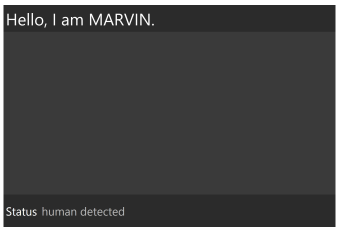
The buttons are designed to be clear and intuitive to use. They are large enough to touch (45 mm $\times$ 22 mm) and use vibrant colours to contrast against the dark grey background. The buttons' text (used to explain the buttons' purpose) uses a large (60 pt) white font that contrasts against the vibrant button colour, improving its clarity. The buttons' colour follows a coherent theme to help make the buttons' purpose more intuitive. Positive buttons are green (such as ″yes″ and ″start″), negative buttons are red (such as ″no″, ″cancel″ and ″stop″) and option buttons are blue (such as the guide locations).

Figure 7.3 presents a screen shot of each of the GUI's frames. The figure letters correspond to the states in figure 7.1.


Initialising.

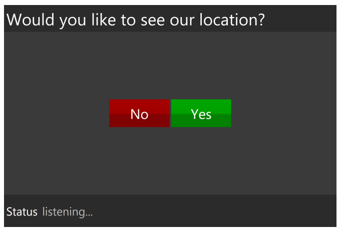
Status connecting to MARVIN...

(a) Initialisation


Hello, I am MARVIN.

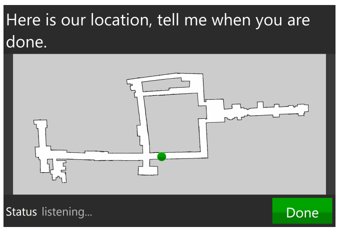
Status human detected

(b) Introduction


Would you like to see our location?

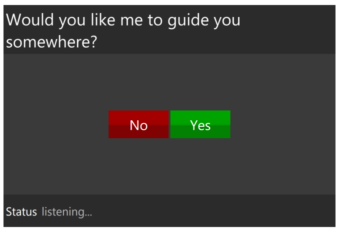
Status listening...

(c) Location Ask


Here is our location, tell me when you are done.

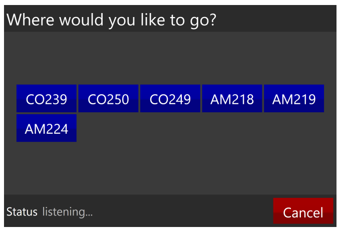
Status listening...

(d) Location Show


Would you like me to guide you somewhere?

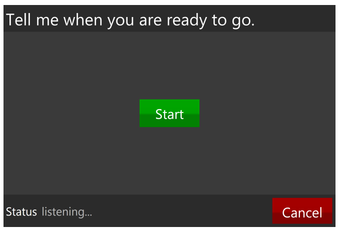
Status listening...

(e) Guide Ask


Where would you like to go?

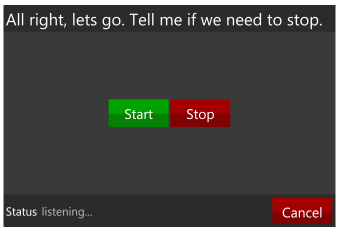
Status listening...

(f) Guide Options


Tell me when you are ready to go.

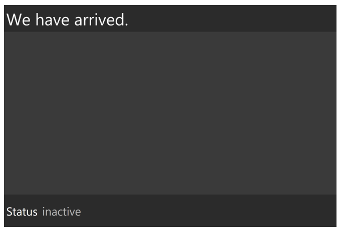
Status listening...

(g) Guide Start


All right, lets go. Tell me if we need to stop.

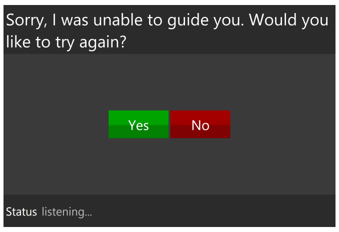
Status listening...

(h) Guide Active


We have arrived.

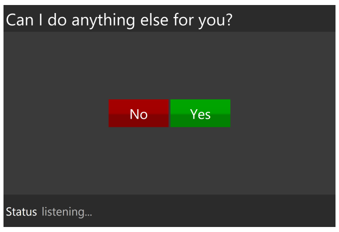
Status inactive

(i) Guide Success


Sorry, I was unable to guide you. Would you like to try again?

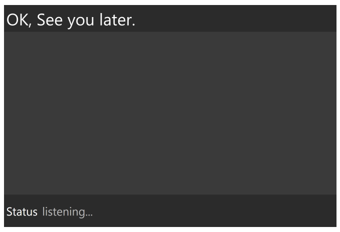
Status listening...

(j) Guide Error


Can I do anything else for you?

Status listening...

(k) End Ask


OK, See you later.

Status listening...

(l) End

Figure 7.3: HRI application's GUI frames.

7.2.2 Vocal Interface

As explained at the start of section 7.2, the UI is written using the WPF. This allows the integration of Microsoft's .NET speech platform, which is implemented to meet the HRI objective 4, as explained in section 7.1. Listing 7.1 shows the setup required for the voice synthesiser. Listing 7.2 shows the setup required for the voice recogniser.

```
1  using System.Speech.Synthesis;
2
3  // Setup the Voice Synthesiser
4  SpeechSynthesizer synth = new SpeechSynthesizer();
5  synth.SetOutputToDefaultAudioDevice();
6  synth.SelectVoiceByHints(VoiceGender.Male);
```

Listing 7.1: Voice Synthesiser Setup

Line 1 includes the .NET speech platform voice synthesis library. Line 4 initialises the speech synthesis object. Line 5 sets the audio output for the synthesiser to the default audio device (the tablet's speakers). Line 6 sets the voice of the synthesiser to a male voice (if one is installed). Unfortunately Microsoft have removed native support for different voices, so the standard Microsoft Ann voice is used. This should be replaced with a more MARVIN appropriate voice in future.

```
1  using System.Speech.Recognition;
2
3  // Setup the voice recogniser
4  SpeechRecognitionEngine recogniser = new
       SpeechRecognitionEngine();
5
6  // Add the dictionary of commands to choices
7  Choices choices = new Choices();
8  foreach (KeyValuePair<string, List<string>> entry in _commands)
9      foreach (String choice in entry.Value)
10         choices.Add(choice);
```

```
11
12 // Build the grammar for the voice recognition.
13 GrammarBuilder gb = new GrammarBuilder();
14 gb.Append(choices);
15 Grammar g = new Grammar(gb);
16 recogniser.LoadGrammar(g);
17 recogniser.SpeechRecognized += new
       EventHandler<SpeechRecognizedEventArgs>(sre_SpeechRecognized);
18 recogniser.SetInputToDefaultAudioDevice();
19 recogniser.RecognizeAsync(RecognizeMode.Multiple);
```

Listing 7.2: Voice Recogniser Setup

Line 1 includes the .NET speech platform voice recognition library. Line 4 initialises the speech recogniser object. Lines 7 to 10 initialises a *Choices* data set and adds the values from the dictionary of commands to it. The dictionary of commands is further explained in the following paragraph. Lines 13 to 15 initialises a *GrammarBuilder* object, loads the set of choices into it and then creates a *Grammar* object from the *GrammarBuilder*. Line 15 loads the *Grammar* object into the speech recogniser. Line 17 creates an event handler for the speech recogniser. This allows the recogniser to act like a button press when it detects a known word. Line 18 sets the speech recogniser's audio input to the default audio device (the tablet's microphone). Line 19 starts the voice recogniser in its asynchronous mode, which allows it to operate in parallel (on a separate thread) with the main program.

Originally, likely alternatives for each response were added to the *SpeechRecognitionEngine's GrammarBuilder*. For example, the positive response included "yes", "yeah", "yea", "yup", "OK" and "sure". However, during testing it was found that the larger the set of recognisable words in the data base, the larger the chance of false positives. False positives can trigger the UI to change state unexpectedly, which can frustrate the user. It was also noted during a pilot study that the majority of people used the buttons' text as their verbal response, rather than using alternative words. Therefore the number of possible responses was restricted to the button labels. These known keywords are stored in the dictionary of commands, which is loaded into the *GrammarBuilder*.

The full GUI code can be found in Appendix C.

7.3 Interactive Poses

As explained in section 6.2, MARVIN's robotic torso pose can be changed depending on the detected human's location, through the use of configurable zones. This feature is used to attempt to meet the HRI objective 6, as explained in section 7.1. The HRI guide task makes use of five zones: *Close*, *Nominal*, *Left*, *Right* and *Far*. These zones are visualised in figure 7.4.

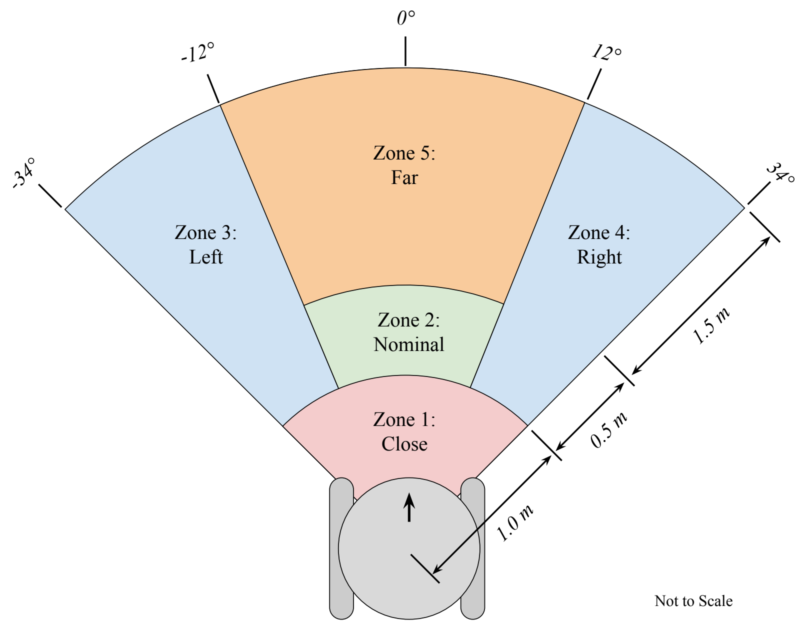
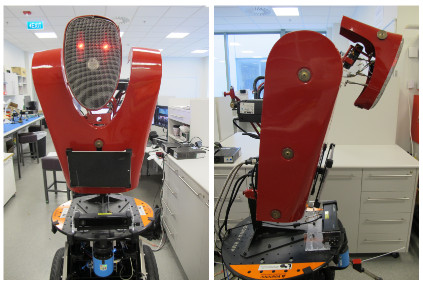


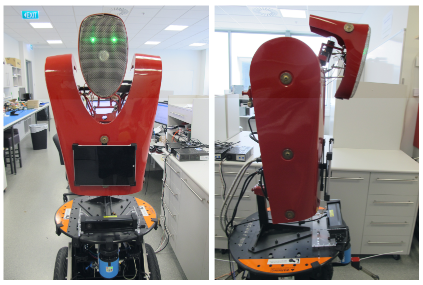
Figure 7.4: Configured Interaction Zones

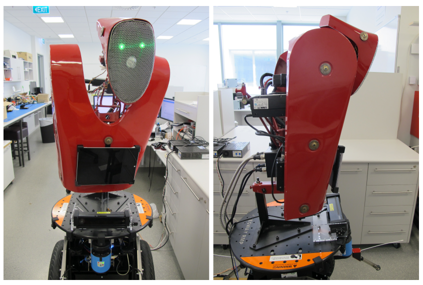
If a user is in the *Close* zone, they are considered to be too close to MARVIN. They might get injured or startled if MARVIN moves suddenly. As an attempt to dissuade people from entering this zone, the torso is set to an aggressive pose (as shown in figure 7.5a). The neck and shoulders are extended and the eyes are set to red. This aggressive pose is used to make the user feel uncomfortable and take a step back. However, through a pilot study it was found that this made the users excessively uncomfortable and discouraged them from further interaction with MARVIN. While this would be effective if MARVIN were operating as a security guard, it is counterproductive when acting as a guide. Because of this, the *Close* zone was disabled during the usability study.

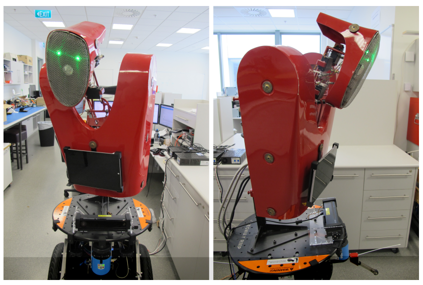
The *Nominal* zone is the optimal zone for standard interaction. The touch screen is easily observable, the voice synthesis is at an audible volume (60 to 65 dB) and the voice recognition is at its most reliable. The torso is set to an attentive pose, as shown by figure 7.5b. The shoulders are back, neck halfway extended (so that MARVIN's eyes are approximately in line with the user's) and the eyes are green. While the user is in this zone, Segway tracking is enabled. This commands the Segway to turn towards the user, giving the appearance that MARVIN is looking at the user.

If the user is within the *Left* or *Right* zones the torso's pose is changed to lean towards them, as shown in figures 7.5c and 7.5d. This gives the gives the appearance that MARVIN is straining to look at them, encouraging the user to move in front of MARVIN (into the *Nominal* zone). If the user is within the *Far* zone, the shoulder and neck extends. This gives the appearance that MARVIN is craning towards them, as if it were struggling to see or hear the user properly. This is shown in figure 7.5e.


(a) Zone 1: Close


(b) Zone 2: Nominal


(c) Zone 3: Left


(d) Zone 4: Right

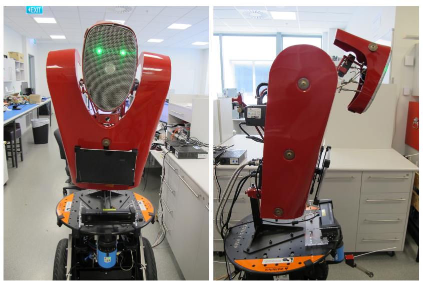

(e) Zone 5: Far

Figure 7.5: Torso poses for each interaction zone.

7.4 Usability Study

As stated at the start of this chapter, the HRI guide task is developed to demonstrate MARVIN's effectiveness as a HRI research platform. To accomplish this, the HRI task's implementation must be evaluated against the objectives specified in section 7.1. This evaluation is achieved through a usability study involving 18 participants with the required ethics approval obtained (included as appendix B). The participants were selected nondiscriminatively with people of differing backgrounds, including: sonic arts, law, bio-medical, physics, economics, computer science and engineering. The majority of participants were students with ages ranging from early 20's to mid 30's. This may introduce some response bias (such as aptitude towards technology). Further research should be conducted in the future to investigate how a broader range of ages react to MARVIN. No prior training is provided, other than a brief explanation of what MARVIN is.

MARVIN is initialised and set to the *Inactive* state. Participants interact with MARVIN one at a time, without seeing MARVIN in operation prior to the test. Each participant is asked to perform the following set of tasks:

1. Approach MARVIN and wait for it to initiate interaction.
2. Request a map of your current location.
3. Request to be guided to a location.
4. Follow MARVIN to the location.
5. Request to be guided to the original location.
6. End your interaction with MARVIN.

These tasks were conducted twice for each participant, once with the voice recognition disabled, then again with the voice enabled. This allows the participant to directly compare each input method.

Following the completion of these tasks, the participants are asked to fill in a questionnaire. The questionnaire contains a series of statements that the participants rate on the Likert scale, where 5 corresponds to ″Strongly Agree″ and 1 corresponds to ″Strongly Disagree″. Table 7.1 presents an summary of these results.

Table 7.1: Summary of usability study results

Number	Question	Mean	Std. Dev.
1	MARVIN's voice was easy to understand.	4.7	0.5
2	MARVIN's voice was loud enough to hear clearly.	4.4	0.7
3	The touch interface text was large enough to read.	4.8	0.4
4	The touch interface layout helped me interact with MARVIN.	4.4	0.7
5	Interaction with MARVIN was at a comfortable distance.	4.1	0.6
6	MARVIN understood my voice well.	3.3	1.1
7	Voice interaction (synthesis and recognition) was more useful than the touch screen interface.	3.8	1.2
8	Requesting my location was easy.	4.3	0.7
9	The displayed location map was easy to read.	4.5	0.9
10	Requesting to be guided was easy.	4.3	0.8
11	MARVIN moved too slowly.	2.2	1.0
12	MARVIN moved too quickly.	1.9	0.7
13	MARVIN's upper body movement improved my interaction experience.	3.6	1.1

Question 1 shows that 100% of the users found the voice easy to understand

(72% strongly agreed and 28% agreed). This demonstrates that Microsoft's speech platform is an effective method for voice synthesis.

Question 2 shows that 88.9% of the users felt that MARVIN's voice was loud enough to hear clearly. None of the users disagreed with this statement; however, 11.1% were neutral towards it. This suggests that the tablet's speakers are loud enough for low background noise (measured at 42 ± 0.5 dB in the test environment, with the voice volume measured at 60 to 65 dB from 1 to 1.5 m in front of MARVIN). However, in more noisy environments additional speakers may be required to ensure voice synthesis clarity.

Question 3 shows that 100% of the users found the text large enough to read (78% strongly agreed and 22% agreed). This demonstrates that the tablet's 12″ screen is appropriate for MARVIN's HRI and that the text sizes used can be used for future developments.

Question 4 shows that 88.9% of participants felt that the layout of the GUI helped them interact with MARVIN (50% strongly agreed and 38.9% agreed). This suggests that the layout is effective and understandable, and could be used for future HRI applications.

Question 5 shows that none of the users felt that the integration distance with MARVIN was uncomfortable (83.3% of people felt that it was comfortable and 16.7% of people were neutral). This also suggests that the tablet's 12″ screen is sufficiently large to read at a comfortable distance and that MARVIN's appearance as a whole is not particularly unsettling.

Question 6 shows that the voice recognition was effective for some users, but not others. The participants' answers are fairly bimodal, with 50% of users agreeing that MARVIN understood them well and 27.8% disagreeing. This suggests that Microsoft's speech recognition algorithm is not robust enough for HRI in its current form. This could be improved through the addition of better audio recording hardware (such as a microphone array

[88], rather than using the tablet's inbuilt microphone), or through the implementation of different speech recognition software (such as Nuance's Dragon NaturallySpeaking). These improvements are further discussed in section 9.2.

The speech recognition discrepancy is carried through to Question 7, which asks if the user agrees that the vocal interface was more useful than the touch interface. 72.2% of users agree with this statement, 16.7% disagree and 11.1% are neutral. This produces a standard deviation of 1.2, which makes it difficult to draw reliable conclusions. However, if the set of answers is split into users who found MARVIN could understand them or were neutral (≥ 3) and users who felt MARVIN could not understand them (> 3), then this discrepancy can be resolved.

Of the users who felt MARVIN could understand them, 78.6% agreed that the vocal interface was more useful and only 7.1% disagreed. This increases the mean to 4.1 and drops the standard deviation to 0.9, which provides a stronger relationship suggesting that most people found the vocal interface useful if MARVIN understood their voice.

Of the users who found that MARVIN did not understand their voice, 50% agreed and 50% disagreed that the vocal interface was more useful than the touch interface. This produces a mean of 3.0 and a standard deviation of 1.8, which cannot be used to draw any robust conclusions. However, this might mean that some participants felt that the voice interface would be more useful than the touch interface if MARVIN understood their voice more reliably.

The results from questions 6 and 7 suggest that despite the voice recognition software not understanding some users, the majority of people found it useful. As such, the combination of touch screen and voice interfaces complement each other and should continue to be used in future HRI applications.

From questions 11 and 12, no users felt that MARVIN moved too quickly and 16.7% of users felt that MARVIN moved too slowly. During these tests MARVIN's maximum velocity was set to 1.0 ms^{-1}. These results suggest that this limit was slightly low, and a maximum velocity of 1.1 or 1.2 ms^{-1} would be more appropriate. This aligns with the average human walking speed of 1.41 ms^{-1} [17].

