A Mobile Robot for Home Help Applications

Submitted to the School of Electrical and Electronic Engineering, UCD in partial fulfillment of the requirements for the ME Electronic and Computer Engineering degree

Emma Foley

Supervisors: Robert Shorten & Brian Mulkeen

University College Dublin
College of Engineering and Architecture
School of Electronic and Electrical Engineering

April 2017

Abstract

This project proposes a robot and camera system for home help applications, intended to provide assistance to elderly people who wish to remain in their homes as they age. The system is implemented using readily available components to keep costs low and is intended to provide an alternative to currently available assistance robots which tend to be more expensive and complex.

Navigation strategies for the robot within a home environment are examined and the results presented. Potential use cases for the platform are also examined.

Contents

List of Figures

1. Introduction

The world's population is aging. The old age dependency ratio, defined by the UN as the ratio of the population aged 65+ per 100 population aged 20-64, is expected to increase from 14.4% in 2015 to 28.7% in 2050 [1]. Additionally surveys show an overwhelming preference by seniors to remain in their homes as they age [2]. This preference for 'aging in place' means that as the population ages there will be a need for innovative solutions to adapt homes to the needs of their elderly residents. The potential for assistance robots to fill this need is obvious and research in this area has been ongoing for many years. Examples include the Giraffplus [3], intended for telepresence, monitoring, and simple household tasks, and the Care-o-Bot [4], a realisation of the robot butler often predicted in science fiction. However this research effort has not yet produced a widely adopted or commercially successful product, for several key reasons.

Firstly, these systems tend to be expensive, with the Giraffplus retailing for approximately €10000. The focus on state-of-the-art hardware and complex user interfaces push the costs beyond the means of the average retired person. Secondly, the reaction to these systems has been ambivalent. User trials have yet to strike a balance between a robot which provides a useful service and one which intrudes on the lives of its users. Particular concerns are that the robots will be used as a replacement for social interaction, or that the privacy of users will be compromised.

This project examines a different approach to a service robot. Given that visual and aural degradation are a common cause of concern for otherwise independent adults, a system is proposed with applications such as carrying a phone to a person with hearing difficulties, or acting as a guide to a person with a visual impairment. The system is developed using existing components including an Asus Xtion camera for user tracking and a Roomba as a mobile platform. The cost for the components is kept low at approximately €400 which positions this platform as a lower cost alternative to the expensive systems described above.

1.1. Overview

This paper is structured as follows: Section 2 reviews the state of the art in home assistance technologies and robot navigation. Section 3 describes work completed in a previous project which forms the basis of the proposed system. Section 4 outlines the aims of the project including the proposed use cases for the platform. Section 5 covers the hardware components while Section 6 describes the software written for the project and the operation of the system. Section 7 describes the tests performed and the results obtained. Section 6 discusses the limitations of the platform and explores the potential for further development of the system. Finally Section 7 concludes the paper with an assessment of the success of the project.

2. Literature Review

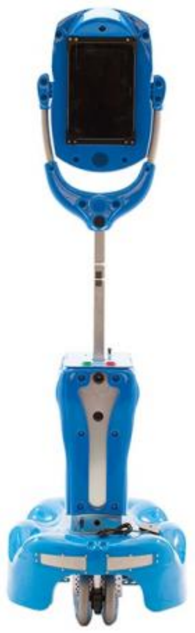
This section reviews the material researched during initial development of the project. The material discussed encompasses two main subject areas. Firstly the current state of assisted living technology is reviewed with particular attention to robotic systems. Secondly literature relating to navigation strategy is examined, including approaches to Roomba-like robots in navigation research. Path planning and mapping strategies suitable for use in a home environment are also explored.

2.1. Assisted Living and Aging in Place

With the advent of smartphones in recent years and the proliferation of the Internet of Things soceity is becoming more open to the concept of a 'smart home'. Wearables such as the Apple Watch or FitBit have become commonplace, Google Home and Amazon Echo listen for voice commands, ambient sensors control climate and cleaning robots such as the Roomba assist with household tasks. Such devices have the potential to greatly improve quality of life for the aging in place market. [5] identifies several areas of need in the home which can be addressed by technological means. These include facilitating carers to monitor a residents activity, wellness and safety; automating tasks that the resident cannot perform; alerting caregivers in case of difficulty; and providing a social link for the resident through telepresence and audio-visual units. Some of these needs can be best fulfilled by ambient sensor networks, while others like the automation of household tasks require an active platform such as a service robot.

Monitoring of the person needing care allows peace of mind for carers while maintaining the independence of the subject. Wearables and sensor networks are the approaches best suited to monitoring applications. Wearables in the digital health field are far from a new concept, with wrist based devices for fall detection and heart rate monitoring being the most widely known examples. Devices such as the Philips Lifeline provide an alert system for emergency services in the case of a fall [6], and have been modernised to include features like GPS tracking. Newer systems even have predictive capabilities. For example the Kinesis QTUG [7], in development in partnership with Shimmer Sensing, analyses the gait of the wearer to anticipate a fall before it happens. The VivaGo [8] is an example of a with a smartphone-synced wristband to display information on heart rate and other symptoms and alert the wearer to any anomalies. The major disadvantage of these systems is uptake; they require the user to wear the device consistently, the success of which is highly user-dependent.

Ambient sensors are becoming more sophisticated and for some use cases can provide a sufficient monitoring system without the need for wearable devices. In [9] a system is presented based on a network of PIR or presence infra red sensors placed at strategic points throughout a home. These sensors monitor the activities of the residents such as what time they get up and go to bed, how often they use the bathroom or cook, whether they leave the house and how long they take to return. The aim is to monitor the autonomy of the resident and to provide an alert in the case of a loss of autonomy, indicated by a change in routine. Systems have also been proposed which use speech and video data from cameras in the home to monitor a patients health [10]. The speech data can be analysed to determine whether the user is in pain. while video can detect fall events or be used to monitor routine as in [9]. Cameras can also be used for simple surveillance. With any monitoring system, and in particular those that stream data


(a) Giraffplus
Source: robotics.usc.edu

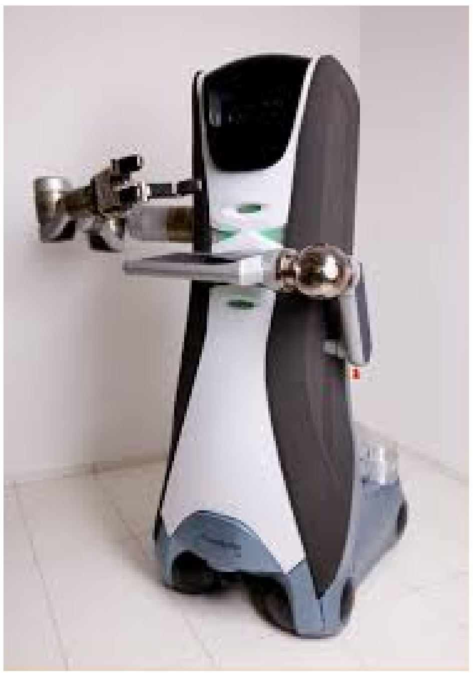

(b) Care-o-Bot 3
Source: care-o-bot.de

Figure 2.1.: Examples of service robots

directly to the carer rather than simple alerts, there will be a concern about data privacy. Feedback from target users of a telemonitoring device in [11] stated that they would be uncomfortable with having a video stream active in their home.

Monitoring systems can be extremely useful in facilitating carers, but more resident focused applications may require a system within the home to respond to the users needs. The concept of service robots is moving closer to mainstream with a handful of products on the market today and many more in development. These robots can perform a broad range of duties from household chores to facilitating social links through telepresence.