Question 13 has a large spread with 55.6% of users agreeing that MARVIN's torso poses improved their experience and 33.3% of users disagreeing. This makes it difficult to draw any reliable conclusions, however it does suggest that further development is required to reach the full potential of the actuated torso. The current zone configuration is relatively simple and could be extended with more complex responses to the user's requests and actions. Another improvement would be to replace the LED eyes with an LCD screen, enabling full facial expressions such as the Baxter robot [8].

In addition to the 13 questions, the participants are also asked write down any comments they had under the following questions:

- What aspects did you enjoy about interacting with MARVIN?
- Are there any additional methods of interaction that you would like incorporated into MARVIN? For example, hand gestures such as pointing?
- Is there any additional functionality you would like incorporated into MARVIN?
- Was there anything about MARVIN's appearance that made interacting with it more difficult?

- What do you feel could be improved about MARVIN's existing features?
- Do you have any other comments?

What aspects did you enjoy about interacting with MARVIN? This question resulted a number of aspects that the users enjoyed when interacting with MARVIN, including: six comments about the voice interaction, three comments about the novelty of interacting with a self-balancing and/or autonomous robot, five comments about MARVIN's perceived polite character, five comments about MARVIN's human detection ability and looking at the user, six comments about the interactive poses (some overlap with the human detection comments), two comments about MARVIN's functionality, two comments about MARVIN's responsiveness and two comments about MARVIN's ease-of-use.

Are there any additional methods of interaction that you would like incorporated into MARVIN? This question resulted in six comments expressing interest in hand gestures (33.3% of participants), four users felt that the voice and touch interfaces are sufficient (22.2%), one comment suggesting the addition of an LCD screen for facial expression and simple emotions, one comment suggesting a mobile-phone app, one comment suggesting physical buttons and one requesting more head movement from MARVIN.

Is there any additional functionality you would like incorporated into MARVIN? Five comments suggested interest in MARVIN acting as an information point, answering questions like the popular personal assistant programs (Google Now, Apple Siri or Microsoft Cortana), or providing location specific information such as showing a requested location on the map. Three comments requested more personable features, such as telling jokes or doing a trick. One comment suggested providing an estimated time of arrival to the selected goal location, one comment suggested a follow me command, one comment suggested MARVIN asks detected humans to move out of the way when navigating (rather than the current ”stop and wait” behaviour) and one comment suggested a gentle tone to warn that MARVIN is about to move.

Was there anything about MARVIN's appearance that made interacting with it more difficult? Six comments noted that MARVIN could be intimidating at times, particularly in close proximity. This could be of particular use when used for security tasks. Two comments suggested that the tablet could be mounted higher and angled up slightly to improve readability and improve the usability of the touch interface. One comment noted that the tilt limiters and whiskers stick out and can be cumbersome.

What do you feel could be improved about MARVIN's existing features? Six comments requested more robust voice recognition, two comments requested natural language processing, two comments suggested more facial features, two suggestions to add a skirt to cover the electronics and mobility platform, one comment requested a male voice to match the masculine torso and one complaint that the LED eyes are too bright.

Do you have any other comments? The majority of these comments fit into one of the other categories and have been included in the previous paragraphs. However, there were six additional comments suggesting that the users enjoyed interacting with MARVIN.

A scan of the full responses can be found in appendix C.

7.5 Summary

This chapter demonstrated MARVIN's effectiveness as a HRI research platform through the implementation of a guide task. This included the development of a GUI and a vocal interface. The implementation of this task was evaluated against its objectives through a usability study.

This usability study found that:

- MARVIN can detect humans in a real-world situation. This was a feature that many participants commented about positively.
- MARVIN's localisation and HRI functionality are fully integrated. Users found it easy to request their location and read the displayed map.
- MARVIN's navigation stack and HRI functionality are fully integrated and operate with humans acting as dynamic obstacles. Users found it easy to request to be guided, but found 1.0 ms^{-1} slightly slow.
- MARVIN can communicate clearly through voice synthesis; however, voice recognition will require some further development to improve its robustness.
- The developed GUI was clear and intuitive to use. Many of its features could be used in future HRI applications.
- Many users enjoyed MARVIN's actuated torso poses; however, it could be further developed through the addition of an LCD screen to display facial expressions (in place of the LED eyes).
- Many users displayed an interest in hand-gestures and the development of MARVIN into an information point.

The successful implementation of this guide task demonstrates that MARVIN is capable of succesful HRI. MARVIN's locomotion, sensing, localisation and motion planning must now be evaluated to ensure MARVIN meets the objectives outlined in section 1.2. The results required to evaluate these systems are presented and discussed in Chapter 8.

Chapter 8

Overall Results

This chapter presents the results and analysis of MARVIN's key systems, including: locomotion, sensing, localisation, obstacle detection, human detection and motion planning. This is achieved through the characterising of the Segway RMP (section 8.1), the evaluation of MARVIN's navigation stack (section 8.2) and the analysis of MARVIN's human detection capabilities (section 8.3).

8.1 Segway RMP Characteristics

As explained in section 3.1.1, MARVIN uses a Segway RMP for locomotion. This mobility platform is characterised to determine safe velocity limits during autonomous operation, as per objective 3a. This is achieved by commanding it to drive in a straight line (from stopped) until a target velocity is met, then commanding MARVIN to stop. A large flat measurement surface is placed in front of MARVIN so that the odometry, laser rangefinder and Kinect data can be compared and analysed. This test is used to analyse:

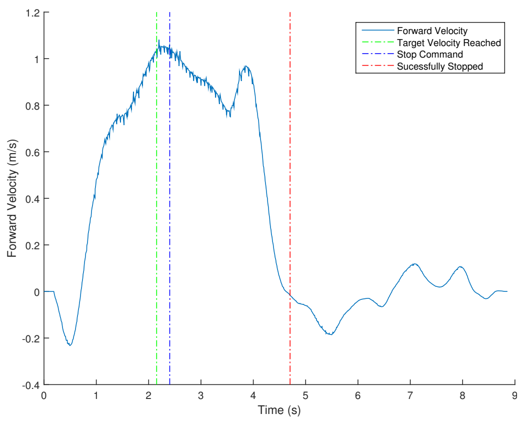
- Acceleration profile
- Deceleration profile
- Stopping distance
- Odometry data
- Laser Rangefinder data
- Kinect data

MARVIN can stop in two ways; standard-stop and emergency-stop. A standard-stop commands MARVIN to reduce its velocity to zero in a controlled manner, which maintains the Segway RMP's balance. An emergency-stop disables the Segway RMP's motors, causing MARVIN to fall onto its tilt limiters. The emergency stop is uncontrolled and should only be used as a last resort. Each of these stopping methods are tested at two target velocities; 1.0 ms^{-1} (a common maximum velocity for indoor mobile robots, such as *Jinny*) and 1.4 ms^{-1} (the average human's walking speed [17]).

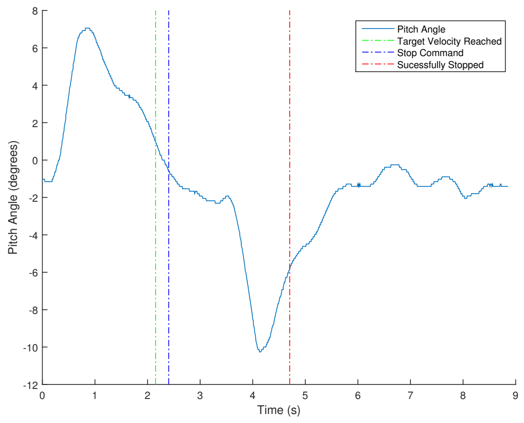
Section 8.1.1 presents the results for the standard-stop tests and section 8.1.2 presents the results for the emergency-stop tests. Section 8.1.3 provides a summary of the Segway RMP's characteristics.

8.1.1 Standard-Stop

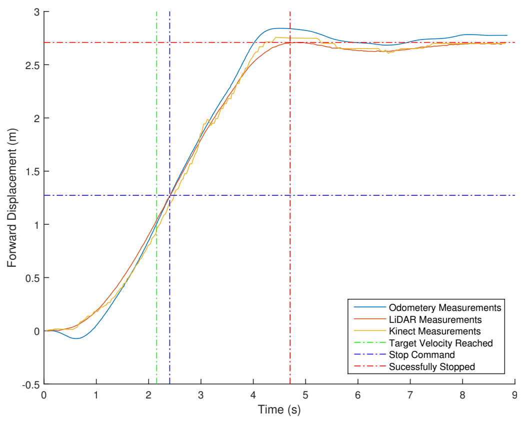
Figures 8.1 and 8.2 present a typical run of a standard-stop test with a target velocity of 1.0 ms^{-1} and 1.4 ms^{-1} respectively. Figures 8.1a and 8.2a show the velocity of MARVIN throughout the runs, figures 8.1b and 8.2b show the pitch, figures 8.1c and 8.2c show the forward displacement and figures 8.1d and 8.2d show the yaw displacement.



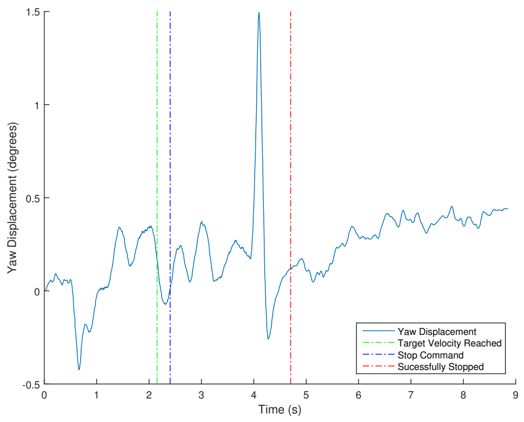
(a) Forward Velocity



(b) Pitch



(c) Forward Displacement



(d) Yaw Displacement

Figure 8.1: Standard stopping distance with a target velocity of 1.0 ms^{-1}

Figure 8.1a presents MARVIN's acceleration and deceleration profiles. When MARVIN is first commanded to move, it appears to initially move backwards (shown by the negative velocity peak at 0.5 s). However, when observing figure 8.1c, it can be seen that the laser rangefinder and Kinect do not detect this backward movement. Because of this, it can be determined that this backward motion is caused by odometry drift. Velocity is calculated from the odometry data (by integrating the forward displacement over time), so this odometry drift introduces errors into the velocity estimates. The odometry drift is caused by the way the Segway moves. To accelerate, the Segway leans forwards (as shown by the 7° peak at 0.5 s in figure 8.1b). To lean forward it applies enough torque to the motors to change the Segway's pitch, but not enough to rotate the wheels. However, this introduces a relative rotation between the Segway and its wheels, which is picked up by the odometers. This makes it appear that the Segway has move backwards, when in fact, it has remained stationary.

After this initial "deceleration", the Segway accelerates to its target velocity between 0.5 s and 2 s. This results in a slight overshoot ($\sim 5\%$); however, this is to be expected in a closed loop control system that is maintaining both pitch and velocity. Once the target velocity has been reached and maintained for the specified time threshold (0.2 s), the Segway is commanded to stop.

The Segway then decelerates slowly (at ~ 0.2 ms^{-2}), before a velocity peak at 4 s. Once again, this peak is caused by odometry drift and is artificial. This can be seen by the -10° peak at 4 s in figure 8.1b, where the Segway leans backwards to slowdown. This is essentially the inverse of the effect the initial acceleration caused (at 0.5 s). Because of this, the shape of both peaks are inverted versions of one another. This period of slow deceleration (2.5 s to 4 s) is likely caused by the Segway's control system attempting to maintain balance. This is suggested by the absence of this slow deceleration in the emergency-stop trial, shown in figure 8.3a.

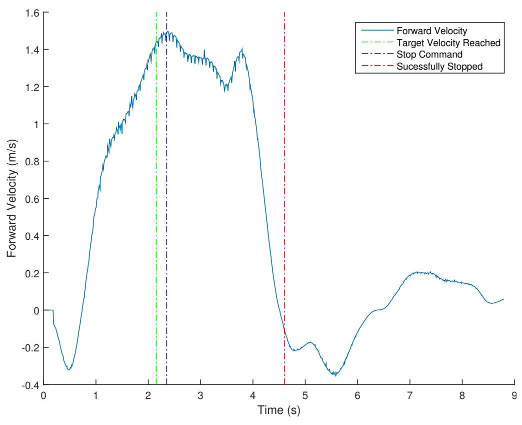
After the slow deceleration, the Segway's velocity drops suddenly (at ~ 1.8 ms^{-2}) until it reaches 0 ms^{-1}. As expected, this is followed by an overshoot ($\sim 20^\circ$), resulting in some backwards movement. After 6 s, the Segway's velocity continues to oscillate as it maintains balance.

Figure 8.1d shows that the yaw displacement is maintained within $0 \pm 0.5^\circ$, except for a spike at 4 s. This spike coincides with the ~ 1.8 ms^{-2}

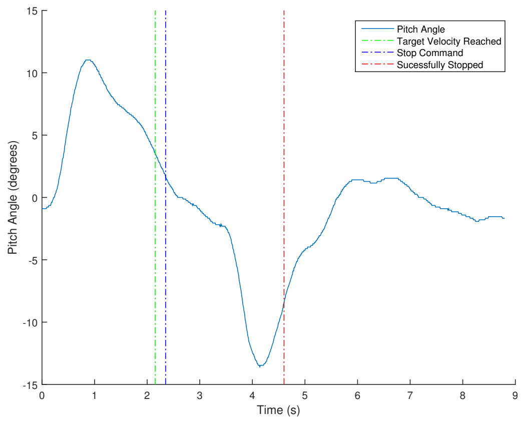
deceleration. It is likely caused by the wheels stopping at slightly different times.

Figure 8.1c, shows that the Kinect and laser rangefinder are consistent with one another. The odometry is consistent with the range measurements, except at points of large acceleration (or deceleration) which results in odometery drift. The consistency of the laser rangefinder and Kinect data demonstrates that the Kinect processing, particularly the ground removal algorithm, is effective over the range of Segway pitches ($-10°$ to $8°$).

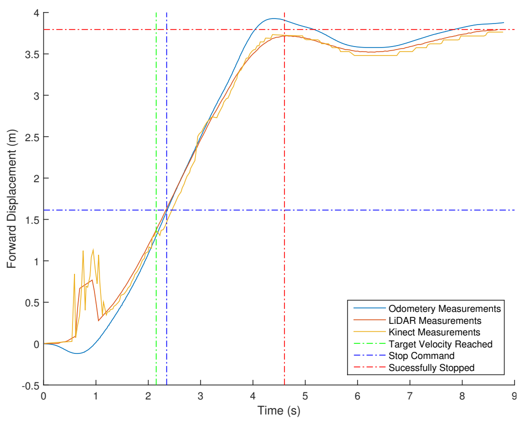
Figure 8.2 presents a typical run of the standard-stop test at 1.4 ms^{-1}. Many of the same observations can be made as the 1.0 ms^{-1} test; however, there are three key differences. Firstly, the Segway accelerates at ~ 0.7 ms^{-2}, resulting in an overshoot of $\sim 7\%$ and decelerates at ~ 2.8 ms^{-2}, resulting in an overshoot of $\sim 25\%$. Secondly, the Segway's pitch ranges from $-14°$ to $12°$. Thirdly, there are errors in the laser rangefinder and Kinect measurements from 0.5 s to 1.1 s. These errors are caused by the $12°$ peak in the Segway pitch. At this angle the laser rangefinder, briefly detects the ground plane, measuring points closer than the measurement surface. This problem is fixed through the addition of the laser scan ground filter, explained in section 4.3.2. The Kinect's error is caused by noise introduced by sudden changes in pitch. This issue is addressed in section 8.2.1.



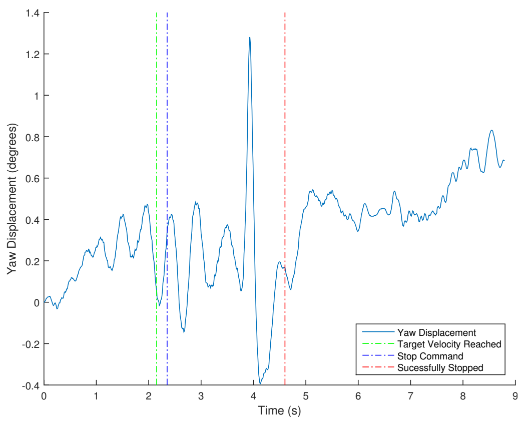
(a) Forward velocity



(b) Pitch



(c) Forward displacement



(d) Yaw

Figure 8.2: Standard stopping distance with a target velocity of 1.4 ms^{-1}

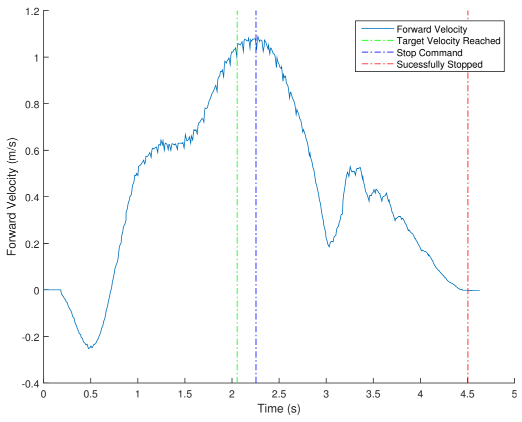
8.1.2 Emergency-Stop

Figures 8.3 and 8.4 each show a typical run of a emergency-stop test with a target velocity of 1.0 ms^{-1} and 1.4 ms^{-1} respectively. Figures 8.3a and 8.4a show the velocity of MARVIN throughout the runs, figures 8.3b and 8.4b show the pitch, figures 8.3c and 8.4c show the forward displacement and figures 8.3d and 8.4d show the yaw displacement.

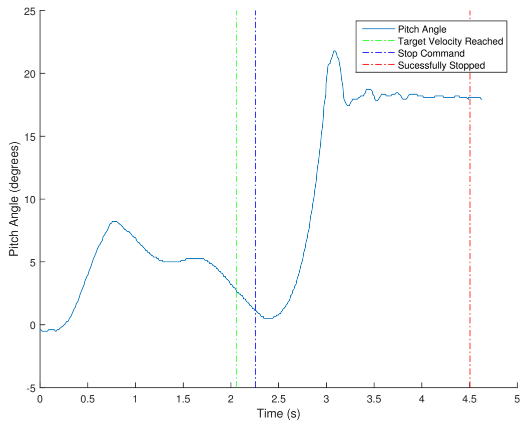
As expected, the acceleration profile is consistent with the standard-stop tests. However, as the emergency-stop disables the Segway's motors, as opposed to performing a controlled stop, the deceleration profile has some major differences. Because the Segway makes no attempt to control its balance, there is no artificial peak or period of slow deceleration after the stop command. Instead, the Segway falls forwards onto its tilt limiters, resulting in a 20° pitch. Once the the Segway lands, its velocity slowly decreases (at ~ 0.4 ms^{-2}) due to friction as it rolls on its tilt limiters. As the motors are not powered, this results in a much wider variation in yaw when compared to the standard-stop.

The Kinect, laser rangefinder and odometry data is constant up until the emergency-stop command, but diverge afterwards. Due to the Segway's pitch, the laser rangefinder can detect the ground plane, which produces invalid measurements. Any errors in the Kinect calibration are amplified by extreme Segway pitches. The 20° pitch causes the Kinect to produce invalid data; however, this is not an issue as the Segway can only tilt this far in fail conditions. The odometery continues to measure the wheel rotations and (as the Segway is not attempting to balance) it is less prone to drift. Because of this, the odometery is considered the most reliable of the three measurements for determining the stop distance for the emergency-stop tests.