An example of a telepresence robot on the market today is the Giraffplus [3], which is designed to act as a virtual visitor in the home. The Giraffplus boast a tablet interface for calls and can be remotely operated by a carer through a web application, or controlled by the user. Cybi [12] is another telepresence device, and has been described as a robot companion. Similarly to the Giraffplus, Cybi facilitates video calls from carers and also provides a video stream for monitoring by a remote carer.

Physically assistive robots such as the HOBBIT [13] and the Care-o-Bot [4] perform tasks such as fetching and carrying objects and in the case of the Care-o-Bot, potentially chores such as laundry. However the Care-o-Bot has been in development since 1998 without a suitable commercial application being found, and users in a trial of the HOBBIT in [14] found the robot to be frustrating to use and not providing enough benefit for the cost. The indications are that an assistive robot has not yet been designed that elderly people would actually use.

Service robots such as the examples above have many potential benefits. However in trials, users have raised concerns about the impact of using robots as a replacement for real social interaction. In [4] healthcare workers acknowledged the utility of the robot for household tasks such as laundry but were concerned about the impact replacing human care could have on their patients. Another study involving a group of Americans over the retirement age found that that those interviewed acknowledged the utility of service robots but did not see themselves using one, maintaining that they were more independent than stereotypes suggest and that they would prefer human interaction [11].

2.2. Robot Navigation and Mapping

The Roomba and similar robots have been used extensively in navigation research due to their low cost and simple geometry - the compact footprint and the ability to turn in place is convenient for route planning. The most common approach in literature is to mount laser scanners on the robot [15], which produce a two-dimensional odometry scan at ground level. By making continuous scans as the robot moves around its environment and aligning similar features in the scans a map can be built. The drawback of these scanners is that they are expensive. With the advent of affordable depth sensing cameras such as the Asus Xtion, similar navigation strategies using such cameras have been explored. In [16] the depth information from an RGBD camera is used for localisation. The camera is mounted on a wheeled robot and the robot captures point clouds from its surroundings. The image produced is down projected to approximate a laser scan, and the scan is matched to known map data to determine the location of the robot. In [15] the authors propose an open source robotic 3D mapping framework based on 'stop-scan' datasets gathered from spherical RGBD cameras mounted on mobile robots. In [17] a series of RGB and depth images from a hand held Kinect camera are used to generate a 3D map of an indoor environment. Here the depth map is used to generate a 3D point cloud which in turn is used to extract features and create the map. In [18], a Nao humanoid robot navigates a 3D environment using the Asus Xtion camera mounted on its head. The map is constructed by filtering a 3D space into 'voxels' or volume elements, and determining whether these elements are free or occupied using point cloud data. The 3D voxel grid can then be used by the robot for navigation.

A common theme in many of these approaches is the use of point clouds and the Point Cloud Library. Point clouds are a structure representing 3D information on the points in a scene. The information is usually taken from a depth image although other approaches such as stereo images or repeated laser scans can be used. The Point Cloud Library [19] is an open source library with functions to capture, process and utilise point clouds.

2.3. Path Planning

The problem of producing a navigable map from laser scan or image data has many approaches. [20] identifies three:

1. Road map: Identify a set of routes within the free space.
2. Cell decomposition: Split the environment into discrete cells and distinguish between free and occupied spaces.
3. Potential field: Impose a mathematical function over the space

Both road maps and cell decomposition are popular for indoor mobile robotics applications as they allow efficient graph based path planning algorithms to be utilised.

A simple initial path planning algorithm is the Breadth-First search. Breadth-First search begins at a source node and explores equally in all directions from the this node until the goal is found. This algorithm is useful as a starting point but for applications where efficiency is important there are obvious optimisations to be made [21].

Counter to the Breadth-First search is the Depth-First algorithm, which begins at the start node, selects a node to explore and continues along this path until it is blocked. At this point the algorithm returns to the start node and repeats the process. In [22] a sonar scanner is used to explore and map an environment based on advancing frontiers, with unknown cells given higher priority. The path planner uses a Depth-First search to find a path to the unknown frontier. Depth first search is suited to this application as the goal is not a single node but a frontier, and so searches in many directions are likely to reach it.

Another well known algorithm path planning algorithm is Dijkstra's algorithm. This algorithm improves on the Breadth-First Search by introducing the concept of variable cost. For example, in an environment including both flat and rough terrain, the map cells covering rough terrain will be more difficult to traverse than the flat areas. By

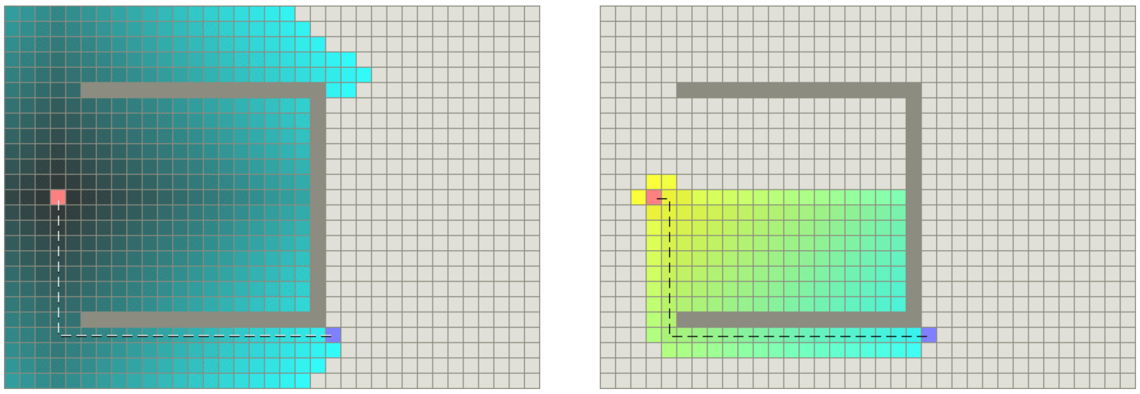

(a) Dijkstra (b) A*

Figure 2.2.: Comparison of Dijkstra's and A* algorithms
Source: [21]

marking the rough terrain cells with a higher cost it is possible to give preference to paths over flat terrain. Dijkstra's algorithm explores all possible paths similarly to the Breadth-First search until there is no path with a lower cost than the current cost to the goal. [21][23]. In this way the optimal path to the target will be found. However the search is still inefficient.

Dijkstra's algorithm may be further improved upon by introducing the concept of a heuristic function. Heuristic functions use a 'rule of thumb' to give priority to certain nodes in the search. For example, a best-first search algorithm would give priority to nodes in the direction of the goal. A well known heuristic search algorithm is A*. Like Dijkstra's algorithm A* evaluates the cost to reach each node but also the estimated cost between that node and the goal node. A* performs at least as well as Dijkstra's algorithm and much better for particular kinds of obstacles; see fig 2.2. [21] [23]

Many variants of A* have been developed, for example D* which accounts for transitions across non-adjacent cells. [23] Field D* [24] is a further variation with the intention of producing smoother paths in some applications by allowing paths in any direction.

3. Prior Work

This project builds on a foundation developed in a previous ME project entitled “A Semi-Autonomous System to Aid the Eldery”[25]. The aim of this project was the production of a commercially viable support system for deployment in a home environment.

The platform was developed for the specific use case of acting as a mobile phone for a user with hearing difficulties, but was intended to be flexible enough to be adapted to other tasks. The aim was to automate the process of answering the phone while also decreasing the distance from the ringing phone to the user. These two functions were projected to significantly decrease the likelihood of a missed call, making it easier for family members to reach the user. This allows the family peace of mind while also allowing the user to maintain their independence.

3.1. Hardware Components

The current state of the art of home care technology was examined and it was concluded that a robotic platform could best fulfil the stated aims. The platform was required to fulfil four functions:

1. A mobile robot, to perform tasks such as carrying the phone
2. User localisation to allow the robot to navigate to the user
3. Localisation of the robot within the space

4. Operation of the phone through the mobile platform

For reasons of cost and user acceptance it was determined that the mobile platform should be low-profile and should require minimal user input when not in use. The aim was that the system should be as unintrusive as possible. The robot chosen for this purpose was the iRobot Create2 Roomba. The Roomba is small in size and its popularity as a cleaning robot indicates that users are comfortable with its presence in their homes, something notably lacking in user trials of larger robots. The Roomba is equipped with a suite of useful sensors for indoor navigation, including cliff sensors, collision sensors, wheel encoders and an infrared receiver for docking. In addition there is an open interface available for development, the Create2 Open Interface.

The robot is controlled by an Intel Edison board, connected via a custom designed circuit to the serial port of the robot (see Figure 3.1). The circuit functions as an adapter to connect the Arduino board pins to the serial port of the robot, as well as stepping down the voltage from the unregulated, up to 21V output by the serial port to the 7-15V required for safe operation of the Edison board. This allows the board to be powered by the robot which is an essential requirement for the platform to be mobile.

Some additional configuration is needed to enable the serial port connection and to set the roomba director script to run automatically. Specifically a script should be run to enable the relevant GPIO pins on the board and start the roomba program and this script should be run on startup. This configuration is detailed in the appendices.

Wearables were examined as a possible user localisation system but were rejected for the reason that user uptake of such devices is unreliable. Instead an RGBD camera was then chosen for user localisation, aim of a single camera per room. The Asus Xtion is ideal for this purpose: it connects to the master system via USB and is compatible out of the box with a number of useful open source libraries - OpenNI, OpenCV, NiTE. Installation of these libraries is detailed in appendices.

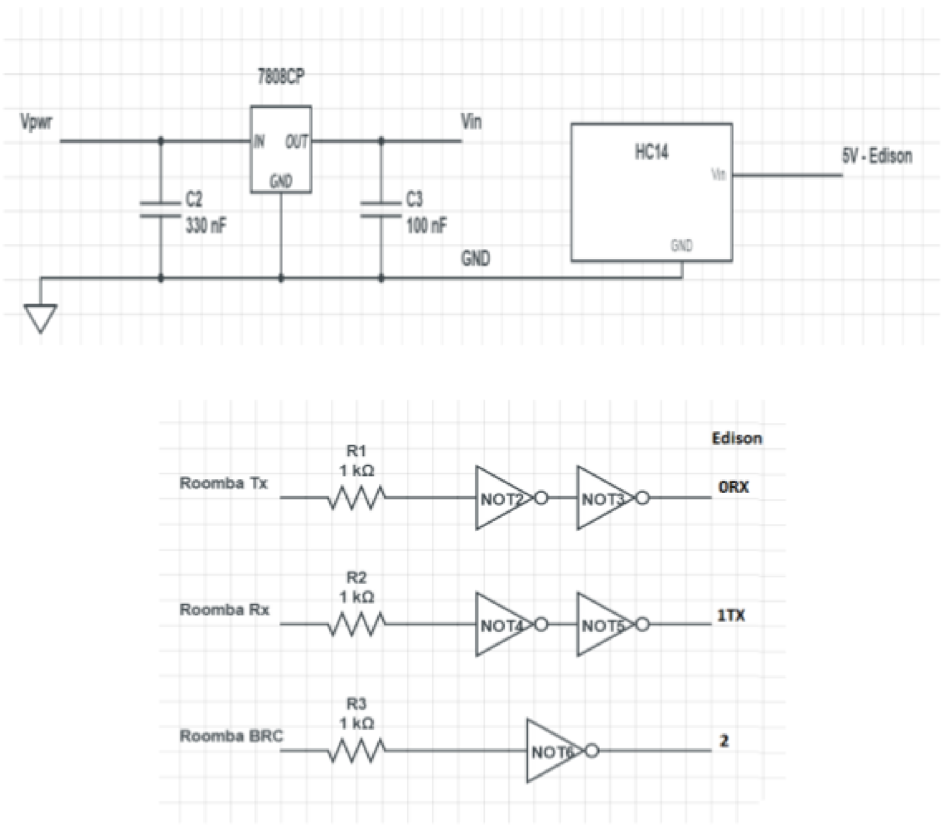


Figure 3.1.: Circut diagram for custom interface board
Source: [25]

For the purpose of detecting call events an Android smartphone was chosen. Used in conjunction with a Bluetooth speaker mounted on the robot the mobile platform is complete. The speaker used is a Sonivo speaker which can be powered via the USB sockets on the Edison board.

3.2. Software

An implementation of the mobile phone use case was developed. User tracking was implemented using the Asus camera and a navigation strategy for the robot was developed which was referred to as 'augmented dead reckoning'. The robot navigated to a given set of coordinates and once this destination had been reached the robot would check for a position reading from the camera, updating its internal position estimate if data was available. Communication between the robot, Android phone and base station was handled via UDP sockets.

The program execution was as follows:

1. Robot is idle on the dock. Polls UDP sockets regularly to detect a call event from the phone. Robot is assumed to be positioned on the dock, facing away from the area of operation.

2. Call event is received. Robot checks UDP socket for user destination from base station. Robot backs up from the dock and turns 180°. The distance and angle from the robot position to the user is calculated and the robot turns and heads in the direction of the user. If an obstacle is encountered a simple back-up-and-turn manoeuvre is used to recover.

3. Destination has been reached and call is in progress. Robot idles near the user and polls UDP sockets to detect a hangup event.

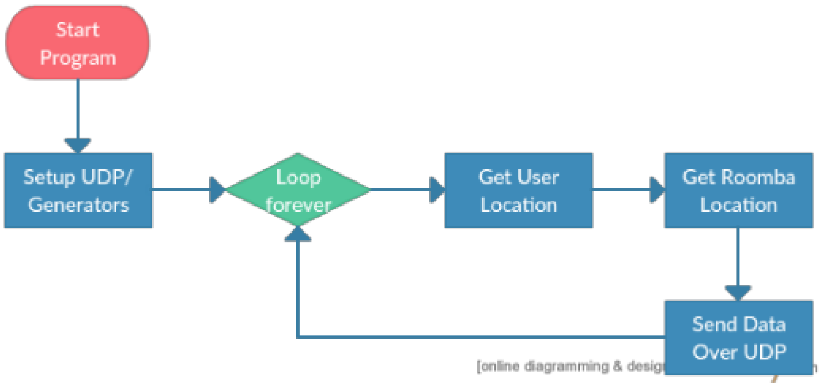


Figure 3.2.: Program execution
Source: [25]

4. Hangup event received. The robot navigates back to the dock using the navigation procedure outlined in 2 and 3.

Camera tracking

The user tracking software leverages the OpenNI and NiTE libraries using the python wrappers available in PyOpenNI. OpenNI is an open source framework designed to facilitate 'natural interaction' in applications, meaning control by means of movement and gestures as well as audio commands. The OpenNI framework first requires a context to be defined which holds the complete state of all applications using OpenNI. A generator node is then defined, in this case the depth camera image, which identifies the information to be used for tracking. Skeleton trackers are initialised which will identify the 3D position of twenty different points on the body. From this information the (x,y) coordinates of the subjects torso can be extracted; this is the definition of user position used for this application.