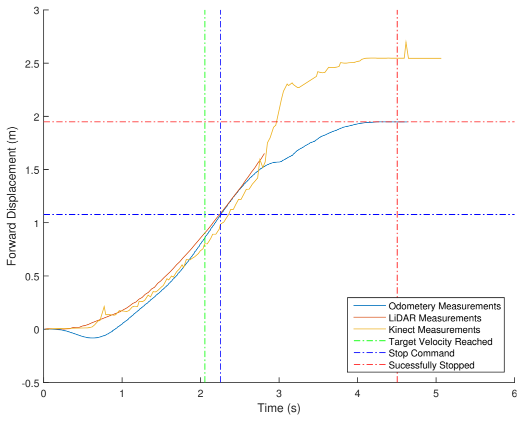
Figure 8.4 presents a typical run of the emergency-stop test at 1.4 ms^{-1}. Many of the same observations can be made as the previous tests. The data prior to the stop command is consistent with the data in the 1.4 ms^{-1} standard-stop test, as shown in figure 8.2. This includes the laser rangefinder's and Kinect's measurement errors from 0.5 s to 1.1 s. The data after the stop command is consistent with the 1.0 ms^{-1} emergency-stop test, as shown in figure 8.3; however, with larger velocity, displacement and pitch ranges.



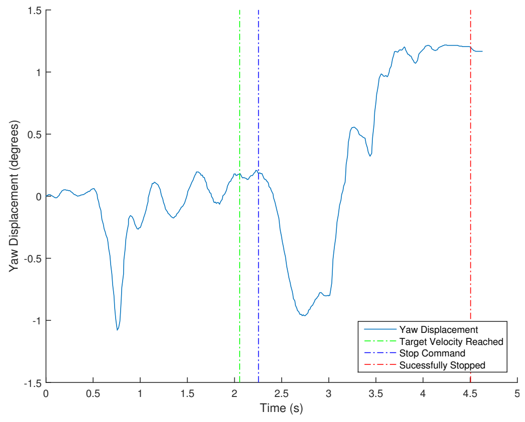
(a) Forward velocity



(b) Pitch

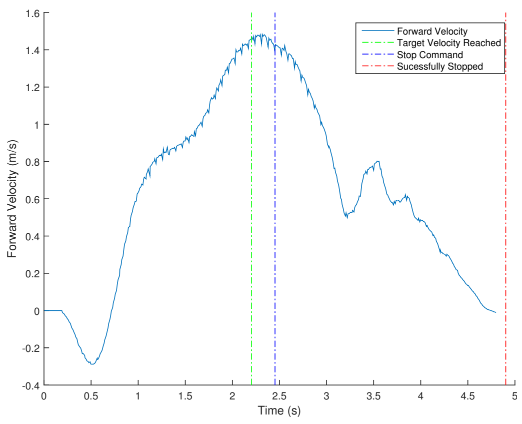


(c) Forward displacement

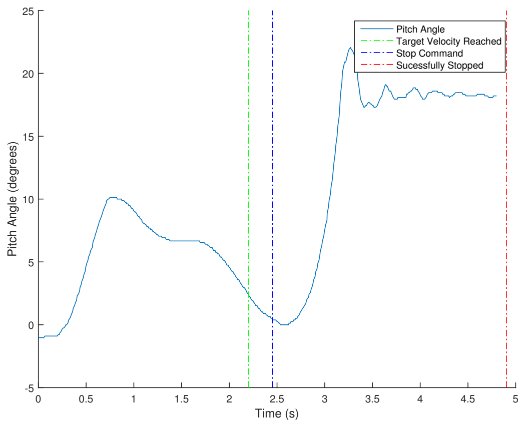


(d) Yaw

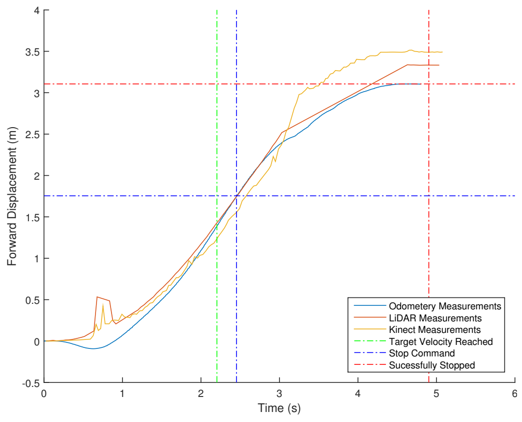
Figure 8.3: Emergency stopping distance with a target velocity of 1.0 ms^{-1}



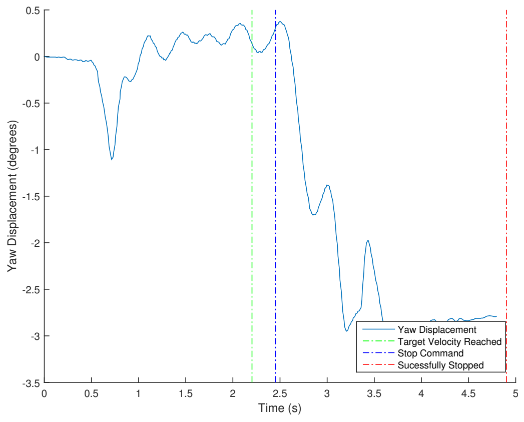
(a) Forward velocity



(b) Pitch



(c) Forward displacement



(d) Yaw

Figure 8.4: Emergency stopping distance with a target velocity of 1.4 ms^{-1}

8.1.3 Summary

Table 8.1 presents a comparison of the four stopping distance tests: standard-stop at target velocities of 1.0 ms^{-1} and 1.4 ms^{-1}, as well as emergency-stop at target velocities of 1.0 ms^{-1} and 1.4 ms^{-1}.

Table 8.1: Summary of stopping distance tests.

Test Type	**Target Velocity (ms^{-1})**	**Stopping Distance ($m \pm \sigma$)**	**Velocity Range ($ms^{-1} \pm \sigma$)**	**Pitch Range ($^\circ \pm \sigma$)**	**Pitch Range ($^\circ \pm \sigma$)**
Standard Stop	1.0	1.48 ± 0.05	-0.25 ... 1.08 $\pm$ 0.04	-9.22 ... 7.5 $\pm$ 0.54	-0.57 ... 1.06 $\pm$ 0.36
	1.4	2.08 ± 0.07	-0.36 ... 1.49 $\pm$ 0.04	-1.35 ... 10.3 $\pm$ 0.49	-0.61 ... 1.19 $\pm$ 0.43
Emergency Stop	1.0	0.76 ± 0.10	-0.25 ... 1.07 $\pm$ 0.02	-1.24 ... 21.9 $\pm$ 0.66	-1.29 ... 1.04 $\pm$ 1.13
	1.4	1.53 ± 0.26	-0.31 ... 1.48 $\pm$ 0.02	-12.44 ... 22.1 $\pm$ 0.74	-2.03 ... 2.76 $\pm$ 3.36

When comparing the standard-stop and emergency-stop results it can be seen that the emergency-stop has shorter stopping distances on average, but are typically less constant (they have larger σ). This is because MARVIN slides to a stop once the motors are turned off, and so the stopping distance is dependent on external factors (such as the friction with the ground). Because of this it is not recommended to perform an emergency-stop on sloped ground.

By comparing the laser rangefinder, Kinect and odometery data, it can be seen that the range sensors produce valid measurements in the typical operating pitch of -10° to 10°. This comparison also shows that the odometery suffers from drift, especially during periods of high acceleration or deceleration. This demonstrates the need for localisation as part of MARVIN's navigation stack.

When operating in open spaces (areas larger than 4 m $\times$ 4 m) or travelling along the centre wide corridors (2.5 m or wider), a stopping distance of 2.1 m is acceptable. If MARVIN is travelling along the centre of a 2.5 m corridor, there is $\sim$ 900 mm on either side of it. This gives enough room for a person to step around a blind corner, or out of a doorway without intercepting MARVIN's path. This suggests that the 1.4 ms^{-1} velocity limit is safe for these areas. When operating in less open spaces such as areas 3 m $\times$ 3 m (or larger) and corridors 1.4 m (or wider) a 1.5 m stopping distance is acceptable. This suggest that the 1.0 ms^{-1} velocity limit is safe for these areas. When operating in confined areas, such as cluttered offices or narrow corridors a 0.8 ms^{-1} velocity limit should be used.

Through the characterisation of the Segway RMP presented in this section, objective 1a has been met. The implementation of the discussed velocity limits meets objective 3a.

8.2 Navigation Stack

As a research platform, MARVIN must be capable of operating in environments that may vary from the test environment. Because of this, the test environment needs to be generalised to encompass all predicted operating environments. From section 1.1.1, it is known that the likely operating environments will be contained in the second floor corridors of the Alan MacDiarmid and Cotton buildings, in the Kelburn campus of VUW (or in similar office-like environments). These areas have the following generalised key features:

- Minimum measured corridor width of 1.4 m. However the minimum allowable width for a disabled access building is 1.2 m [40].
- $90°$ corridor corners.
- Minimum measured doorway width of 800 mm.
- Areas populated with unpredictable obstacles (objects that are relocated regularly), such as office furniture.

By evaluating each of these key features, it is possible to generalise MARVIN's navigation performance. This provides confidence that MARVIN can operate in any area in the predicted operating environment. Corridor traversal is evaluated in section 8.2.1, doorway navigation is discussed in 8.2.2 and unknown obstacle navigation is evaluated in 8.2.3.

8.2.1 Corridor Test

This section evaluates MARVIN's ability to traverse corridors. These tests involved manually controlling MARVIN to the starting point at one end of the corridor, then providing a high level goal at the other end of the corridor after a 90° corner. As MARVIN travels from the start pose to the goal pose, the travelled path, segway pitch, forward displacement, yaw displacement, linear velocity and angular velocity are recorded. This demonstrates both MARVIN's ability to travel down a corridor as well as its ability to turn in the corridor's limited area. Figure 8.5 shows a *rviz* screen-shot of the corridor test.

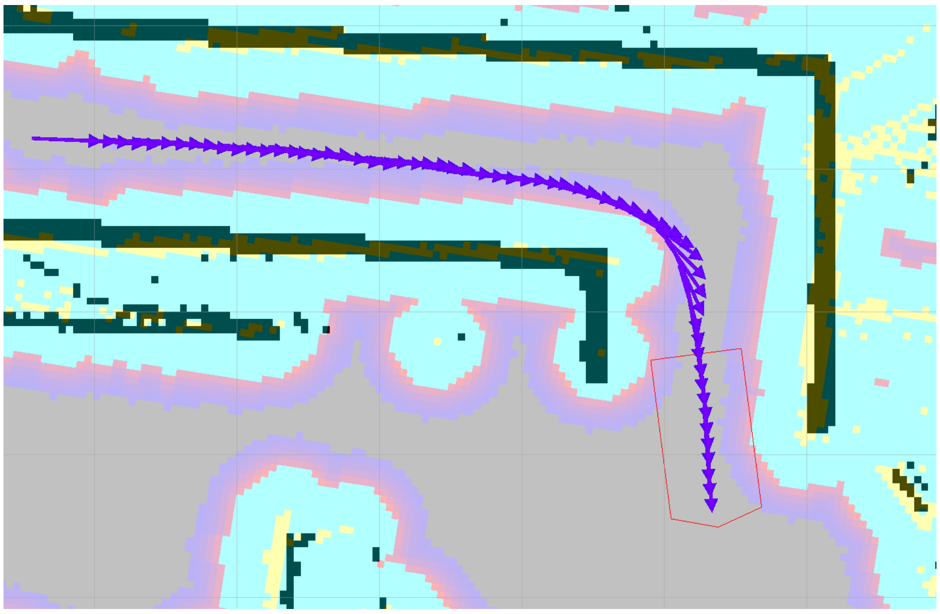


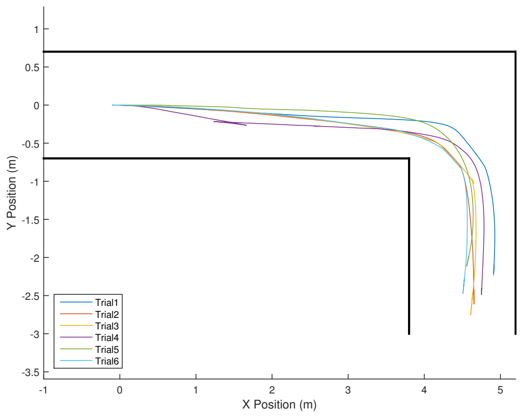
Figure 8.5: *Rviz* screen shot of a 1.4 m wide corridor test.

The black lines form the pre-built map of the test environment (generated using the SLAM techniques discussed in section 5.4), the yellow points show the detected obstacles and the blue/red areas show the inflated areas that MARVIN's motion planner knows to avoid (generated using the costmap discussed in section 5.2).

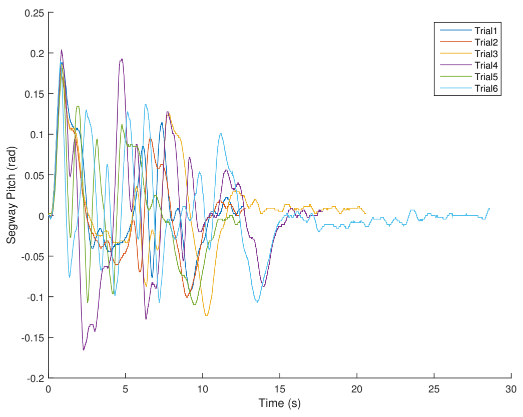
The first test used a corridor width of 1.4 m and a maximum motion planner velocity of 1.4 ms^{-1}. This test was run for 6 trials. Figure 8.6a shows the travelled path for each of the trials, figure 8.6b shows the Segway pitch, figure 8.6c shows the forward displacement (how far MARVIN travelled), figure 8.6d shows the yaw displacement (how much MARVIN turned), figure 8.6e shows the linear velocity (how quickly MARVIN travelled) and figure 8.6f shows the angular velocity (how quickly MARVIN turned).

Figure 8.6a shows that in all six trials MARVIN successfully traversed the corridor; however, figures 8.6b to 8.6f show that MARVIN's motion was not smooth. This was caused by artifacts in the Kinect's depth image. When the Segway's pitch changes rapidly (such as when the Segway accelerates) noise is generated in the depth image. This noise is not filtered by the ground and ceiling plane removal algorithm and gets translated into the produced *laser_scan* message. This introduces false obstacles into the costmap, which the motion planner attempts to avoid. This slows the Segway down, reducing its pitch and removing the noise. The motion planner interprets this as if the false obstacles have been removed, increasing the Segway's velocity and reintroducing the noise. This oscillatory behaviour results in the unsteady movement seen in figure 8.6. These trials were 6.5 m long and had a duration of 10.3 ± 1.83 s, resulting in an average velocity of 0.61 ± 0.10 ms^{-1}.

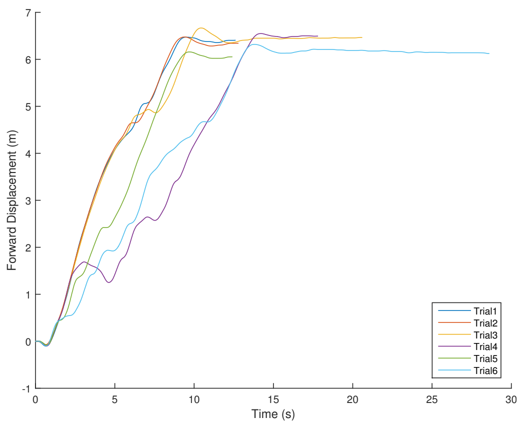
This problem is solved using the intensity based filter discussed at the end of section 4.4.1. Figure 8.7 presents the results of this test after the Kinect intensity filter was implemented.



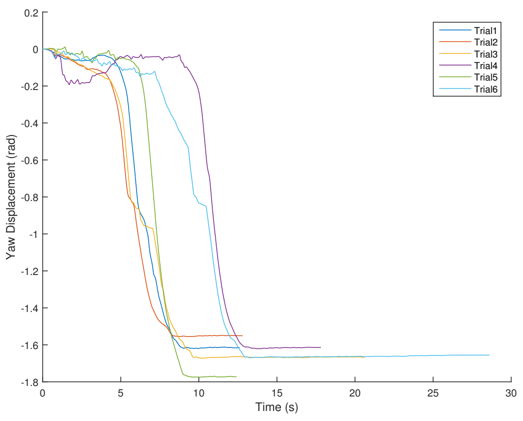
(a) Navigation Path



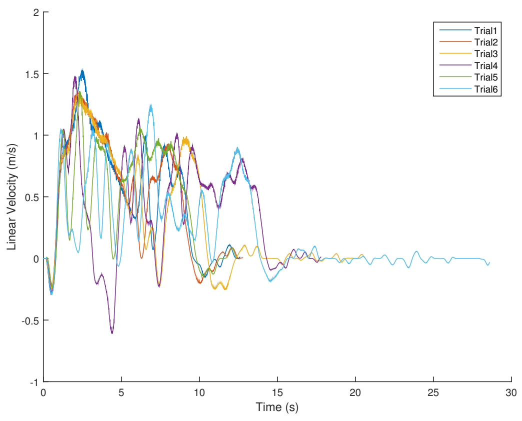
(b) Segway Pitch



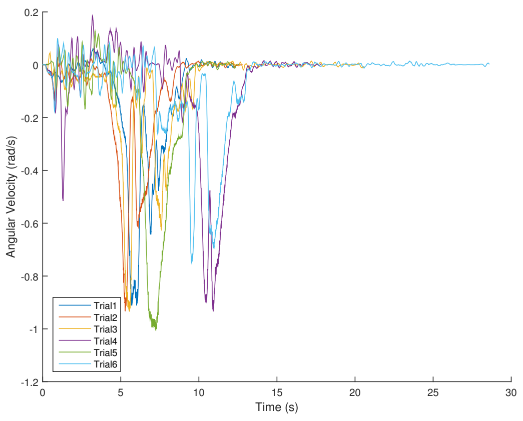
(c) Forward Displacement



(d) Yaw Displacement

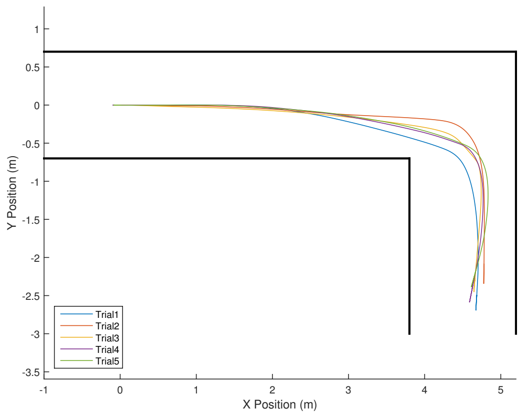


(e) Linear Velocity

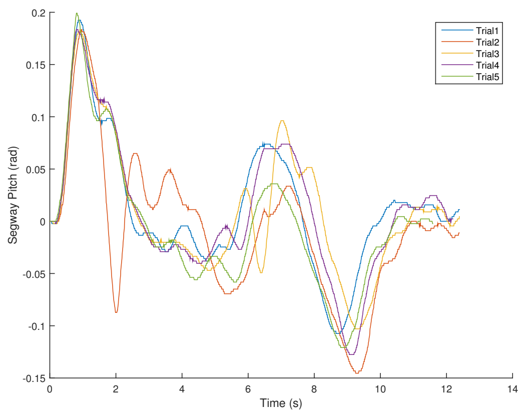


(f) Angular Velocity

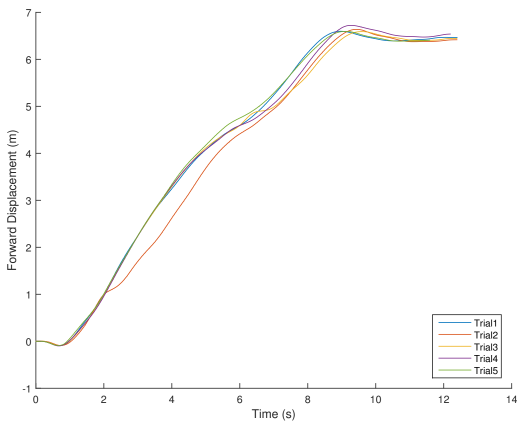
Figure 8.6: $1.4\ m$ wide corridor test with a target velocity of $1.4\ ms^{-1}$, without Kinect filter.