The robot tracking is performed using the OpenCV library, a computer vision library which provides functions for image processing. The colour and depth images are captured

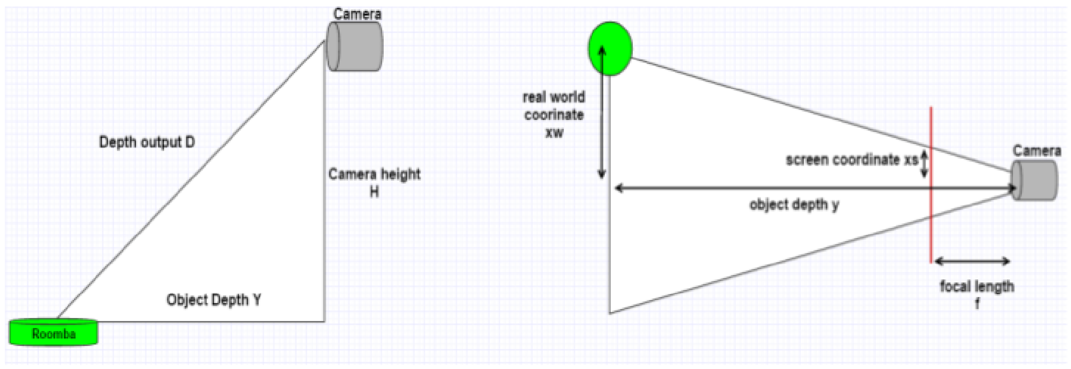


Figure 3.3.: Calculation of robot coordinates
Source: [25]

and shaped into numpy arrays for processing. The bright green colour of the robot's top plate is used to identify the robot in the scene - a mask is applied to the image to extract pixels matching this colour. Contours in this masked image are identified and their centre pixel identified. The robots real world position xw, yw can then be determined using information from the depth image and simple trigonometry.

$$yw = \sqrt{D^2 - H^2}$$

where D is the depth reading for the robots center pixel and H the camera height;

$$xw = yw \times \frac{xs}{f}$$

where xs is the horizontal distance of the pixel from the centre of the screen, and f is the focal length of the camera. (See Fig. 3.3)

The user and robot tracking runs in a continuous loop. The two positions are calculated, converted to a string and sent via a UDP socket on each loop. The user tracking is assumed to be robust as it uses standard OpenNI library functions. A simple error check is performed on the robot position where it is compared to the last recorded value. For a large difference the reading is assumed to be in error. For this error case or if either the robot or the user is not detected their position will be recorded at (0,0).

Robot director

The robot navigation builds from a library called PyRoomba which provides simple wrappers for the instructions provided in the Create2 Open Interface. These functions include basic driving and turning as well as reading from various robot sensors.

Encoder polling is implemented to determine the distance and angle travelled by the robot. As the robot moves the left and right wheel encoders are polled continuously and the change in encoder value between two readings can be used to compute the distance travelled and angle turned:

$$\Delta distance = \frac{1}{2} \times \frac{\delta left + \delta right)}{encoderticksperrotation} \times \pi \times wheeldiameter$$

$$\Delta angle = (\delta right - \delta left) \times \frac{wheeldiameter \times \pi}{encoderticksperrotation}$$

The robot operation is a series of discrete moves. The encoders are continuously polled in a loop. To turn, the robot is given an instruction to begin turning and the current encoder values are captured; the change in angle is calculated from each new reading until the required angle is reached. An instruction to stop turning is then sent and the new orientation of the robot is recorded. A similar process is followed for drive moves, with the addition that once the drive move is completed the position is compared to the current camera reading; If the two readings are close then robots internal position estimate is updated to match the camera reading. For the mobile phone application, the robot continuously polls the UDP sockets until a signal for a call event is received. It then backs up off the dock and calculates the distance and angle from its current position to the user. It turns and drives towards the user using the procedure described above. After each move it compares its current position to the destination. If this value is within a given tolerance it will stop; otherwise it will repeat the process. Collision recovery is achieved using a set back-up-and-turn manoeuvre. Depending on the location of the collision, the robot will back up a set distance and turn right or left. For a head on collision a direction is chosen at random. The robot then drives forward again. In this way it gradually finds its way around an obstacle.

3.3. Results

Six tests were carried out on the system:

1. Camera Accuracy - the camera was used to track both the robot and user at stationary set points and the error in terms of the Euclidean distance between the measured and actual position was recorded. The camera was found to have error rates between 50 and 200 mm with an average of 95 mm over both measurements. The effective area for the camera was found to be an area of 2m $\times$ 4m in front of the camera, and the remaining tests were carried out within this range.

2. Low-speed dead reckoning: the robot was sent to a range of targets at 200mm/sec, without camera updates. The time to delivery and the error on arrival was measured. The robot reached its destination on all tests with an average error of 250mm and an average time to delivery of 20s for distances of about 4m.

3. High speed dead reckoning; the same experiment was repeated for a driving speed of 400mm/sec, although turning speed remained at the lower rate. As could be expected the time to delivery for this experiment was faster at around 10s, however the error rate was much higher with the robot arriving between 600 and 900 mm from the target. Successful delivery was defined as an error below 400mm and so these tests had a 100% failure rate.

4. Augmented dead reckoning: the same targets were used as in the previous experiment and the speed was unchanged but camera updates were now included. The robots error rate was reduced to an average of 260 mm/s, comparable with the low speed case. The time to delivery increased slightly due to additional moves required for correction once the updates were received.

5. Augmented dead reckoning with obstacles: an obstacle was placed in the path of the robot, both directly in the middle of the path and at a position to cause a small perturbation, and the ability of the robot to recover was tested. For this experiment the accuracy actually showed an improvement, which was determined to be due to the fact that the robot received updates before and after the collision recovery moves. This resulted in more updates overall and so a more accurate position estimate. There was some loss in time to deliver, however with the improved accuracy these effects were small, with the average time increasing from 17s to 20s.

6. Complete demonstration: the mobile phone use case was demonstrated, with the robot routed to a user standing at the targets and triggered by call and hangup events on the phone. Four out of six runs reached the user successfully with a similar error and time to delivery to the static target case. On the other two runs the robot drove out of the scene and was lost by the camera.

The author concluded that the system was successful in the uncomplicated test environment but that it would have more problems in a larger or multi-room environment. Suggestions for future development were proposed including more frequent camera updates and the possibility of having the robot learn a map of the environment by recording collisions. It is also noted that the camera could be utilised for further development including fall detection.

4. Project Goals and Potential Use Cases

This section summarises the goals that this project set out to achieve. Potential use cases for the platform developed are examined, including the differences in navigation strategy required for each.

4.1. Goals

The aim of this project is to develop a robust navigation strategy for a mobile robot operating in a home environment. Improvements will be made to the localisation system of the robot so that accurate position information is available at all times. A map of the environment will be built using the camera already in place for user tracking and will be augmented using information from the robot's collision sensors. Routes will be planned using this map and followed by the robot, with experiments carried out for comparison of different routing strategies.

The first goal is to update the camera localisation and hence improve accuracy of the robots internal position estimate. Where previously the robot received updated position information on completion of a move, it will now continuously update its position estimate as it drives to match the position data received from the camera. Two way communication between the camera and robot programs is also added, which improves the camera readings by reducing the search area of the camera to the immediate neighbourhood of the robots reported position. These improvements to the position accuracy are vital to implement a successful navigation strategy. The intended use cases of the project will require the robot to operate continuously and have the ability to take long routes, and accurate position information is key for both of these aims. Experiments will be conducted to confirm the utility of the improved camera localisation.

Secondly the robot will be programmed to take planned routes around objects in the environment rather than using trial and error to find a path. Clearly in order for routes to be planned for the robot the environment first needs to be mapped. The Asus Xtion camera in position for user tracking can be utilised to build an initial map. The 3D depth image information will be processed to transform it into a 2D occupancy grid map suitable for navigation by the robot. For areas out of view of the camera the robots sensors will be used to augment the map. As collisions occur the location of the obstacle will be recorded so that future paths will avoid it. Over time this will allow the robot to learn its environment and produce consistently more accurate routes.

Included in the mapping process is a calibration to determine the location and angle of the floor plane. This calibration will be used to improve the flexibility of the tracking system so that a specific camera height and angle is not required for correct operation.