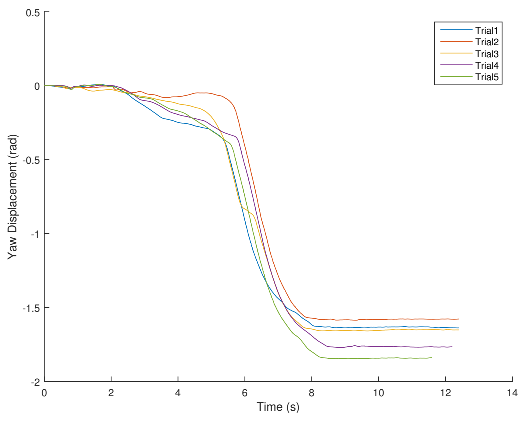
(a) Navigation Path



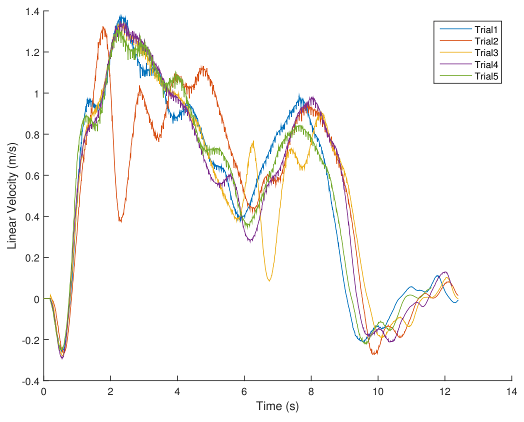
(b) Segway Pitch



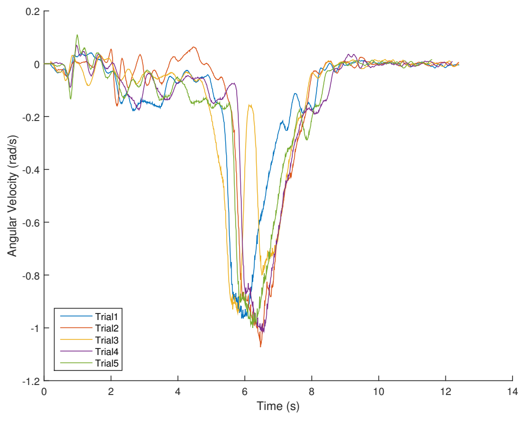
(c) Forward Displacement



(d) Yaw Displacement



(e) Linear Velocity



(f) Angular Velocity

Figure 8.7: $1.4\ m$ wide corridor test with a target velocity of $1.4\ ms^{-1}$, with Kinect filter.

Figure 8.7a shows that all 5 trials were once again successful. However, figures 8.7b to 8.7f show that MARVIN's motion was smoother and more repeatable that the previous trials. This results in a higher average velocity and faster trial times.

Figures 8.7c and 8.7e show that MARVIN's velocity increases until it reaches the 1.4 ms^{-1} velocity limit, then drops while navigating around the 90° corner. After the corner it accelerates again before decelerating to stop at the goal location. As explained in section 8.1, the Segway's pitch is dependent on its velocity. This can be seen in figures 8.7b and 8.7e as the pitch curve is an inverted form for the velocity curve. Figures 8.7e and 8.7f show that MARVIN travels in a straight line (with some minor corrections) before and after the 90° corner.

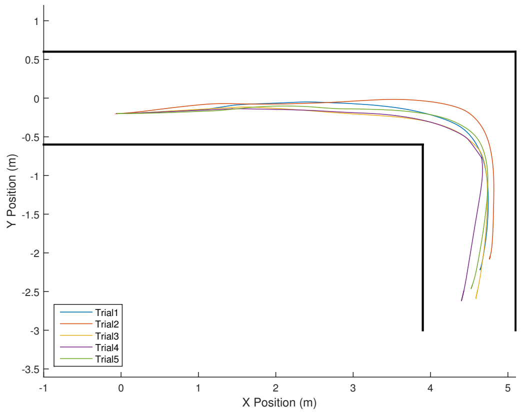
Four of the trials in figure 8.7 are consistent; however trial 2 (represented as orange) is noticeably different. At the start of this trial the Segway tilted more violently than normal, possibly due to one of the wheels getting slightly stuck. However, the Segway's controller quickly compensated for this and stabilised its balance, which resulted in a sudden velocity drop (at ~ 2.5 s). After this event, the rest of trial 2's motion is consistent with the other four trials. This shows that MARVIN is capable of robustly traversing a 1.4 m wide corridor with a maximum velocity of 1.4 ms^{-1}. It also shows that the Segway and motion planner are capable of recovering from unexpected inputs into the system. It should also be noted that there will be some slight discrepancies between the trials introduced by human error when manually controlling MARVIN to the start position (before each trial).

These trials were 6.5 m long and had a duration of 9.0 ± 0.5 s, resulting in an average velocity of 0.72 ± 0.04 ms^{-1}. This shows that the Kinect's intensity filter has increased the average trial velocity by 0.11 ms^{-1} and reduced the standard deviation by 0.06 ms^{-1}.

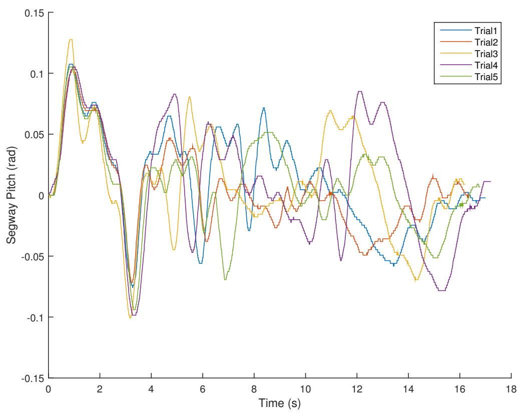
Figure 8.8 presents the results of a similar test; however, with a 1.2 m corridor and a maximum velocity of 0.8 ms^{-1}. This demonstrates that MARVIN is capable of navigating the minimum corridor allowable in the operating environment, and as such should be able to navigate all corridors it is likely to encounter in future.

Many of the same observations can be made as in the 1.4 m corridor test; however, there are two notable differences. Firstly, due to the lower velocity limit MARVIN was not required to reduce its linear velocity while traversing around the 90° corner. Secondly, MARVIN's motion was not as smooth as in the 1.4 m corridor test. This is due to the costmap's inflation leaving a narrow region of allowable space for the motion planner. This means the motion planner has to meet tighter tolerances, which requires more corrections to MARVIN's path. These corrections introduced the jerky movement. This tight tolerance would also prevent MARVIN from traversing a 1.2 m corridor if there were any obstructions along it, such as people. These trials were 6.5 m long and had a duration of 14.3 ± 1.2 s, resulting in an average velocity of 0.46 ± 0.04 ms^{-1}. This shows that MARVIN can travel through a 1.2 m corridor at an average velocity 0.26 ms^{-1} (36%) slower than a 1.4 m corridor.

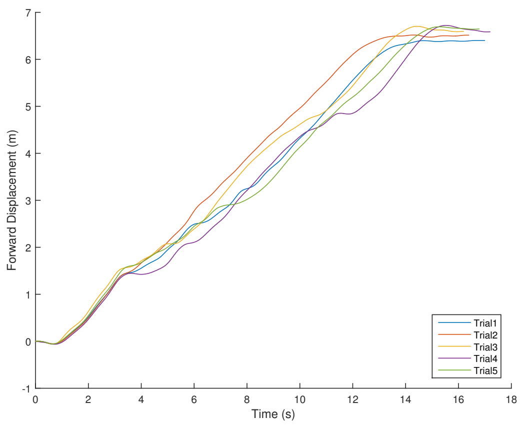
One method to improve the performance in narrow corridors would be to dynamically switch the motion planner depending on MARVIN's situation. In general the DWA approach (discussed in section 5.3.2) performs well; however, in narrow regions a wall following technique may exhibit better performance. This idea of dynamically switching motion planners was used by the *Jinny* robot [16], as discussed in 2.1.1.



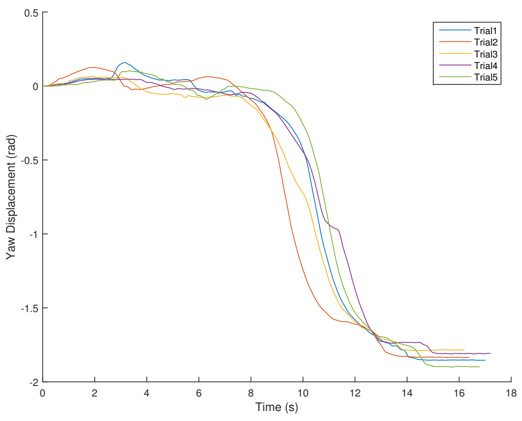
(a) Navigation Path



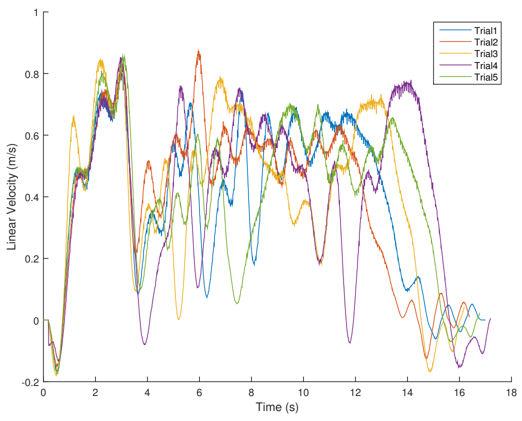
(b) Segway Pitch



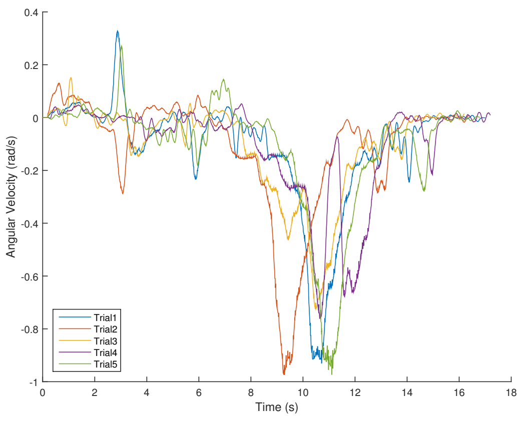
(c) Forward Displacement



(d) Yaw Displacement



(e) Linear Velocity



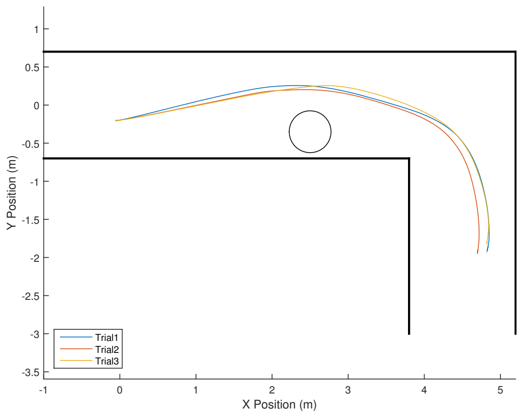
(f) Angular Velocity

Figure 8.8: $1.2\ m$ wide corridor test with a target velocity of $0.8\ ms^{-1}$.

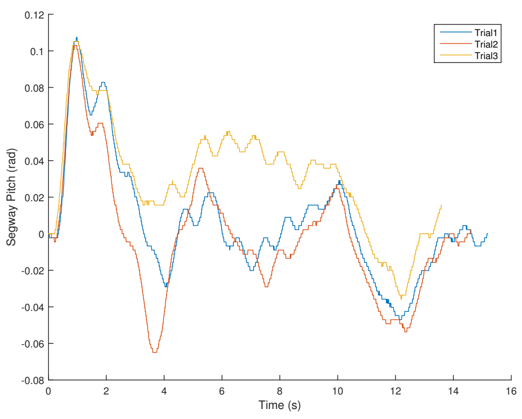
Corridors often have obstructions along them, such as: people, couches, tables or columns. To ensure MARVIN can operate in real-world situations, the navigation stack should be able to traverse a corridor with limited obstructions. This is evaluated through another 1.4 m corridor test, however with a chair obstructing part of the corridor. The chair has a diameter of ~ 600 mm, leaving a passable gap of ~ 800 mm. Figure 8.9 presents the results of this test, where the black circle represents the chair.

Figure 8.9 shows that MARVIN can smoothly navigate around the chair as it traverses down the corridor. All three trials conducted are consistent and many of the same observations from the empty 1.4 m corridor test can be made. The primary difference is that MARVIN detects the chair and immediately creates a plan to move around it. This can be seen in the smooth changes in yaw displacement, shown in figure 8.9d. During the trials the linear velocity is fairly constant, except for a slight drop as MARVIN travels around the chair (between 4 s and 4.5 s). These trials were 6 m long and had a duration of 11.2 ± 0.3 s, resulting in an average velocity of 0.54 ± 0.01 ms^{-1}.

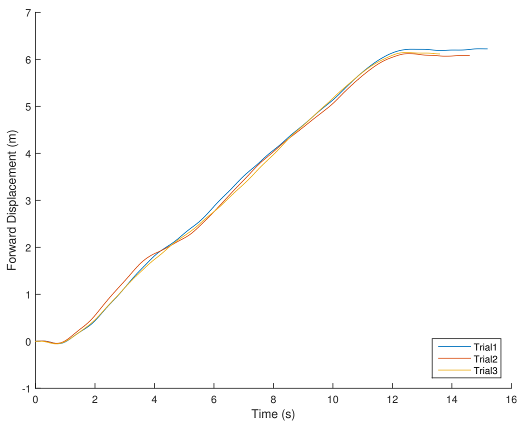
The operating environment's corridors are not guaranteed to be void of humans. As such, it is important to evaluate MARVIN's behaviour when encountering a dynamic obstacle while traversing a corridor. Therefore the previous test was repeated; however, the chair was replaced with a human. Initially, the human stood in front and to the right of MARVIN. As MARVIN approached them, they moved to the other side of the corridor. The results of this test are presented by figure 8.10. The human's original position is represented by the dotted circle, their final position is represented by the solid circle and their path is represented by the dotted line. The blue dashed line shows MARVIN's behaviour if the human does not move.



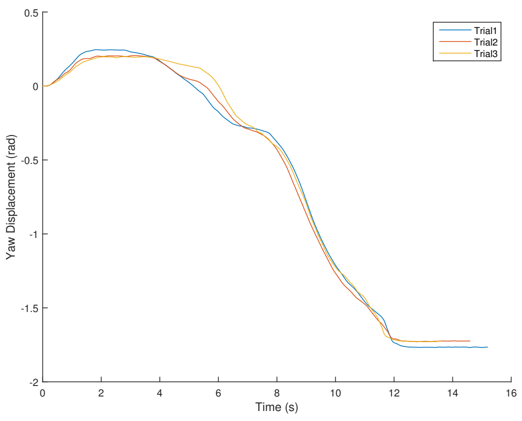
(a) Navigation Path



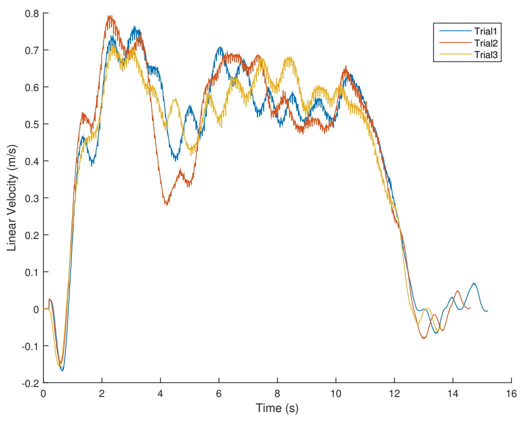
(b) Segway Pitch



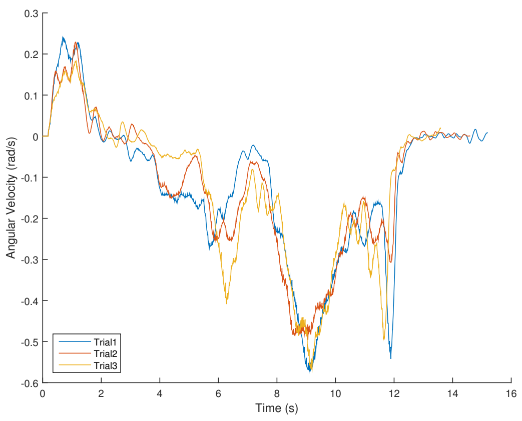
(c) Forward Displacement



(d) Yaw Displacement

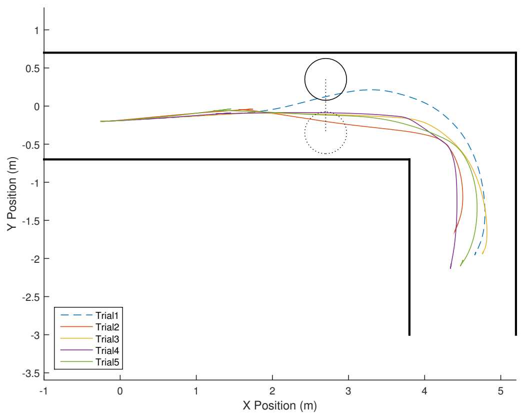


(e) Linear Velocity

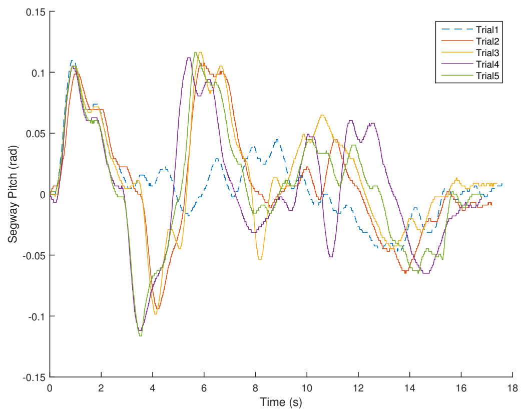


(f) Angular Velocity

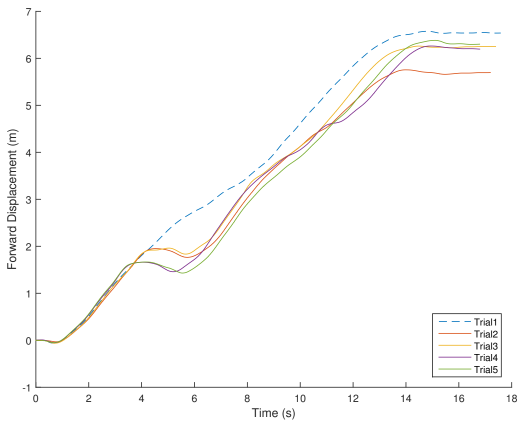
Figure 8.9: $1.4\ m$ wide corridor test with a target velocity of $0.8\ ms^{-1}$, with static obstruction.



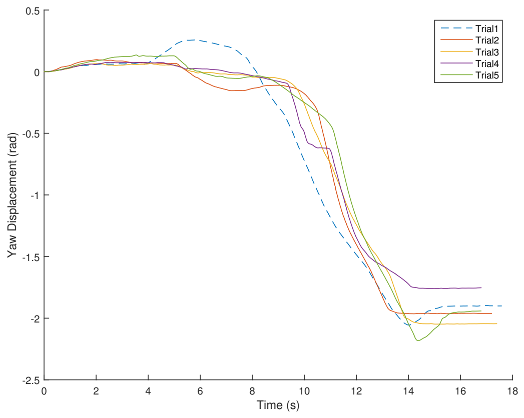
(a) Navigation Path



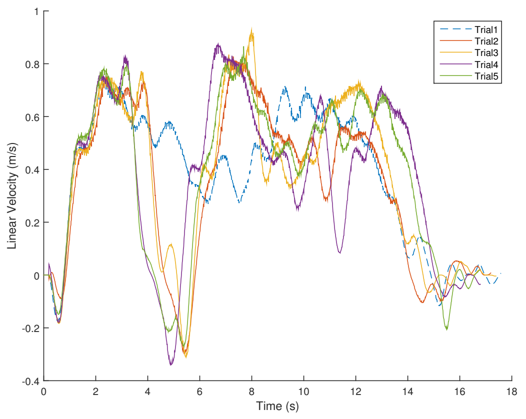
(b) Segway Pitch



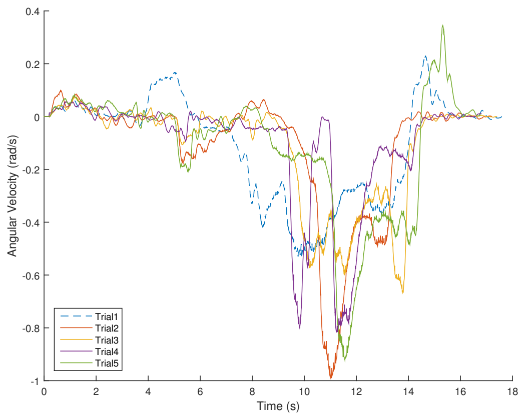
(c) Forward Displacement



(d) Yaw Displacement



(e) Linear Velocity

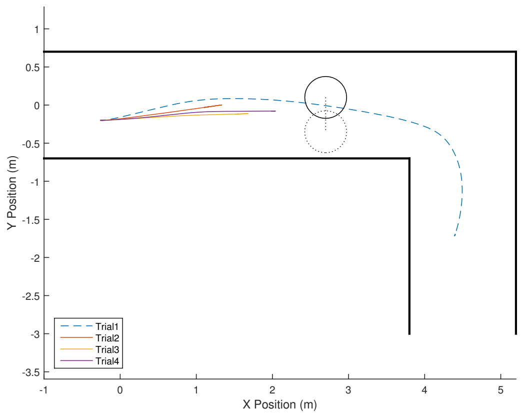


(f) Angular Velocity

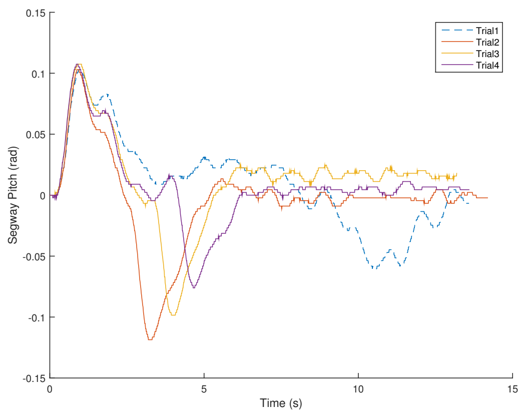
Figure 8.10: $1.4\ m$ wide corridor test with a target velocity of $0.8\ ms^{-1}$, with human moving across MARVIN's path.

As expected, this test starts similarly to the static obstacle test. MARVIN detects the human and the motion planner immediately generates a path to avoid them. However, as the human moves in front of MARVIN to get to the other side of the corridor, MARVIN stops. This can be seen in figure 8.10e, from 4 s to 6 s. Once the human is at the other side and no longer blocking MARVIN's path, the motion planner recalculates a path around the human's new position and continues to reach the goal location. Once again all three trials are consistent; however, there are some slight timing variations caused by human error when walking in front of MARVIN. These trials were 6 m long and had a duration of 14.3 ± 0.6 s, resulting in an average velocity of 0.42 ± 0.02 ms^{-1}. Compared to trial 1 (when the human remains stationary), which had a duration of 12.5 s and an average velocity of 0.48 ms^{-1}, the dynamic trials are ~ 1.8 s (12.5%) slower. This is as expected, as MARVIN has to stop to allow the human to pass in front of it.

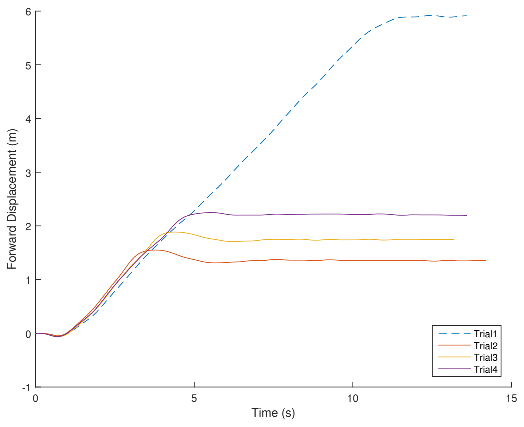
MARVIN's behaviour should also be observed when its path is fully blocked. It is important that MARVIN does not attempt move around obstacles if there is not enough room. This is tested using the same method as demonstrated in figure 8.10; however, the human stops in front of MARVIN rather than continuing to the other side of the corridor. The results of this test are presented by figure 8.11. As with figure 8.10, the human's original position is represented by the dotted circle, their final position is represented by the solid circle and their path is represented by the dotted line. The blue dashed line shows MARVIN's behaviour if the human does not move.



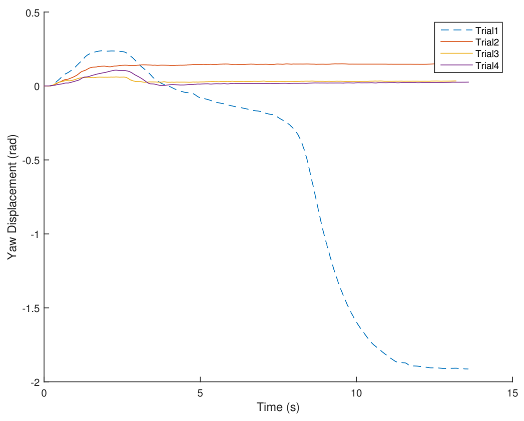
(a) Navigation Path



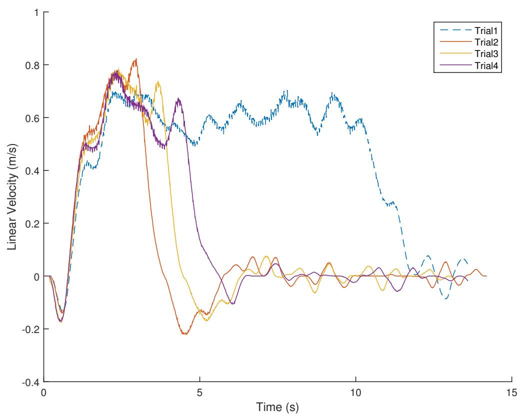
(b) Segway Pitch



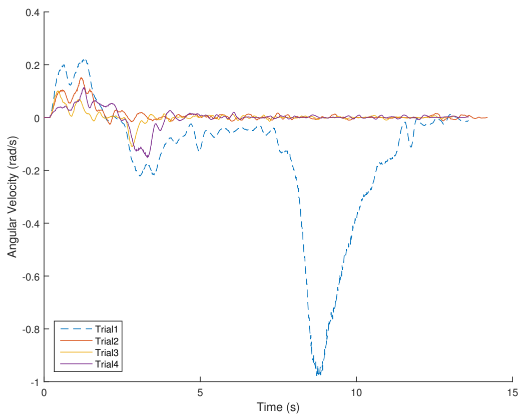
(c) Forward Displacement



(d) Yaw Displacement



(e) Linear Velocity



(f) Angular Velocity

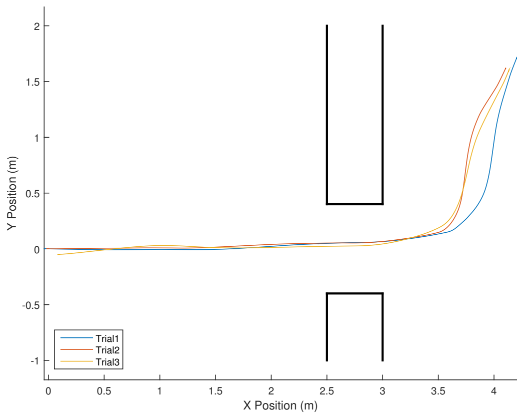
Figure 8.11: $1.4\ m$ wide corridor test with a target velocity of $0.8\ ms^{-1}$, with a human moving to block MARVIN's path.

Again, the test starts similarly to the static obstacle test. MARVIN detects the human and starts to move around them. Then when the human moves in front of MARVIN, intercepting its planned path, MARVIN stops. MARVIN makes no attempt to move around until there is a viable path to its goal location. This behaviour is referred to as ”stop and wait”. As introduced in section 2.4, it is deemed the safest behaviour when dealing with unpredictable obstacles, such as humans. During each trial MARVIN was allowed to travel ~ 0.5 m further, before the human stepped in front of it. This can be seen in figure 8.11c. This demonstrates that MARVIN's response time (including sensing and control) is sufficiently fast to have no noticeable effect on the stopping distance of the motion platform, as evaluated in section 8.1.

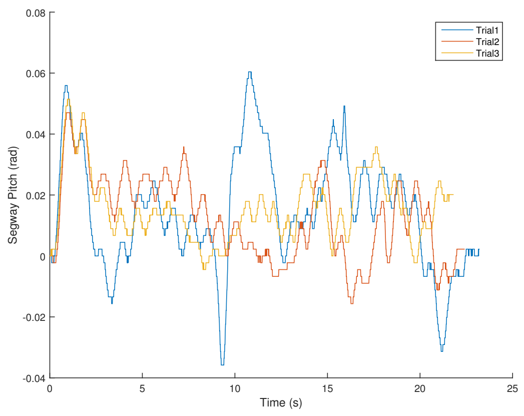
These corridor tests demonstrate that MARVIN is capable of traversing the corridors likely to be found in the operating environment, even when there are static or dynamic obstacles present. This meets the corridor navigation objective outlined in 1.2.

8.2.2 Doorway Test

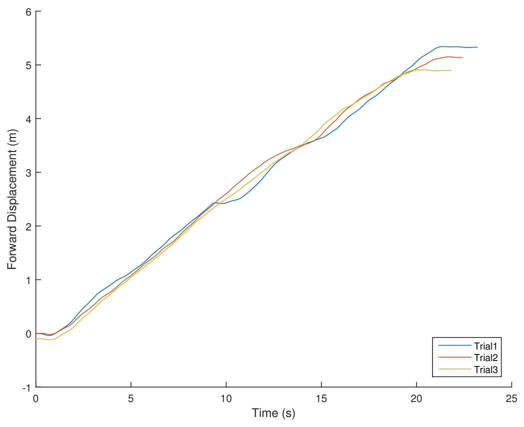
MARVIN's ability to traverse through open doorways is evaluated. All of the doorways measured in the operating environment were at least 800 mm wide, and therefore a 800 mm wide gap between two walls was created as a general case to represent the typical doors in the operating environment. MARVIN was given a high level goal on the other side of the door, after a 90° turn. The results of this test are presented in figure 8.12.



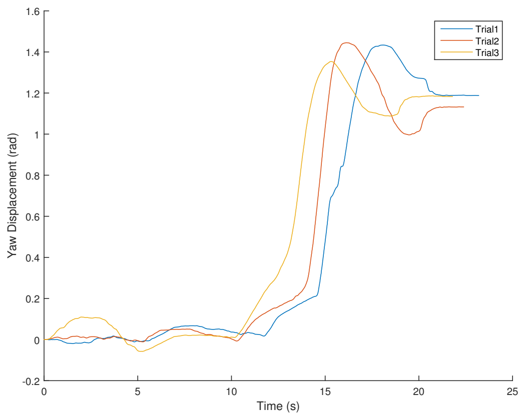
(a) Navigation Path



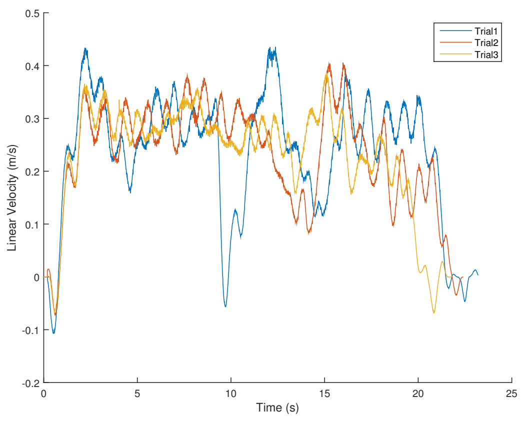
(b) Segway Pitch



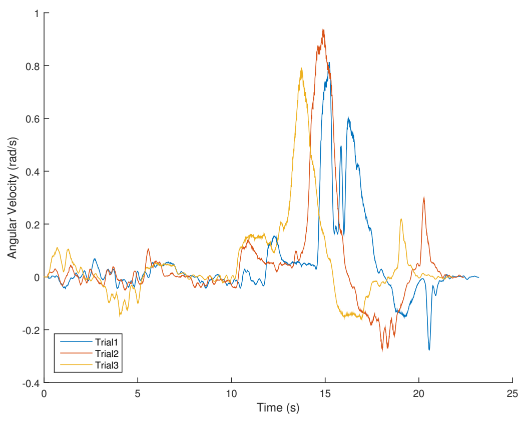
(c) Forward Displacement



(d) Yaw Displacement



(e) Linear Velocity



(f) Angular Velocity

Figure 8.12: $0.8\ m$ wide doorway test with a target velocity of 0.4 ms^{-1}.

Figure 8.12 shows that MARVIN could traverse through the 800 mm doorway, with all three trials providing consistent results. These trials were 4.7 m long and had a duration of 19.1 ± 0.2 s, resulting in an average velocity of 0.25 ± 0.03 ms^{-1}. However, this required the motion planner configuration to be tuned, specifically reducing the costmap's inflation radius and reducing the maximum velocity to 0.4 ms^{-1}. Reducing the inflation radius allows the motion planner to travel closer to detected obstacles, however, this requires lowering of the maximum velocity to maintain safe path generation (paths that do not result in collisions). In general circumstances 0.4 ms^{-1} is too slow, especially when interacting with people (the usability study in section 7.4 found that 1.0 ms^{-1} was slightly slow for most people). Additionally, MARVIN could only reliably traverse the doorway when approaching it head on.

This could be improved by using a separate motion planner developed specifically to navigate doorways, and switching to it when ever a doorway is detected (similarly to *Jinny's* dynamic motion planners as discussed in section 2.1.1). Any doorways in the map could be marked out so that the global planner knows when to switch to the doorway local planner.

8.2.3 Obstacle Course

Many of MARVIN's potential operating environments will contain unknown obstacles. These are obstacles that are not part of the known map and may be regularly moved (such as: chairs, rubbish bins and other objects commonly found in office-like environments). MARVIN must be capable of operating in an environment that contains obstacles that MARVIN has no prior knowledge of, as specified in section 1.2. To evaluate this, an obstacle course was constructed out of objects with various sizes. A photo of this course is shown by 8.13.

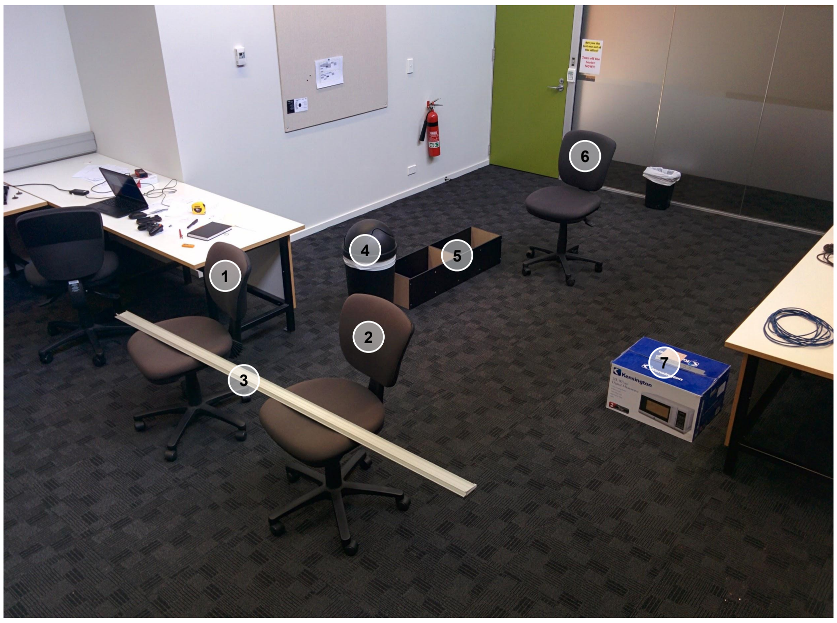


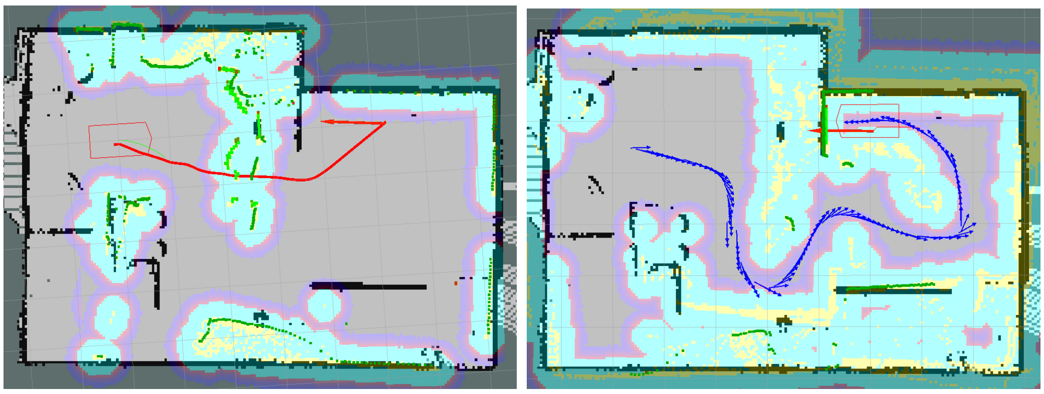
Figure 8.13: Photo of test obstacle course.

The objects used to construct the obstacle course are numbered in the figure 8.13 and classified in table 8.2.

Table 8.2: Classification of Objects in Obstacle Course

Number	Description	Dimensions
1, 2, 6	Office Chair	~ 600 mm diameter ~ 900 mm height
3	Aluminium Extrusion	60 mm × 2500 mm × 10 mm (W × L × H)
4	Rubbish Bin	~ 400 mm diameter ~ 700 mm height
5	Bookcase	300 mm × 1000 mm × 200 mm (W × L × H)
7	Cardboard Box	500 mm × 350 mm × 300 mm (W × L × H)

The selected objects cannot all be detected by all the sensing systems. For example; the laser rangefinder can only detect the chairs and rubbish bin, but not the aluminium extrusion, cardboard box or bookcase. Additionally, the obstacle course is surrounded by desks, walls and glass planes. This is used to evaluate the sensor fusion and obstacle detection (provided by the costmap discussed in section 5.2), the localisation (provided by AMCL as discussed in section 5.1) and the motion planner (provided by the local planner and global planner discussed in section 5.3). Figure 8.14 presents MARVIN's costmap before and after a typical trial (screen shots taken from *rviz*).


(a) Global Path

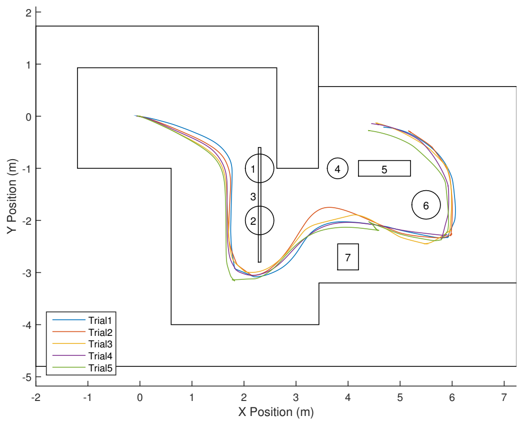
(b) Navigation after local planer corrections.