The base station software will be updated to communicate a planned path to the robot rather sending the user position only. Different path planning strategies will be explored. In particular, the tradeoff between a strategy prioritising speed and a risk averse strategy taking a wide route around obstacles will be examined. Several use cases are proposed for the project and the differences in routing behaviour required to adapt the robot to these use cases will be examined in the next section.

The route planning will also be used to improve the behaviour of the robot in a collision. Instead of using a predefined trial and error strategy to find its way around a new obstacle, a new route which this obstacle into account will be calculated for the robot to follow.

Finally, the software in the system will be updated to implement ROS, the Robot Operating System. This system is used as a standard in many commercial and research robotics applications. The advantage of the system is in communication, simplifying the development effort by providing structured communication between different software components and devices.

4.2. Use Cases

Three use cases have been proposed for the platform:

1. Mobile Phone
2. Emergency Phone
3. Robot Guide

The mobile phone was the original inspiration for the project. In this case, the robot would reside in a home environment and carry a Bluetooth speaker which is connected to the user's mobile phone. When the phone rings, the robot navigates to the user, answering the phone en route. This use case is intended to be helpful to people with hearing loss, making it easier for family members to reach them in their homes.

The second use case of an emergency phone is intended for users who are mostly independent but may have mobility or health issues. In this use case the camera would monitor the user and in the event of a fall, would respond by having the robot navigate to them while dialling a family member or emergency services.

The third use case is intended for a setting unfamiliar to the users. The robot and camera system is set up in a clinic or hospital environment and acts as a guide for users with visual impairments, bringing them to preset locations such as a bathroom.

The behaviour of the robot must be tailored to each of these applications as different use cases will require different considerations. The case of the emergency phone will clearly have speed as the highest priority, but will also be expected to be used rarely. Thus the robot should thoroughly explore its environment when not in use and when an emergency occurs take the fastest route possible. A phone in normal use will be less urgent but will still require some consideration to speed.

Conversely, for the robot guide application priority must be given to user safety, so a wide margin around obstacles is optimal. Here shortcuts such as passing under tables are not possible as the robot must take a route passable by a human user. The system must therefore rely more heavily on the camera for accurate environment detection.

With the system installed and capable of navigating a home environment it is possible to envision a multi-functional robot. For example a robot equipped for the mobile phone use case could also act as an emergency phone. The iCreate2 Roomba could also function as the original was intended and work as a vacuum cleaner when not in use. The cleaning function could even be performed while the robot is mapping its environment.

5. Hardware Components

This section summarises the hardware components used in the platform.

The system has two main components: the robot, which provides a mobile platform, and a base station system with a camera attached for user and robot tracking. Communication is carried out using WiFi, with most of the computation being performed on the base station to minimise computing power needed on the robot. The hardware components were chosen with a low cost specification in mind.

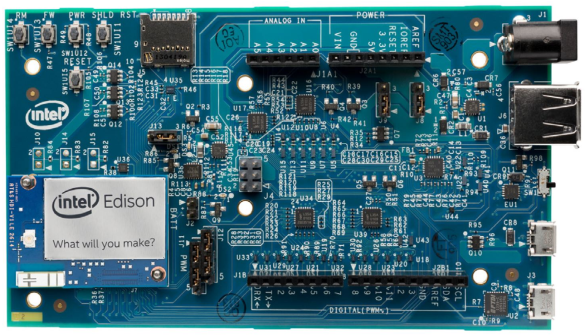
5.1. Robot and Controller

The robot used is the iRobot Create2 Roomba, a developer model of the popular autonomous cleaning robot. The robot is designed for home use and has a number of advantages in this regard: it is low profile and sized to navigate around and under furniture easily (330 mm in diameter and 92mm high at its highest point). It is also equipped with basic sensors for navigation, including wheel encoders to measure distance and angle travelled, a bumper with infrared and physical sensors to detect obstacles, cliff sensors to avoid sudden drops and an infrared detector to detect signals from the docking station. [26]. At time of writing this Roomba module retails for $200 at `http://store.irobot.com/default/create-programmable/`

For software control of the robot an Intel Edison module is connected via the serial port. The Intel Edison was developed as one of a series of low power system on chips

(a) Create2 Roomba
Source: irobot.com

(b) Asus Xtion
Source: asus.com



(c) Intel Edison with breakout board
Source: developer.android.com

Figure 5.1.: Hardware components

designed for use in IoT applications. It contains a high-speed, dual-core processing unit and features the Yocto operating system for ease of use. It is also equipped with integrated WiFi which makes it well suited for this application. For this project the Edison is used in conjunction with an Arduino breakout board. A custom interface circuit is used to connect the robot's serial port to the Arduino inputs. The board also regulates power levels to allow the board to be powered via the serial port. Excluding the custom board the cost for this component is $109 at `https://www.adafruit.com/product/2180`

5.2. Base station and Camera

The camera used is an Asus Xtion Pro Live. It is equipped with both an RGB and depth sensing camera as well as a microphone array. The depth sensor is important for the tracking and mapping applications required as it can provide information about the room in 3D, while the colour image is used to track the robot as described in the software section. The Xtion is also compatible with a number of useful open source libraries which greatly simplify the development effort required. It comes bundled with the OpenNI and NiTE frameworks which provide out of the box capabilities for user tracking, and is compatible with the PointCloud ans OpenCV libraries required for the mapping and robot tracking functionality. The camera retails for around \$150.

The camera connects via USB to the base station which for the purposes of this project was a laptop running Linux, but more practically could also be a microcontroller similar to the Intel Edison used for the robot.

6. Software

The software written for the system is broken up into four distinct programs. All code is written in Python and is available at `https://github.com/emmafoley2/roomba_navigation`

1. Mapping - captures a point cloud from the camera and converts it to a 2D occupancy grid map. Also performs a calibration to determine camera height and angle.
2. Tracking - uses OpenNI and OpenCV libraries to track the user and robot
3. Path Planning - plans a path from the robot to the user.
4. Robot control

The first three programs run on the base station. The mapping and tracking programs both utilise data from the camera which is connected to the base station via USB, so this is the logical location for both. The routing program also runs on the base station, but receives all its information from other programs and could equally run on the robot. Keeping it on the base station however reduces the computing power required for the robot controller. Although the Edison board used here is quite powerful, there is utility in developing a system that could be run on a more lightweight controller.

Information is shared between programs using the Robot Operating System (ROS). ROS is an open-source software package designed to provide a structured communications layer on top of each host operating system. The system has the advantage of simplifying the development effort by handling lower level operations such as TCP setup and transmission, and provides a manageable way to coordinate continuous data streams from the different programs. Messages are published to a topic by one program and this topic can be subscribed to by any number of other programs. The subscriber class handles message processing asynchronously by default so that incoming messages and processing never block the main thread. The topic framework is designed to provide a standard communication protocol so that components can be switched out simply by having a different program publish to the relevant topic, for example if a different user localisation system became available this could be easily substituted for the camera tracking program. This modularity is in keeping with the original aim of having a flexible platform.

6.1. Program flow

The execution of the program is as follows:

1. Initially the robot is docked, its position assumed at (0,0) and orientation at 270°. The mapping program is run to populate the initial occupancy grid.

2. The map is built and routes can be planned. The localisation program continuously tracks and publishes the users position. The robot publishes its position as (0,0) along with a flag to indicate that the route should be calculated from the home position. The routing program captures these messages and computes a path from the home position to the user. This route is continuously published as a set of waypoints and captured by the robot.