Figure 8.14: Obstacle course *rviz* screen shots.

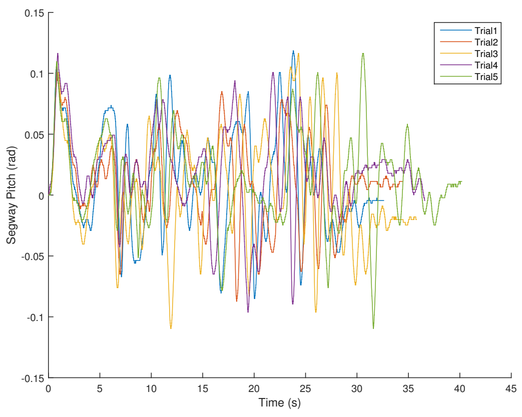
The red polygon represents MARVIN's current pose, the red arrow represents the goal pose, the red line represents the global plan, the blue arrows represents MARVIN's odometery data (showing the travelled path), the green points represents the current sensor data, the yellow points represent detected obstacles on the costmap, the blue regions show the costmap's inflation radius and the black lines represents the known map (generated prior to the test using SLAM). Figure 8.14a shows that the map only knows about the room's walls and sections of the desks. The initial global plan attempt doesn't know about the obstacles and generates a path directly to the goal. Figure 8.14b shows that as MARVIN attempts to follow the global plan, the sensors detect obstacles that are added to the costmap and the local planner attempts to avoid them. This results in a safe path (a path that does not result in collisions), as shown by the blue arrows.

Five trials were conducted for this test, with their results presented in figure 8.15. At the completion of each trial, MARVIN is manually controlled back to the starting pose. As with the corridor tests, this could introduce some slight discrepancies between each trial; however, these discrepancies are minor relative to the other factors in the trial.

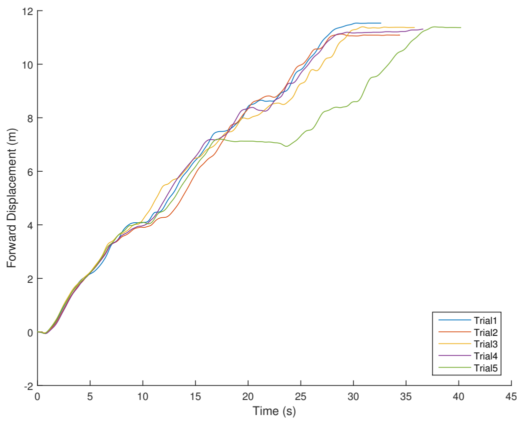
Figure 8.15a shows the paths taken by MARVIN in each trial. These paths are fairly consistent at 11 ± 0.2 m long with a duration of 29.8 ± 4.1 s, resulting in an average velocity of 0.37 ± 0.04 ms^{-1}. However, trial 5 is noticeably different from the other four. This is more clearly seen in figures 8.15c and 8.15d. This discrepancy occurred as MARVIN approached obstacle 6. As MARVIN turned round obstacle 7, its angle of approach and velocity would have resulted in a collision with obstacle 6. To prevent this MARVIN stopped, then backed slowly. Once the motion planner determined that there was enough room to navigate around obstacle 6, it resumed its journey to the goal pose. Trial 5 is consistent with the other four trials before and after this event. This shows that MARVIN is capable of recovering after detecting a dangerous trajectory, and creating a safe path to avoid the potential collision. Ignoring trial 5, the other four trials result in an average duration of 28.0 ± 0.8 s and an average velocity of 0.39 ± 0.01 ms^{-1}.



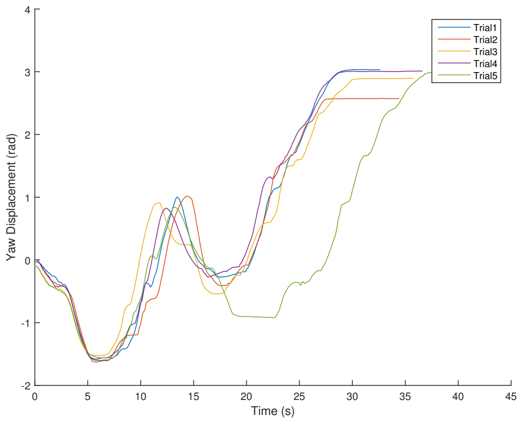
(a) Navigation Path



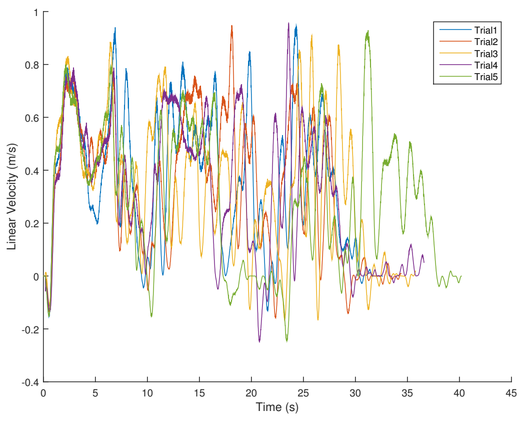
(b) Segway Pitch



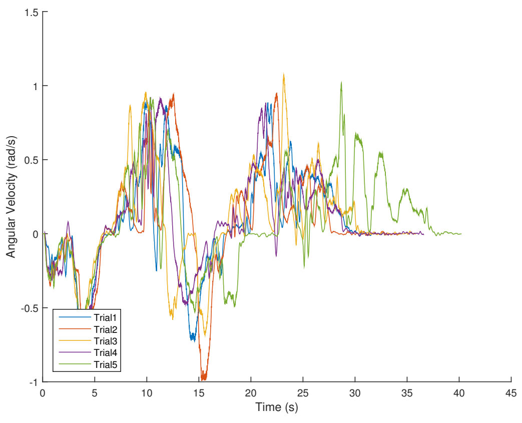
(c) Forward Displacement



(d) Yaw Displacement



(e) Linear Velocity



(f) Angular Velocity

Figure 8.15: Path taken through an unknown obstacle course with a target velocity of $0.8\ ms^{-1}$.

Figures 8.15b, 8.15e and 8.15f show that while MARVIN's responses start off consistent, after ~ 10 s the minor discrepancies introduced by operating in a real-world environment (such as: noisy sensor data, wheel slippage and differing starting poses) combine over time, causing inconsistent data. Despite this, the majority of the trials maintain a consistent path. This demonstrates MARVIN's ability to operate in real-world environments and still produce consistent results.

By successfully navigating the obstacle course, the integration of MARVIN's subsystems has been demonstrated, including the sensing systems (laser rangefinder, Kinect, whisker sensors and USN), localisation (used to maintain a correct pose estimate throughout these trials), control stack, mobility platform and motion planning. Prior to this project, MARVIN did not have the ability to detect the majority of the objects used to construct the obstacle course, nor the ability to navigate it.

8.2.4 Summary

By evaluating MARVIN's ability to traverse corridors, doorways and obstacle courses, MARVIN has been demonstrated to be capable of locomotion, sensing, localisation, obstacle detection and motion planning. As introduced in Chapter 2, these are the characteristics required of autonomous mobile robots. By this definition, MARVIN is capable of performing as a autonomous mobile robot.

The successful completion of the obstacle course demonstrates that the 3D sensor objective (1b) and sensor fusion objective (1c) have been met. The corridor traversal and obstacle course tests both demonstrate that the localisation objective (1d) and motion planning objective (1e) have been met.

8.3 Human Detection

As introduced in section 1.2, the aim of this project is to develop MARVIN into a HRI research platform. To properly interact with people, MARVIN must be capable of detecting them. This is achieved through the processing of the Kinect's depth images, as discussed in section 4.4.3. The performance of this feature is evaluated in three sections: its ability to detect a single human irrespective of their pose (section 8.3.1), its ability to detect multiple people in a single frame (section 8.3.2) and its behaviour when observing non-human objects (section 8.3.3).

As this is a binary classification problem (where all objects are considered as human or non-human) each result can be considered as one of four categories: true-positive (human detected), true-negative (non-human not detected), false-positive (non-human detected) or false-negative (human not detected). These categories will be used throughout this section.

As a comparison point, figure 8.16 presents a base-line depth image. This is a depth image of the empty test scene with no objects in frame. It has been processed by the algorithms discussed in section 4.4 (such as ground and ceiling removal).

Figure 8.16: Base line depth image for human detection analysis.

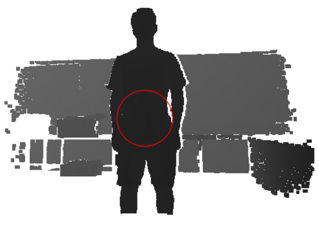
8.3.1 Human Poses

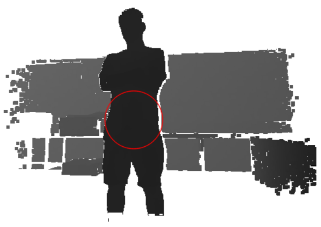
As explained by the human detection objective (2a), it is important to detect humans irrespective of their pose. This has been achieved through the implementation of *OpenCV's* blob detection algorithm, as discussed in section 4.4.3. Because the human detection algorithm detects blobs as opposed to specific shapes, the limbs of the human become irrelevant. Instead the torso is detected by configuring the blob detector to only detect blobs of the right size, shape and depth value. This allows the human detection algorithm to detect humans in almost any pose, as demonstrated in figure 8.17.

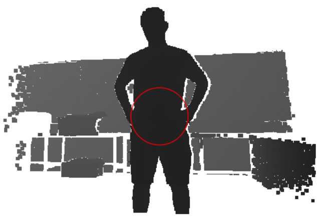
As with the other depth images in this thesis, darker pixels represent closer distances and lighter pixels represent further distances. The red circles represent the detected humans (if any). The circles are centred on the blobs classified as humans and have a diameter proportional to the blobs' areas.

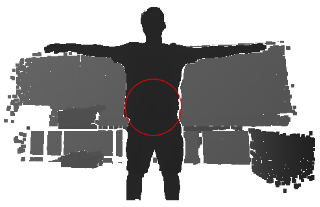
The only poses found to generate false-negatives are crouching, bending-over or squatting, as demonstrated in figure 8.17l. This is caused by the human's blob having an inertia ratio that is too high (it is too circular and not elliptical enough). However, this is unlikely to cause any issues during interaction. People will typically be standing when interacting with MARVIN and the crouching human is still detected as an obstacle.

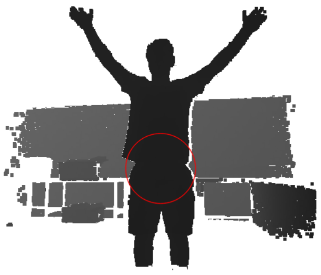
All the other tested poses returned true-positive results. This included the human sitting on a chair (figure 8.17i), which is important for any users that may be wheel chair bound. It should be noted that the sitting human was detected irrespective of the angle they were facing. This is demonstrated by figure 8.18.

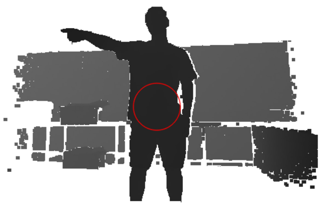

(a) Arms by sides

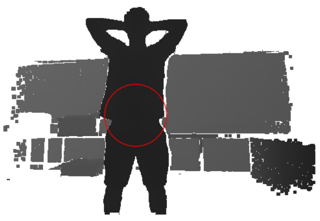

(b) Arms crossed

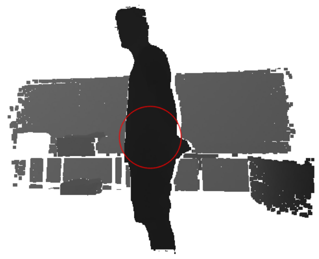

(c) Hands on hips

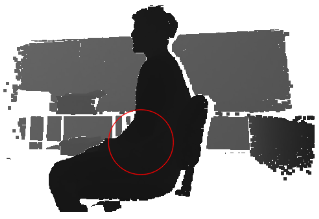

(d) Arms out

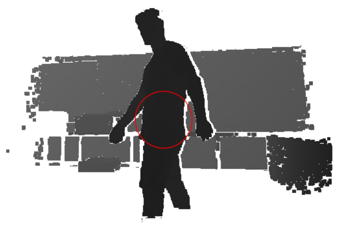

(e) Arms Up

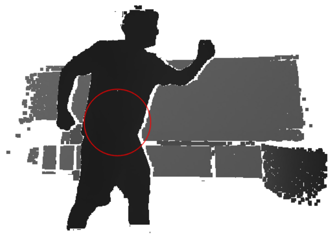

(f) Pointing


(g) Hands on head


(h) Side on


(i) Sitting


(j) Walking


(k) Running

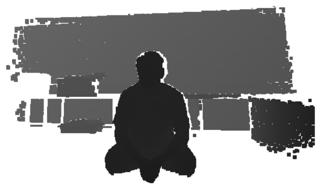

(l) Crouching

Figure 8.17: Human Detection Common Human Poses

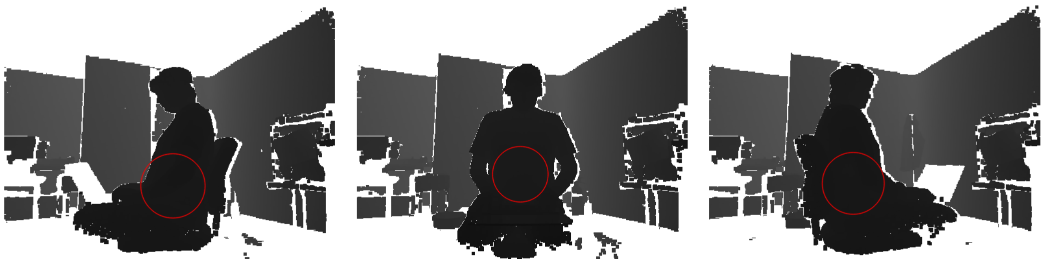
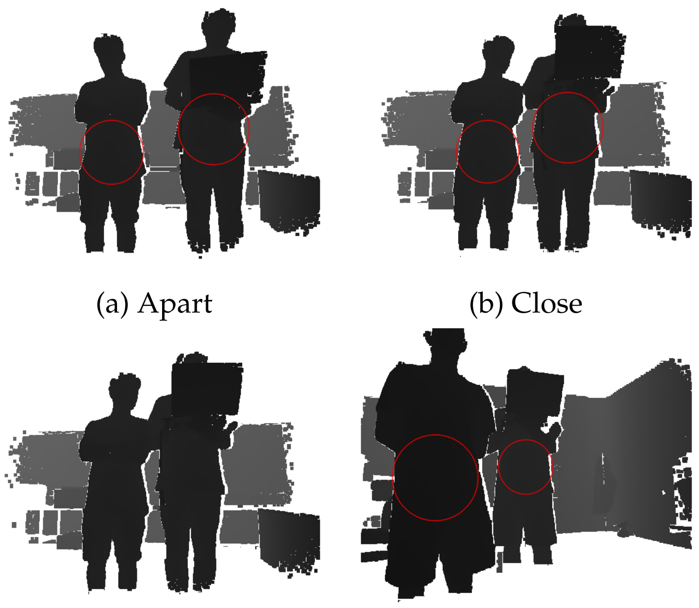

(a) Facing Left (b) Facing Front (c) Facing Right

Figure 8.18: Human Detection Sitting Poses

8.3.2 Multiple People

During HRI tasks, there will likely be multiple humans standing in close proximity together. MARVIN's ability to distinguish between these humans is evaluated in this section. Figure 8.19 shows three frames as two people stand progressively closer to one another.

MARVIN is capable of detecting both humans in figure 8.19a, 8.19b and 8.19d; however, figure 8.19c produces a false-negative. This demonstrates that MARVIN can individually detect multiple people unless they are in contact with each other. People typically stand apart from each other (the average human's personal space can be estimated as a circle with a 1.21 m radius [89]), so this problem should be infrequent. However, this issue could potentially be improved by using the RGB data from the Kinect in addition to the depth image human detection algorithm. This could be used to segment the depth image by colour (which could separate multiple people if they were wearing different coloured clothing), or for face tracking (such as the skin colour based face detection algorithm described by [90]). Face tracking could be further improved by installing additional RGB cameras at head height, to improve the robustness of the face detection algorithm.



(a) Apart (b) Close

(c) In Contact (d) Behind

Figure 8.19: Human Detection Multiple People

8.3.3 Non-Human Objects

It is important that MARVIN does not falsely classify non-human objects as humans, otherwise it could attempt to interact with inanimate objects. Figure 8.20 presents a subset of objects resulting in true-negative classifications and figure 8.21 presents the objects with false-positive classifications.

Figure 8.20a, shows that office chairs are correctly classified as non-human. This is because the seat and the back of the chair get separated into individual blobs. This occurs during the image segmentation step of the human detection algorithm, further explained in section 4.4.3. Each blob has an inertia ratio that is too high (they are too circular) and an area that is too small to be considered human. This can be generalised to say that objects with relatively complex and separable shapes (shapes that contain segments connected by narrow regions) typically result in true-negative classifications. This includes objects such as: chairs, stools and tables.

(a) Chair (b) 1 m×1 m flat panel (c) Vertical Pole

Figure 8.20: Human Detection True-Negatives

Figure 8.20b, shows that large flat surfaces are also classified as non-human. They have a high inertia ratio and a large area. This includes other large flat surfaces, such as walls.

Figure 8.20c, shows that objects with low inertia ratios (not circular enough) result in true-negative classifications. This includes objects such as: poles, railings, cables and rubbish bins.

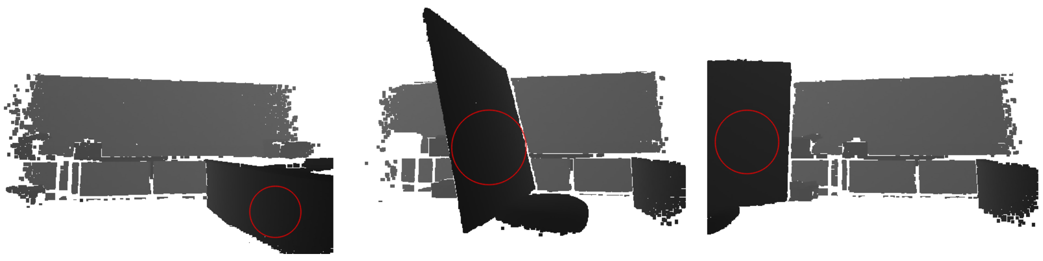

(a) Desk (b) Angled panel (c) Panel half in frame

Figure 8.21: Human Detection False-Positives

By angling flat surfaces away from the Kinect, their inertia ratio lowers and can result in false-positive classifications. This is shown by figures 8.21a and 8.21b. This issue also occurs if the flat surface is only partially in frame, as shown by figure 8.21c.

The human detection algorithm has a low rate of false-negatives making it reliable at detecting humans in a range of different poses, thus meeting objective 2a. However, it suffers from a relatively high false-positive rate. To improve MARVIN's effectiveness during HRI tasks, this false-positive rate should be reduced. This could be accomplished in a number of different ways, including:

Segmenting the input image into depth regions. By segmenting the image into depth regions, angled surfaces such as those in figures 8.21a and 8.21b will be broken into separate blobs. Each individual blob will have inertia ratios too low to be considered as human, thus preventing their false-positive classifications. This method has the benefit that it would be computationally efficient; however, it has two major issues. Firstly, it does nothing to prevent the false-positive classifications of narrower flat surfaces parallel with the Kinect, such as figure 8.21c. Secondly, depending on how the image is segmented, humans may be segmented at certain distances causing false-negative classifications. For example, if a human was standing at a distance from MARVIN that coincided with a border between two depth regions, the human will get segmented into two blobs, one blob containing the closer pixels and the other containing the further pixels.