3. The robot receives a signal to go to the user, triggered by a phone call or other event. It backs up from the dock to the home position and turns through 180°to

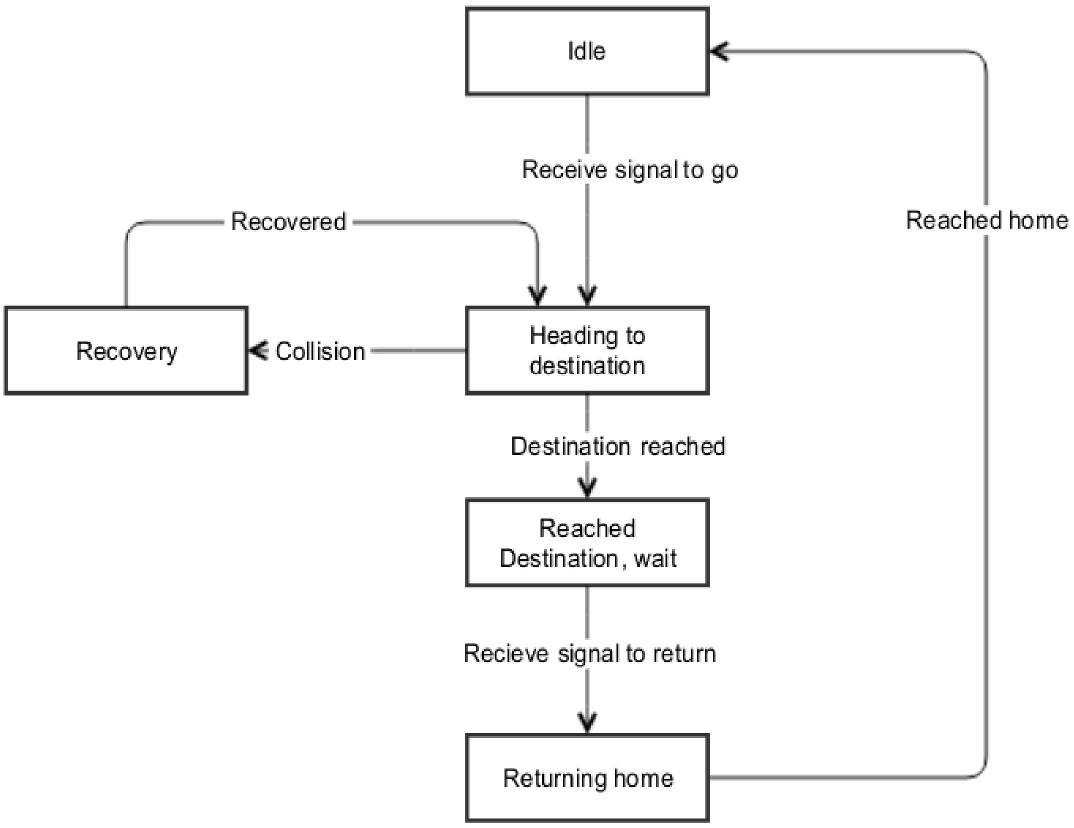


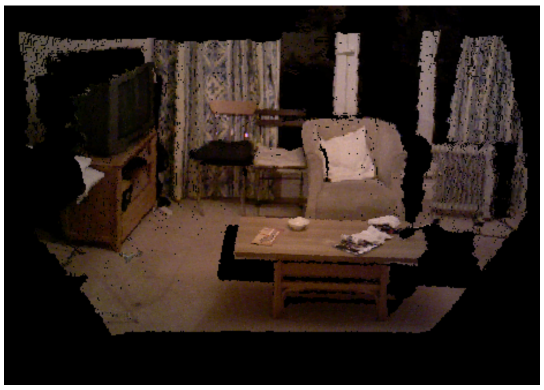
Figure 6.1.: Program flow

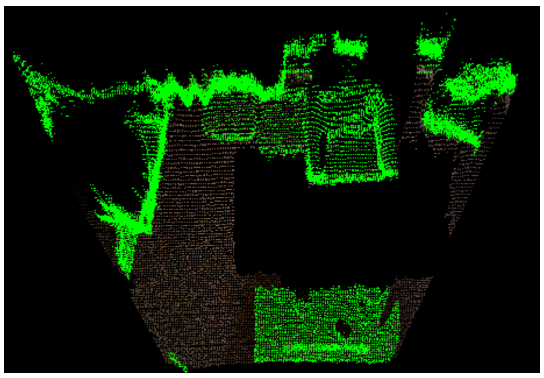
face the scene. The distance and angle to the first waypoint in the route are calculated and the robot turns and drives to it. This is repeated for each waypoint until the destination is reached. As the robot moves along the route it continuously publishes its position information. The camera tracking program publishes (0,0) for the robot position until it is in view when its recorded position will be published. The robot director captures these messages and compares the two estimates of its position. If the camera report is within half a meter of the internal estimate this internal estimate is updated to match the new data. The robot director also publishes a flag to indicate that it is en route to its destination. The routing program responds to this flag by calculating routes from the most recent robot position to the user.

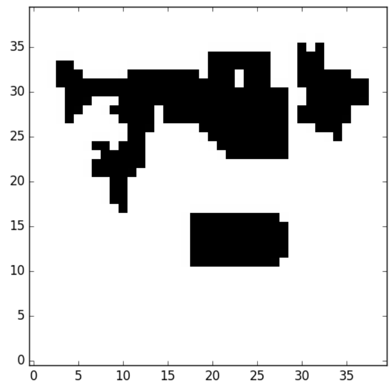
4. Even with the map available the robot may encounter unknown obstacles resulting in collisions. To recover from a collision the robot backs up a set distance. Meanwhile the routing program adds the new obstacle to the map so that the next route calculated will avoid this obstacle. The new route is then published as normal and the robot sets off on the new path. In the case that a collision occurs within a short distance of the user the robot may instead determine that its destination has been reached. Alternatively in the case where several attempts at recovery fail to find a clear path the robot will return to the dock.

5. The robot reaches its destination and waits for a signal that its task is finished, in the mobile phone use case signalled by a hangup event. A flag is set to indicate that the route planned should now return the robot to the home position.

6. The robot receives signal to return home. Again the most recent route is processed and the robot follows the waypoints one by one back to the dock. The robots built in docking algorithm is used to return it to the dock and the robots charging state


(a) Original pointcloud


(b) Overhead view


(c) Occupancy Grid Map

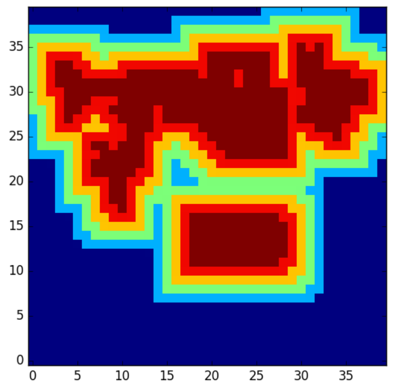

(d) Weighted Map

Figure 6.2.: Mapping process

is monitored to determine whether the dock has been reached. Once the dock is reached the home position flag is set once again and the robot awaits a new signal.

6.2. Map

The mapping program uses the depth camera data to produce a 2D occupancy grid map of the environment.

The mapping program uses point clouds as its basis. Point clouds are structures that represent the 3D position of each point in the scene relative to the camera. By manipulating this 3D structure a 2D occupancy grid can be obtained.