Filtering the detected blobs by standard deviation (σ). Humans typically have uneven surfaces, unlike the surfaces demonstrated in figure 8.21. If the blobs are filtered by a surface roughness metric, it may be possible to reduce the false-positive rate without negatively affecting the true-positive rate. This could be achieved by measuring the σ of each blob. The surfaces in figures 8.21a and 8.21b are angled away from the Kinect, giving them a wide range of depth values. This would give them a relatively high σ. The flat surface in figure 8.21c is parallel with the Kinect and has no notable features. This causes it to have a low range of depth values, giving it a low σ. Humans are mostly flat, but have some uneven features. This would give them a σ between the angled and parallel surfaces. This provides an additional metric to filter the detected blobs, reducing the false-positive rate for all the examples demonstrated in figure 8.21, without reducing the true-positive rates. This method would be more computationally expensive than the segmentation method. However, *OpenCV* contains many efficient functions for standard image processing methods (such as mean and standard deviations through *cv::meanStdDev*), so this cost should be minimal.

Another extension to the human detection algorithm would be to improve the range at which humans can be reliably detected. As explained in section 4.4.3, the current detection range is 2 m. This is because the blob detection algorithm classifies humans using area as one of its metrics. If a human is closer to the Kinect, they will take up more of the Kinect's FOV. This increases the number of pixels associated with them, increasing their blob's area. This means that having a static area threshold limits the range in which humans can be correctly classified. If a dynamic area threshold is used (the further the depth, the smaller the area threshold), it may be possible to detect humans over a larger range.

These extensions to the human detection algorithm require further development and testing. They should be considered for future work, as discussed in section 9.2.

8.4 Summary

This chapter has presented the results and analysis of MARVIN's mobility platform, navigation stack and human detection algorithm. Through the characterisation of the Segway RMP, objectives 1a and 3a are met. The evaluation of the navigation stack addresses the 3D sensor objective (1b), sensor fusion objective (1c), localisation objective (1d) and motion planning objective (1e). The analysis of the human detection algorithm demonstrates the completion of objective 2a. With objectives 2b, 2c and 2d addressed by the usability study (section 7.4), and objectives 3b and 3c addressed in sections 3.2.1 and 6.1, all the objectives detailed in 1.2.4 have been met.

With the objectives met, Chapter 9 can conclude the thesis. It reviews the chapters, provides a summary and discusses potential future work.

Chapter 9

Conclusions

This chapter reviews the completed work, outlines options for future work and concludes by comparing MARVIN's capabilities against the original objectives detailed in section 1.2.

9.1 Review

Chapter 1 introduced the motivation to transform the exiting MARVIN system into a research platform for human-robot interaction. Through the background research presented in Chapter 2, a list of required functionality was determined, including: locomotion, sensing, localisation, motion planning, receptive elements and expressive elements. This thesis presents the design and development of this functionality, enabling MARVIN to operate fully autonomously and interact with humans.

MARVIN is capable of locomotion through the inherited Segway RMP. This provides an agile mobility platform with fast acceleration/deceleration profiles, high top speeds and the ability to maintain its balance. This makes the Segway more versatile than standard differential drive systems, allowing it to climb over small obstacles, traverse steeper slopes and recover from external forces (such as getting pushed). However, this self balancing functionality introduces a dynamic pitch that increases the complexity when processing MARVIN's sensing systems.

MARVIN inherited a laser rangefinder and ultrasonic network for the detection of opaque and transparent obstacles. However, both sensing systems have planar measurement areas and would fail to detect the majority of obstacles in an office-like environment. To overcome this, two additional sensing systems are implemented. First, Microsoft's Kinect v2 RGB-D camera is introduced as a 3D sensor. Second, four custom built whisker sensors are mounted on each of MARVIN's corners, providing tactile feedback to MARVIN's control systems. However, to interface with these new sensors a newer control computer was required. An Intel NUC was integrated into MARVIN due to its compact size and high performance hardware. ROS is installed on the NUC, allowing the development of MARVIN's control software.

The range sensors' data needed to be processed into a coherent form that MARVIN's navigation stack can understand. The ultrasonic network and laser rangefinder are interfaced with ROS through *laserScan* messages, which store the sensor data in polar coordinates. This is the standard method used by the majority of mobile robots developed using ROS; however, the Segway's dynamic pitch introduced two challenges. First, the origin of the sensors are dependent on the Segway's pitch. This causes all of their range measurements to be offset. Second, the angle of the sensors is depended on the Segway's pitch. This introduces parallax errors into the range measurements. These errors are accounted for by dynamically recalculating the sensors' transforms and projecting the sensors range measurements to the horizontal plane each time the Segway's pitch is updated. The Kinect stores its range data in depth images. These depth images are converted into *laserScan* messages so they can be fused with the laser rangefinder's and ultrasonic network's measurements. To do this, the ground and ceiling planes must be removed from the depth images. Once again, the Segway's dynamic pitch increases the complexity of this process. As the Segway's pitch changes, the relative angles of the ground and ceiling planes also change. To overcome these challenges, a novel algorithm is developed to process the depth images directly, as opposed to converting the images in 3D coordinates (point-clouds). This removes the need for complex 3D equations, reducing computational overhead. Once the ground and ceiling planes are removed, the depth images are further processed to produce *laserScan* messages and to detect humans. The *laserScan* messages are further processed by an intensity filter to reduce the effects of any noise that the depth image processing failed to remove. Humans are detected using *OpenCV's* blob detector algorithm. The detection of humans has a very low false-negative rate, but suffers from a higher false-positive rate. Possible solutions are provided for this problem in section 8.3.

The processed sensor data is used by MARVIN's navigation stack for localisation, obstacle detection and motion planning. Localisation is achieved using the adaptive Monte Carlo localisation technique, which makes use of particle filters. Obstacle detection involves the fusion of the range sensors' data into a local grid-based map, called a costmap. The detected obstacles are then inflated to account for MARVIN's physical dimensions. The costmap is used in-conjunction with a global map of the environment (generated using SLAM techniques) by MARVIN's motion planner. The motion planner consists of a global planner and a local planner. The global planner takes MARVIN's location estimate and generates a high level path to a goal location. The local planner attempts to follow the global planner's path while accounting for the detected obstacles in the costmap. It produces safe velocity commands (commands that don't result in collisions) that are sent to the Segway RMP via MARVIN's control systems.

MARVIN makes use of a hierarchical control architecture. A central control node, *marvin_control*, is used to monitor and control the lower level nodes. *marvin_control* takes requests from the user (either from the interface tablet or from the XBox controller), and distills them into commands for the lower level nodes. These lower level nodes include, the motion controller (*move_-base*), *interaction_control* and *movement_control*. The *interaction_control* node provides the high level control of MARVIN's robotic humanoid torso. It uses the relative location of the detected human to interactively change the torso's and Segway's pose, reacting as the human moves. The *movement_-control* processes the Segway's velocity commands. These commands can come from three sources: from the Xbox controller (via the *manual_control* node), the *move_base* node or the *interaction_control*. The *movement_control* ensures that commands are only executed from one source at a time and disables the Segway if it detects any unauthorised commands. It also provides reactive control, stopping the Segway if the whisker sensors detect a collision.

These systems are fully integrated to enable human-robot interaction. MARVIN makes use of a Surface 3 Pro as the primary interface between itself and humans. This provides an LCD touch screen as well as voice synthesis and recognition. MARVIN was implemented as a robotic guide to evaluate its effectiveness as a HRI research platform. This robotic guide implementation was evaluated with a usability study, which resulted in largely positive responses.

9.2 Future Work

Throughout this thesis, suggestions have been made for possible avenues of future research and development. This section provides an summary of these suggestions.

Section 3.2.1 discussed the selection of the Kinect as MARVIN's RGB-D camera; however, it suggested that Intel RealSense RGB-D cameras could be investigated in future. Due to their compact size, low power consumption ($\sim 10\%$ of the Kinect) and lower cost (99 USD in comparison to the Kinect's 150 USD or the SICK LMS100's 5000 USD MSRP), it could be feasible to use multiple RealSenses. As RealSenses use IR projectors to measure depth, there is a possibility that they may interfere with one another if their FOV's overlap [91]. To avoid this issue, a possible sensor layout would be to have one RGB-D camera on the front, right, left and back of MARVIN. The front RGB-D camera could be the existing Kinect to maintain its larger FOV. The additional RGB-D cameras would allow MARVIN to have better awareness of the obstacles around it, providing it with more flexibility when generating safe paths. It would also give a greater area for detecting humans, for example the rear RGB-D camera could be used to monitor if the person MARVIN is guiding is still following. The additional data produced by the extra RGB-D cameras will likely exceed the processing capabilities of the existing control computer. One possible solution would be to make use of multiple NUC's, one for the high level control of MARVIN (including the navigation stack) and the other dedicated to sensor processing.

First introduced in section 2.1.1, an adaptive navigation stack could be implemented on MARVIN, which would change its motion planner according to the conditions of MARVIN's current environment. The dynamic window approach implemented is effective as a general motion planner; however, in specific situations other approaches may be more appropriate. The motivation for this was demonstrated in the narrow corridor test (section 8.2.1) and doorway test (section 8.2.2).

The usability study presented in section 7.4 demonstrated that the voice recognition could be unreliable for some participants. The current implementation uses Microsoft's .NET speech platform to process the audio signals from Surface 3 Pro's in-built microphone to recognise key words.

Both the hardware and software of this approach could be improved. The in-built microphone could be replaced with a microphone array designed to filter background noise (ambient noise was measured at 42.0 ± 0.5 dB, which increased to 52.2 ± 2.5 dB when the torso was actuated). This should help reduce the number of false-positive key word detections. The software could be replaced with Nuance's Dragon NaturallySpeaking, which is generally considered one of the better voice recognition packages [92, 93]. Another option would be to make use of one of the server based voice recognition approaches, such as SoundHound's Houndify API [94] or *Api.ai* [95]. These approaches send the processed audio signal to an external server to be recognised. This requires a constant internet connection and produces slower results; however, they are also typically much more powerful and easier to integrate with the rest of the software.

While the human detection algorithm was found to work effectively and met the human detection objective (2a), sections 4.4.3 and 8.3 suggest some possible extensions. These extensions include: segmenting the input image into depth regions, filtering the detected blobs by standard deviation, using a dynamic area threshold to detect humans over a longer range, replacing the blob detector with a trained classifier, making use of the RGB data to augment the depth data (for separating close humans or detecting faces), or by using depth images to detect humans' poses and hand gestures.

With the successful development of MARVIN, the possible tasks outlined in section 1.1.2 could be explored. These included being a: robotic guide for the University Library, robotic guide for new students or visitors, an information point for students or as a security guard.

9.3 Summary

This project has successfully transformed MARVIN into a research platform for human-robot interaction, meeting all the objectives detailed in section 1.2.

The Segway was characterised to determine appropriate velocity limits in section 8.1 (objective 1a). Kinect v2 was implemented as a 3D sensor with a novel method developed for removing the ground and ceiling planes from its depth images (objective 1b). A costmap was implemented to fuse the laser rangefinder, ultrasonic network, whisker sensor and Kinect data streams for obstacle detection (objective 1c). The adaptive Monte Carlo localisation technique was implemented for the successful localisation of MARVIN in unmodified office-like environments. Dijkstra's search algorithm was implemented to find high level plans through a map of MARVIN's environment (generated using the Gmapping SLAM technique) and the adaptive window approach is used to produce local plans (objective 1e). This meets all of the autonomous navigation objectives.

The Kinect's depth images are processed to successfully meet the human detection algorithm with a number of possible extensions suggested (objective 2a). The robotic humanoid torso is controlled to interactively change its pose according to the detected human's relative position (objective 2b). The Surface 3 Pro is utilised to provide receptive and expressive elements for MARVIN, including: a GUI, voice synthesis and voice recognition (objectives 2c and 2d). This meets all the human-robot interaction objectives.

The Segway's characteristics are used to determine velocity limits for open spaces, regular spaces and confined spaces (objective 3a). MARVIN utilises laser rangefinders, ultrasonic sensors, Kinect and whisker sensors all of which can detect humans, providing sensor redundancy (objective 3b). A hierarchical control architecture was developed which monitors and controls all aspects of MARVIN, including reactive control of the Segway RMP (objective 3c). This meets all the health and safety objectives.

MARVIN was successfully implemented as a robotic guide, demonstrating MARVIN's effectiveness as a HRI platform. Through this project MARVIN now has the locomotion, sensing, localisation, motion planning, receptive elements and expressive elements to facilitate both autonomous navigation and HRI. These developed features open a diverse range of research directions and HRI tasks that MARVIN can be used to explore.

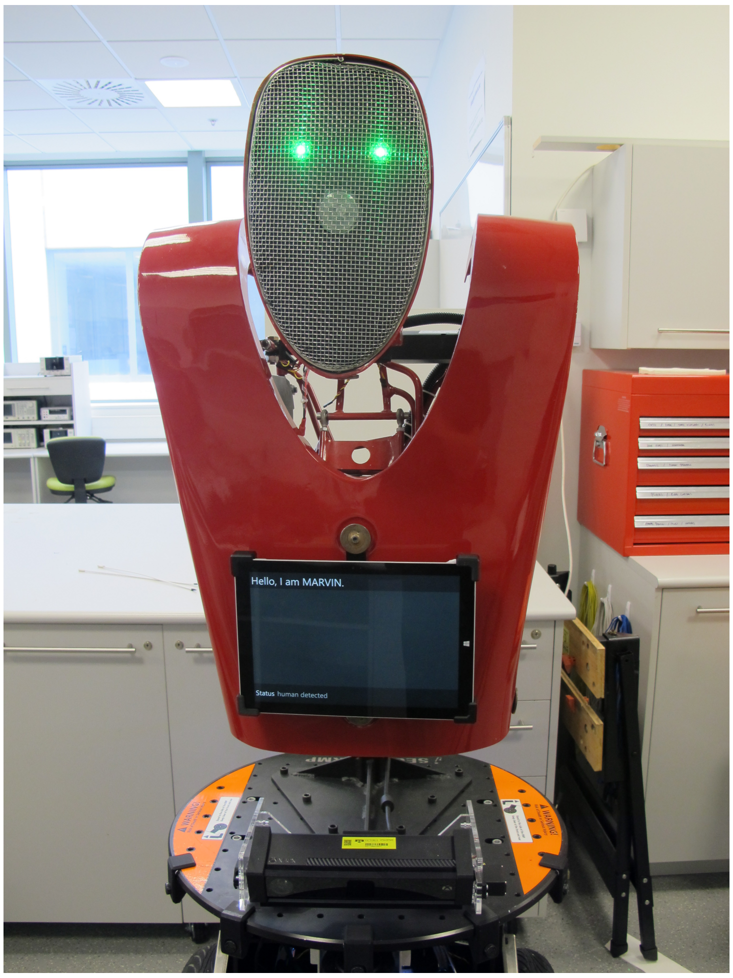


Figure 9.1: Final figure of MARVIN.

Bibliography

[1] CORAL Research Group, "Cobots." http://www.cs.cmu.edu/~coral/projects/cobot/, 2010 – 2014. [Online; accessed 17-Feburary-2016].

[2] Segway Inc., "RMP 200." http://www.segway.co.nz/business/products-solutions/robotic-mobility-platform.html. [Online; accessed 22-February-2015].

[3] C. Robinson and D. A. Carnegie, "Modifying an indoor mobile robot to enhance autonomous operation," in *Proceedings of the 21st Electronics New Zealand Conference*, pp. 14 – 19, ENZCon, 2014.

[4] E. Marder-Eppstein, D. V. Lu, and D. Hershberge, "ROS Costmap 2D Package." http://wiki.ros.org/costmap_2d, 2015. [Online; accessed 30-January-2016].

[5] D. Lu, "ROS global planner package." http://wiki.ros.org/global_planner, 2014. [Online; accessed 31-January-2016].

[6] Intel, "NUC D54250WYKH Specifications." http://www.intel.com/content/www/us/en/nuc/nuc-kit-d54250wykh.html. [Online; accessed 22-February-2015].

[7] C. P. Lee-Johnson, "The development of a control system for an autonomous mobile robot," Master's thesis, University of Waikato, 2004.

[8] Rethink Robotics, "Baxter." `http://www.rethinkrobotics.com/baxter/`. [Online; accessed 25-February-2016].

[9] Willow Garage, "PR2." `https://www.willowgarage.com/pages/pr2/overview`. [Online; accessed 25-February-2016].

[10] O. for Economic Co-operation and Development, *Society at a Glance 2009: OECD Social Indicators*. OECD Publishing, 2008.

[11] Adept Robotics, "Pioneer." `http://www.mobilerobots.com/ResearchRobots/PioneerLX.aspx`. [Online; accessed 25-February-2016].

[12] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The interactive museum tour-guide robot," in *Aaai/iaai*, pp. 11–18, 1998.

[13] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, *et al.*, "Minerva: A second-generation museum tour-guide robot," in *Robotics and automation, 1999. Proceedings. 1999 IEEE international conference on*, vol. 3, IEEE, 1999.

[14] I. R. Nourbakhsh, C. Kunz, and T. Willeke, "The mobot museum robot installations: A five year experiment," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 4, pp. 3636–3641, IEEE, 2003.

[15] R. Siegwart, K. O. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, M. Meisser, *et al.*, "Robox at expo. 02: A large-scale installation of personal robots," *Robotics and Autonomous Systems*, vol. 42, no. 3, pp. 203–222, 2003.

[16] G. Kim, W. Chung, K.-R. Kim, M. Kim, S. Han, and R. H. Shinn, "The autonomous tour-guide robot jinny," in *Intelligent Robots and Systems,*

2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, vol. 4, pp. 3450–3455, IEEE, 2004.

[17] B. Mohler, W. Thompson, S. Creem-Regehr, J. Pick, HerbertL., and J. Warren, WilliamH., "Visual flow influences gait transition speed and preferred walking speed," *Experimental Brain Research*, vol. 181, no. 2, pp. 221–228, 2007.

[18] R. Siegwart and I. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. A Bradford book, Bradford Book, 2004.

[19] ISR Lab, "Photo of Jinny." `http://isrlab.tistory.com/34`, 2010. [Online; accessed 12-February-2016].

[20] S. Rosenthal and M. Veloso, "Using symbiotic relationships with humans to help robots overcome limitations," in *Workshop for Collaborative Human/AI Control for Interactive Experiences*, 2010.

[21] J. Biswas and M. M. Veloso, "Wifi localization and navigation for autonomous indoor mobile robots," 2010. IEEE.

[22] J. Biswas and M. M. Veloso, "Localization and navigation of the cobots over long-term deployments," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1679–1694, 2013.

[23] J. Bruce, S. Zickler, M. Licitra, and M. Veloso, "Cmdragons: Dynamic passing and strategy on a champion robot soccer team," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 4074–4079, IEEE, 2008.

[24] M. J. Milford, G. F. Wyeth, and D. Rasser, "Ratslam: a hippocampal model for simultaneous localization and mapping," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 1, pp. 403–408, IEEE, 2004.

[25] 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*, 2010.

[26] H. Surmann, A. Nüchter, and J. Hertzberg, "An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments," *Robotics and Autonomous Systems*, vol. 45, no. 3, pp. 181–198, 2003.

[27] J. Biswas and M. Veloso, "Planar polygon extraction and merging from depth images," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 3859–3864, IEEE, 2012.

[28] J. Biswas and M. Veloso, "Depth camera based indoor mobile robot localization and navigation," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1697–1702, IEEE, 2012.

[29] B. Choi, C. Meriçli, J. Biswas, and M. Veloso, "Fast human detection for indoor mobile robots using depth images," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 1108–1113, IEEE, 2013.