The ROS OpenNI package `openni.launch` is used to publish a stream of point clouds from the Asus Xtion camera. The mapping program then captures a point cloud from the relevant ROS topic and processes it. The first step is to filter the cloud. Invalid points are removed, for example points in the image for which no depth information could be gathered. Next the cloud is filtered using a voxel grid - a 3D mesh which divides the point cloud into cubic cells and marks these cells as occupied or empty. These two steps reduce the size of the point cloud by a factor of five. Since the original point cloud is of far higher density than is required for the grid, and since the next steps involve iterating over all points, the processing time can be hugely improved by filtering in this way. The point cloud stores position information for each point relative to the camera's coordinate system. To make the map this must be converted to real world coordinates and so a calibration must be performed to determine the camera's position relative to the floor. The point cloud is segmented into planes and the angle and distance from the camera for each plane is compared. To distinguish between planes some assumptions: that the camera is neither upside down nor pointed very steeply at the floor but is positioned for a decent view of the scene, so the angle should be between 0 and 30 degrees; and that the floor is the lowest visible plane, i.e for planes in the correct angle range the floor will be furthest away. The angle and height of the camera can then be calculated and the point cloud is rotated and translated to align with the coordinate axes used in the rest of the system. Finally the point cloud is scaled and down projected to the floor plane to produce a 2D grid consisting of occupied (value 1) and free (value 0) cells.

Currently the mapping is performed once at the beginning of execution, with the assumption that the map does not change over execution time. The grid values, camera height and camera angle are saved to a text file for access by the other components in the system. The augmentation of the map by the robots sensors is performed within the routing program. This approach was chosen largely due to time constraints and improvements could definitely be made here. For testing, mapping once at the beginning of execution was sufficient but in practice this system is intended to run continuously for the duration of its installation in a home. Clearly in this case it would be better to re run the mapping procedure periodically, and to bring the program more in line with the other software components by sharing the map and calibration values via ROS messages.

6.3. User and Robot tracking

The tracking program provides position information for both the user and the robot.

The user localisation system is unchanged from the previous implementation. The Asus Xtion camera is used in conjunction with the OpenNI framework to track the users movements via the depth image.

The robot tracking software is also similar to the previous implementation, with some adjustments. The colour image is processed to extract pixels matching the robot in colour, and then the screen coordinates of the center pixels were transformed to real world coordinates using calculations involving the camera height. Previously the camera height was hard coded. An obvious improvement was to take advantage of the height and angle calibration performed in the mapping process. The mapping program stores this information in a text file which is read from the tracking program on loading. The second improvement was in reducing incorrect reports. In the previous project it was noted and in testing it was observed that the camera could report in error where there is another green object present in the room, or due to confusion by reflections. Previously a simple error check was introduced to compare consecutive reports, which reduced the erratic reports but would fail for two consecutive mistaken reports. The solution was to have the tracking program receive information on the robot's internal position estimate from the relevant ROS topic. In the case where there is more than one green contour

User Position (x,y)mm			Robot Position (x,y)mm		
Actual	Detected	Error (mm)	Actual	Detected	Error (mm)
0, 1000	23,1060	64	0, 1000	(out of view)	
0,2000	15, 2103	104	0,2000	-43,2149	155
0, 3000	-24,3153	154	0, 3000	-59, 3206	214
-1000,1000	(out of view)		-1000,1000	(out of view)	
-1000,2000	-1050, 2136	144	-1000,2000	-872,2149	203
-1000, 3000	-1045, 3166	171	-1000, 3000	-919,3206	221
1000,1000	(out of view)		1000,1000	(out of view)	
1000,2000	900,1954	110	1000,2000	879,2057	133
1000,3000	919,3014	82	1000,3000	827,3156	232

Figure 6.3.: User and Robot position tracking error (average of five readings)

detected in the image, the tracker will report the reading closest to the robot's current belief as this is clearly the most likely to be the robot.

The messages between the robot and camera were converted to ROS topics, with the user and robot position now published in seperate messages. This means that each topic only deals with one piece of information which is in line with the ROS philosophy of modularity. The tracking program loops continuously, tracking both the user and robot. If either is not in sight the coordinates (0,0) are published to the relevant message.

In fig 7.3 the localisation accuracy is tested for both the user and robot for a range of positions in the operating area. The robot and user were positioned at each set of coordinates for five readings and the average value was taken. The error rates detected were consistent with those found in the previous project, with an average of 120 mm in user tracking and 190mm for robot tracking.

6.4. Routing

The routing program subscribes to the robot and user position messages and continuously publishes a route for the robot. The route planned depends on a flag in the robot message. For normal operation the route is from the robot to the user. When the robot reaches its destination the flag indicates that the route should be plotted from the robot to the 'home' position directly in front of the dock. When the robot is docked the route is from the home position as this is where the robot will be when it has backed off the dock and turned.

The router reads the grid map file produced by the mapping program and processes it for robot navigation. The size of the grid cells was chosen to be greater than $\frac{d}{2\sqrt{2}}$, where d is the robot diameter, so that the footprint of the robot lies within one cell of its centre. By padding the obstacles on the map with a margin one cell in width routes can be planned for the robot which consider only the path of this centre point. Extending this idea it is easily seen that if the robot follows a route bordering the blocked reason its edges will be very close to the obstacle and a small error will result in collision. Routes that take a wider path will carry a lower risk. Hence each cell on the map can be given a weighting between 1 and 0 relative to its proximity to any obstacles.

This approach allows a range of routing strategies to be considered, balancing the length of the route with the risk of collision. The A* algorithm is used to plan the route. This algorithm is one of a class of heuristic algorithms which compute the optimal route by comparing the cost of the route to a heuristic. In the case of A* the heuristic is Euclidean distance. So for each node beginning at the starting point the algorithm computes the optimum next step in relation to the Euclidean distance and the cost of moving between the two cells. When the node is reached these optimum steps are retraced to find the optimum path. The path is converted to waypoints by finding the points where it changes direction, these waypoints forming the route message published for the robot.

In order to produce a safer path the cells can be weighted so that the cost of moving to a cell close to an obstacle outweighs the possible benefit of a shorter Euclidean distance from that cell to the goal.

The robot may encounter obstacles that were not included in the map, for example in an area of the map not in view of the camera. Information about the collision status of the robot is included in its position message for this purpose. When the robot encounters an obstacle the routing program can then add this information to the map and calculate a new route which avoids that cell. For certain applications it may be desirable to avoid this situation and in that case cells not in view of the camera should be given a higher weight so that the algorithm will plan paths through known cells where possible.

6.5. Robot behaviour

The robot director program captures a route published by the routing program and follows it point to point. An internal estimate of the robot's position is maintained using a combination of data from the robot's wheel encoders and updates from the tracking program.

The encoder polling, driving and turning functionality developed previously was adapted into a custom library to extend the capabilities of the Pyroomba library. To recap, the wheel encoders for the robot are polled continuously and the change in value from one reading to the next can be converted to a distance travelled and angle turned using the formulae below:

$$\Delta distance = \frac{1}{2} \times \frac{\delta left + \delta right)}{encoderticksperrotation} \times \pi \times wheeldiameter$$

$$\Delta angle = (\delta right - \delta left) \times \frac{wheeldiameter \times \pi}{encoderticksperrotation}$$

Previously, the robot's internal position estimate was updated only at the end of a move, and this estimate was not shared with other software components. For this application it was necessary to have the robot publish its position, and since the routing and tracking programs both require an accurate robot position to be published continuously, it was necessary to make the position updating continuous also. As the robot moves, the encoders are polled and with each incremental move the robot's internal position estimate is updated. Maintaining a consistent estimate of the robots position also allows use to be made of all messages from the camera rather than only at waypoints. With each position update, the robot compares its new position estimate to the latest camera reading. If the difference between them is below a threshold of half a metre the robot's position estimate is updated to match the camera reading. This updating serves to correct errors in the robots position estimate which stem from inaccuracies in the wheel-encoder data and the fact that the robot travels and turns differently on different floor types.

The approach to collision recovery was also changed to take advantage of the continuous position and route planning. While the robot is in operation the routing program it allows the route to be updated as the robot moves, so that if a collision occurs the robot can back up a safe distance and then follow a new route which avoids the discovered obstacle. To facilitate the routing program in adding new obstacles the robot position message includes a flag to indicate the current collision status.