[30] S. Müller, C. Weber, and S. Wermter, "Ratslam on humanoids-a bio-inspired slam model adapted to a humanoid robot," in *Artificial Neural Networks and Machine Learning–ICANN 2014*, pp. 789–796, Springer, 2014.

[31] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real time localization and 3d reconstruction," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1, pp. 363–370, IEEE, 2006.

[32] J. Pérez, J. A. Castellanos, J. Montiel, J. Neira, and J. D. Tardós, "Continuous mobile robot localization: Vision vs. laser," in *Robotics and*

Automation, 1999. Proceedings. 1999 IEEE International Conference on, vol. 4, pp. 2917–2923, IEEE, 1999.

[33] J. Biswas and M. Veloso, "Multi-sensor mobile robot localization for diverse environments," in *RoboCup 2013: Robot World Cup XVII*, pp. 468–479, Springer, 2013.

[34] D. Fox, W. Burgard, S. Thrun, *et al.*, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[35] K. Konolige, "A gradient method for realtime robot control," in *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 1, pp. 639–646, IEEE, 2000.

[36] D. Loughnane, "Design and construction of an autonomous mobile security device," Master's thesis, University of Waikato, 2001.

[37] A. Prakash, "Humanisation of an autonomous guided vehicle," Master's thesis, University of Waikato, 2004.

[38] C. Robinson, "Modifying an indoor mobile robot to enhance autonomous operation." Honours Report, Victoria University of Wellington, 2014.

[39] H. G. Nguyen, J. Morrell, K. D. Mullens, A. B. Burmeister, S. Miles, N. Farrington, K. M. Thomas, and D. W. Gage, "Segway robotic mobility platform," in *Optics East*, pp. 207–220, International Society for Optics and Photonics, 2004.

[40] S. N. Zealand, "Design for Access and Mobility - Buildings and Associated Facilities (NZS 4121:2001)," RFC 1654, RFC Editor, July 1995.

[41] SICK, "LMS100." `http://www.sick.com/group/EN/home/products/product_news/laser_measurement_systems/Pages/lms100.aspx`. [Online; accessed 22-February-2015].

[42] ROS, "Robot operating system." http://www.ros.org/, 2015. [Online; accessed 23-February-2016].

[43] Microsoft, "Kinect for Windows." https://dev.windows.com/en-us/kinect, 2015. [Online; accessed 23-February-2016].

[44] ASUS, "Xtion Pro." https://www.asus.com/3D-Sensor/Xtion_PRO/, 2015. [Online; accessed 23-February-2016].

[45] Intel, "RealSense." http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html, 2015. [Online; accessed 23-February-2016].

[46] Intel, "RealSense Specifications." https://software.intel.com/en-us/articles/realsense-r200-camera, 2015. [Online; accessed 23-February-2016].

[47] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for mobile robot navigation: Evaluation and modeling," in *Advanced Robotics (ICAR), 2015 International Conference on*, pp. 388–394, IEEE, 2015.

[48] T. Butkiewicz, "Low-cost coastal mapping using kinect v2 time-of-flight cameras," in *Oceans-St. John's, 2014*, pp. 1–9, IEEE, 2014.

[49] Microsoft, "Kinect for Windows SDK System Requirements." https://msdn.microsoft.com/en-us/library/dn782036.aspx, 2014. [Online; accessed 22-February-2016].

[50] TP-Link, "TL-WR702N Nano Router Specifications." http://www.tp-link.com/en/products/details/cat-9_TL-WR702N.html#specifications. [Online; accessed 22-February-2015].

[51] Microsoft, "Speech Platform SDK recommended system requirements." https://msdn.microsoft.com/en-us/library/

hh362873(v=office.14).aspx. [Online; accessed 22-February-2015].

[52] Nuance, "Dragon NaturallySpeacking recommended system requirements." http://nuance.custhelp.com/app/answers/detail/a_id/16262/~/system-requirements-for-dragon-naturallyspeaking-13. [Online; accessed 22-February-2015].

[53] ROS, "Laser scan message." http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html, 2015. [Online; accessed 11-January-2016].

[54] T. Foote, E. Marder-Eppstein, and W. Meeussen, "ROS Transform Package." http://wiki.ros.org/tf, 2015. [Online; accessed 21-January-2016].

[55] T. Foote, "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pp. 1–6, April 2013.

[56] K. Banachowicz, "ROS LMS1xx Package." http://wiki.ros.org/LMS1xx, 2014. [Online; accessed 11-January-2016].

[57] J. Binney, "ROS Laser Filters Package." http://wiki.ros.org/laser_filters, 2014. [Online; accessed 11-January-2016].

[58] M. Ferguson, "ROS rosserial Package." http://wiki.ros.org/rosserial, 2015. [Online; accessed 21-January-2016].

[59] Arduino, "Servo library." https://www.arduino.cc/en/reference/servo, 2015. [Online; accessed 21-January-2016].

[60] J. Blake, F. Echtler, C. Kerl, and L. Xiang, "libfreenect2." https://github.com/OpenKinect/libfreenect2, 2014 – 2016. [Online; accessed 20-February-2016.

[61] T. Wiedemeyer, "IAI Kinect2." https://github.com/code-iai/iai_kinect2, 2014 – 2016. [Online; accessed 12-June-2015.

[62] Khronos Group, "Opencl." https://www.khronos.org/opencl/, 2009 – 2016. [Online; accessed 21-January-2016].

[63] Intel China OTC, "Beignet." http://www.freedesktop.org/wiki/Software/Beignet/, 2013 – 2015. [Online; accessed 21-January-2016].

[64] P. Mihelich, "ROS Image Transport Package." http://wiki.ros.org/image_transport, 2015. [Online; accessed 21-January-2016].

[65] C. Rasmussen, K. Yuvraj, R. Vallett, K. Sohn, and P. Oh, "Towards functional labeling of utility vehicle point clouds for humanoid driving," *Intelligent Service Robotics*, vol. 7, no. 3, pp. 133–143, 2014.

[66] J. K. Mackay, "Automated landing site determination for unmanned rotocraft surveillance applications," 2014. Brigham Young University-Provo.

[67] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1–4, IEEE, 2011.

[68] F. Basso, M. Munaro, S. Michieletto, E. Pagello, and E. Menegatti, "Fast and robust multi-people tracking from rgb-d data for a mobile robot," in *Intelligent Autonomous Systems 12*, pp. 265–276, Springer, 2013.

[69] itseez, "OpenCV." http://opencv.org/, 2000 – 2016. [Online; accessed 22-January-2016].

[70] OpenCV, "Morphological Transformations." http://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html#gsc.tab=0, 2016. [Online; accessed 24-January-2016].

[71] J. F. Dave Hershberger, David Gossow, "ROS rviz 3D visualisation tool." `http://wiki.ros.org/rviz`, 2014. [Online; accessed 30-January-2016].

[72] OpenCV, "Blob Detector." `http://docs.opencv.org/master/d0/d7a/classcv_1_1SimpleBlobDetector.html#gsc.tab=0`, 2016. [Online; accessed 24-January-2016].

[73] E. Marder-Eppstein, "ROS Navigation Stack." `http://wiki.ros.org/navigation`, 2014. [Online; accessed 18-January-2016].

[74] B. P. Gerkey, "ROS AMCL." `http://wiki.ros.org/amcl`, 2015. [Online; accessed 18-January-2016].

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

[76] E. Marder-Eppstein, "ROS move_base package." `http://wiki.ros.org/move_base`, 2014. [Online; accessed 31-January-2016].

[77] E. Marder-Eppstein and E. Perko, "ROS local planner package." `http://wiki.ros.org/base_local_planner`, 2014. [Online; accessed 31-January-2016].

[78] B. P. Gerkey and K. Konolige, "Planning and control in unstructured terrain," in *ICRA Workshop on Path Planning on Costmaps*, 2008.

[79] A. Kelly, "An intelligent predictive controller for autonomous vehicles," tech. rep., DTIC Document, 1994.

[80] B. Gerkey, "ROS GMapping Package." `http://wiki.ros.org/gmapping`, 2015. [Online; accessed 18-January-2016].

[81] G. Grisetti, C. Stachniss, and W. Burgard, "OpenSLAM GMapping Algorithm." `http://openslam.org/gmapping.html`. [Online; accessed 18-January-2016].

[82] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," in *IEEE Transactions on Robotics*, 2007.

[83] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *IEEE International Conference on Robotics and Automation*, 2005.

[84] B. Gerkey and T. Pratkanis, "ROS map server package." `http://wiki.ros.org/map_server`, 2012. [Online; accessed 31-January-2016].

[85] ROS, "Geometry twist message." `http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html`, 2015. [Online; accessed 11-January-2016].

[86] K. W. B. G. Morgan Quigley, Brian Gerkey, "ROS joy package." `http://wiki.ros.org/joy`, 2013. [Online; accessed 10-February-2016].

[87] Microsoft, "Windows Presentation Foundation." `https://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx`. [Online; accessed 28-February-2015].

[88] D. Van Compernolle, W. Ma, F. Xie, and M. Van Diest, "Speech recognition in noisy environments with the aid of microphone arrays," *Speech Communication*, vol. 9, no. 5, pp. 433–442, 1990.

[89] E. T. Hall, *The hidden dimension*. Anchor Books, 1 ed., 1990.

[90] S. K. Singh, D. Chauhan, M. Vatsa, and R. Singh, "A robust skin color based face detection algorithm," *Tamkang Journal of Science and Engineering*, vol. 6, no. 4, pp. 227–234, 2003.

[91] Anastasia T, "Multiple RealSense Interference." https://software.intel.com/en-us/forums/realsense/topic/543419, 2015. [Online; accessed 23-February-2016].

[92] Jonathan Knoder, "Voice Recognition Software Review." http://voice-recognition-software-review.toptenreviews.com/. [Online; accessed 2-March-2016].

[93] Stu Robarts, "Speech recognition software: top six on the market." http://www.techradar.com/news/software/business-software/speech-recognition-software-top-six-on-the-market-1259815/2. [Online; accessed 2-March-2016].

[94] SoundHound Inc., "Houndify API." https://www.houndify.com/. [Online; accessed 2-March-2016].

[95] Api.ai, "Conversational Voice Interface." https://api.ai/. [Online; accessed 2-March-2016].

Appendices

Appendix A

Control Status Messages

```
Header header

# Connected Topics
string topic_enable_movement
string topic_enable_manual
string topic_enable_interaction
string topic_enable_navigation
string topic_cmd_vel_manual
string topic_cmd_vel_interaction
string topic_cmd_vel_navigation
string topic_cmd_vel_output
string topic_segway_status
string topic_segway_estop
string topic_whisker_scan
string topic_human_detected

# Command Source Enabled/Disabled
bool enable_movement
bool enable_manual
bool enable_interaction
bool enable_navigation

# Velocity Limits
float32 velocity_linear_max_human       # m/s
```

```
float32 velocity_angular_max_human      # rad/s
float32 velocity_linear_max_normal      # m/s
float32 velocity_angular_max_normal     # rad/s

# Velocity Error Thresholds
float32 velocity_threshold_linear       # m/s
float32 velocity_threshold_angular      # rad/s
float32 velocity_threshold_linear_time  # seconds
float32 velocity_threshold_angular_time # seconds

# Current Velocity Values
float32 velocity_current_linear_max     # m/s
float32 velocity_current_angular_max    # rad/s
float32 velocity_current_linear         # m/s
float32 velocity_current_angular        # rad/s
float32 velocity_measured_linear        # m/s
float32 velocity_measured_angular       # rad/s
float32 velocity_error_linear           # m/s
float32 velocity_error_angular          # rad/s
float32 velocity_error_linear_time      # seconds
float32 velocity_error_angular_time     # seconds

# Whisker Sensors
bool front_left
bool front_right
bool back_left
bool back_right

# Segway Connected
bool segway_connected

# Movement Control OK
bool ok
```

Listing A.1: Movement Control Status

```
Header header

```

```
# Connected Topics
string topic_enable_interaction
string topic_torso_request
string topic_torso_status
string topic_goal_request
string topic_cmd_vel
string topic_human_detected
string frame_base
string frame_human

# Thresholds and Deadzones
float32 zone_change_range_threshold
float32 zone_change_bearing_threshold
float32 segway_tracking_deadzone

# Flags
bool enable_interaction
bool human_detected
bool segway_tracking
bool ok

# Human Tracking
float32 human_range
float32 human_bearing
string zone
```

Listing A.2: Interaction Control Status

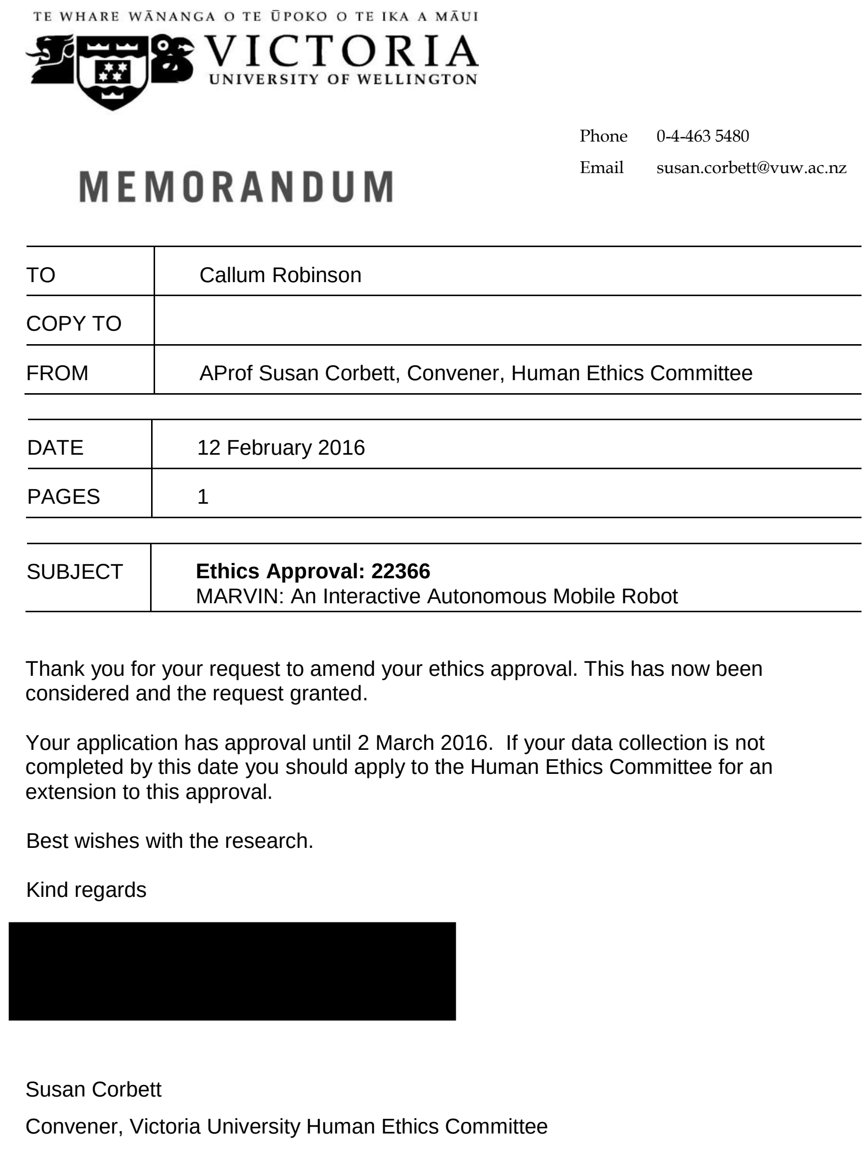
```
Header header

# Connected Topics
string topic_movement_status
string topic_movement_enable
string topic_movement_enable_manual
string topic_movement_enable_interaction
string topic_movement_enable_navigaiton
string topic_interaction_status
string topic_interaction_enable
string topic_power_status
string topic_kinect_angle
string topic_human_detected
string topic_request_manual
string topic_request_movement

# Timeout
float32 timeout_movement      # seconds
float32 timeout_interaction   # seconds
float32 timeout_power         # seconds

# Configuration
float32 process_rate              # hertz
float32 kinect_angle_default      # degrees
float32 kinect_angle_interaction  # degrees
float32 kinect_angle_navigation   # degrees

# State Machine
string state

# Navigation Stack
string current_goal
string navigation_status

# Flags
bool ok_movement
bool ok_interaction
bool ok_power
bool connected_movement
bool connected_interaction
bool connected_power
bool connected_gui
bool connected_navigation
bool enable_movement
bool enable_manual
bool enable_interaction
bool enable_navigation
```

Listing A.3: MARVIN Control Status

Appendix B

Usability Study

Ethics Approval



TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA UNIVERSITY OF WELLINGTON

MEMORANDUM

Phone	0-4-463 5480
Email	susan.corbett@vuw.ac.nz

TO	Callum Robinson
COPY TO	
FROM	AProf Susan Corbett, Convener, Human Ethics Committee

DATE	12 February 2016
PAGES	1

SUBJECT	**Ethics Approval: 22366** MARVIN: An Interactive Autonomous Mobile Robot

Thank you for your request to amend your ethics approval. This has now been considered and the request granted.

Your application has approval until 2 March 2016. If your data collection is not completed by this date you should apply to the Human Ethics Committee for an extension to this approval.

Best wishes with the research.

Kind regards

[redacted]

Susan Corbett

Convener, Victoria University Human Ethics Committee

Questionnaire

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

MARVIN – an Interactive Autonomous Mobile Robot, Victoria University of Wellington: Robot Interaction Evaluation

Please attempt each of the following tasks.

1. Approach MARVIN and wait for MARVIN to initiate interaction.
2. Request a map of your current location.
3. Request to be guided to one of the location options.
4. Follow MARVIN to your selected location. Please follow MARVIN from behind at a distance no closer than 1 m.
5. Request to be guided to start location.
6. Follow MARVIN back to the start location, as with step 4, follow from behind.
7. End your interaction with MARVIN.

Do you agree with the following statements? (Please circle one answer per question).

	Strongly Agree	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
Voice interaction (syntheses and recognition) was more useful than the touch screen interface.	○	○	○	○	○
MARVIN's voice was easy to understand.	○	○	○	○	○
MARVIN's voice was loud enough to hear clearly.	○	○	○	○	○
MARVIN understood my voice well.	○	○	○	○	○
The touch interface text was large enough to read.	○	○	○	○	○
The touch interface layout helped me interact with MARVIN.	○	○	○	○	○
Interaction with MARVIN was at a comfortable distance.	○	○	○	○	○
Requesting my location was easy.	○	○	○	○	○
The displayed location map was easy to read.	○	○	○	○	○
Requesting to be guided was easy.	○	○	○	○	○
MARVIN moved too slowly.	○	○	○	○	○
MARVIN moved too quickly.	○	○	○	○	○
MARVIN's upper body movement improved my interaction experience.	○	○	○	○	○

Please answer the following questions in the boxes provided below.

What aspects did you enjoy about interacting with MARVIN?

Are there any additional methods of interaction that you would like incorporated into MARVIN? For example, hand gestures such as pointing.

Is there any additional functionality you would like incorporated into MARVIN?

Was there anything about MARVIN's appearance that made interacting with it more difficult?

What do you feel could be improved about MARVIN's existing features?

Do you have any other comments?

Appendix C

Digital Content

The attached DVD contains the following:

- *PDF of this thesis*
- **Sensor Processing Code**
  - *ground_filter.cpp*
  - *kinect_calibration.cpp*
  - *kinect_pipeline.cpp*
- **Navigation Stack Code**
  - *whisker_layer.cpp*
- **Control Software Code**
  - *marvin_control.cpp*
  - *movement_control.cpp*
  - *interaction_control.cpp*
  - *manual_control.cpp*
- **HRI Code**

  - *marvin_gui.cs*

- *Video of MARVIN interacting with and guiding a human.*
- *Scan of full responses for Usability Study.*