On startup the robot is assumed to be on the dock. The robots charging state is checked to confirm this and if the robot is not charging its built in docking function is triggered. The dock is assumed to be positioned at (0,0) which in the coordinate system used is directly underneath the camera. The robots orientation is initialised at 270°which corresponds to the robot facing directly away from the area of operation, as this is the position it will dock in. The coordinate system used is centred on the camera with the y axis in the floor plane along the camera direction and the x axis perpendicular.

The robot waits for a signal, publishing a flag which indicates that it is docked. The starting signal was triggered by a keypress for testing purposes but which could also be implemented using the call reporting app developed for the mobile phone use case. On receiving the signal the robot backs up from the dock and turns 180°. The position of the robot after this manouevre is at (0,500) and while the robot is docked the routing program will always calculate routes from this 'home' point, this is to allow the robot to get clear of the docking station before beginning its route. The robot records the last point on the current route as its destination. The distance and angle to the first waypoint on the route is then calculated, and instructions are sent to turn the robot through the required angle and start driving. When the waypoint is reached the same calculation is performed for the next point, and so on, all the while publishing a flag to indicate that it is on the move. With short path segments the waypoint is usually reached on the first attempt, but with longer paths errors may build up and a corrective manouevre may be needed to reach the point. To avoid slowing the robot down by too many corrective manouvres the waypoint is considered reached if the robot is at a distance less than 250mm away. If a collision occurs on the way to a waypoint, the position of the robot is noted so that if the destination is less than a 400mm threshold away the robot determines that the destination has been reached. The robots collision status is published regardless so that the obstacle is noted, and if the destination is further than 400mm away the robot will recover from the collision as detailed above and continue on its way. When the destination has been reached the robot waits for a second signal to indicate that it should return to the home position. The robots route request flag is updated to indicate that the destination has been reached, which means that the next route calculated should be to the home position. The robot follows the route received back to this position, docks using its built in docking algorithm, and returns to waiting for a signal. Once the robot has docked its position and orientation are again updated to the starting position so that errors accumulated on its journey are reset.

7. Results

A testing environment was set up in the laboratory with different maps laid out to test the various routing strategies. The goals of the tests were:

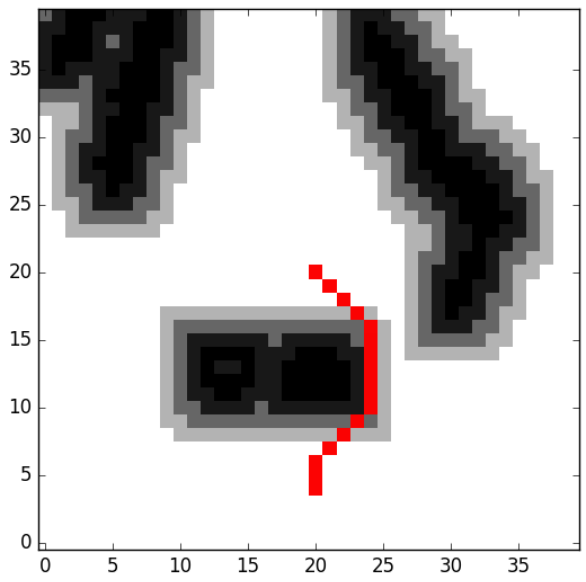
1. To demonstrate successful delivery of the robot to a target over a range of paths, including recovery from collision with an unknown obstacle.
2. To demonstrate the benefit of the camera localisation system by comparing results with the camera updates both on and off
3. To examine how the robot's performance is affected by driving speed
4. To compare the performance of a risk-averse routing strategy to one that prioritises speed

7.1. Initial Tests

To test the robustness of the newly implemented navigation a range of test environments were set up. The robot was routed to a static target and for each run the delivery time and accuracy of the robot was measured and the route noted. A successful delivery is defined as any run where the robot stopped within 400mm of its target, with a time to delivery of less than thirty seconds. These parameters are based on the figures used previously to test the mobile phone use case, where the operational range of the Bluetooth speaker and the average time a caller will wait for an answer were the aims. For the emergency phone use case the priority will obviously be speed, while for the guide use case accuracy and a smooth path are more important. These cases are considered when examining the results.

The speed of the robot was set to 250 mm/s for these initial test runs and the camera updating was in operation. The test runs were under the following conditions: A totally clear path for first confirmation, a path with an obstacle in view directly between robot and camera, a path with a narrow route between two obstacles, and a long route around several obstacles. In each case the robot successfully delivered to the target, demonstrating that the navigation is robust and reliable under a variety of test conditions.

To test collision recovery an obstacle not present on the map is placed in the robots path. The aim of this test is to determine whether the robot can re route around this obstacle and also to show that it will avoid this obstacle in future runs, or in this case on the return journey. It was found that on average the robot is able to take a new route away from a small obstacle (of the same magnitude as the robot diameter) within 1-2 re routing maneouvres, and take a clear path on the return journey. The limited range of the robot sensors mean that it cannot easily determine the extent of a new obstacle and larger obstacles pose more of a problem, requiring more attempts to re route and sometimes causing the collision limit to be reached, resulting in a failed delivery.



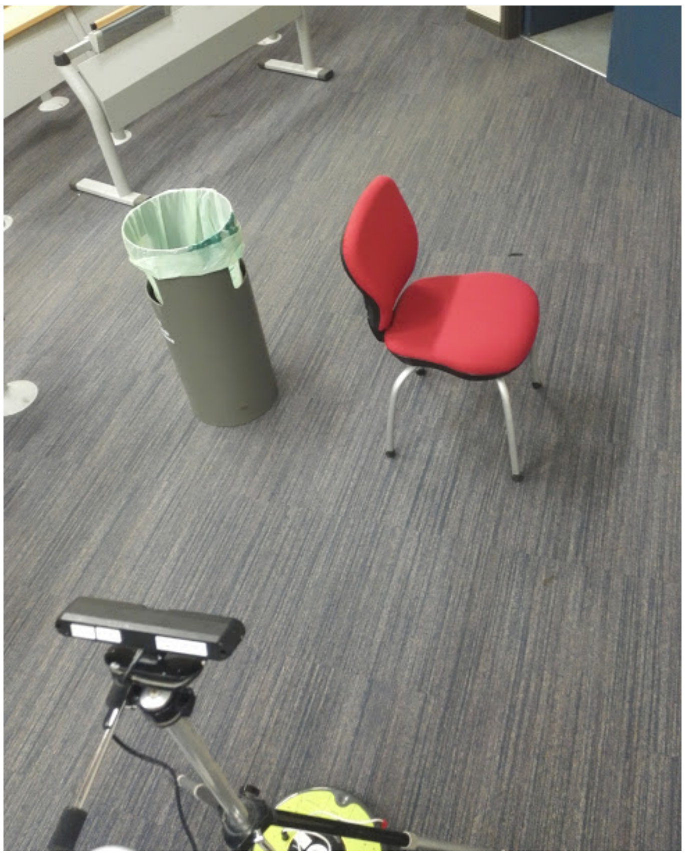
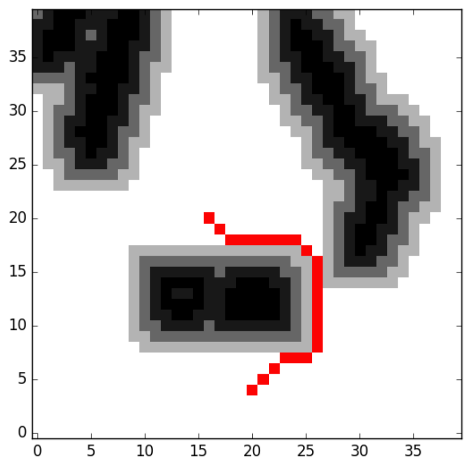
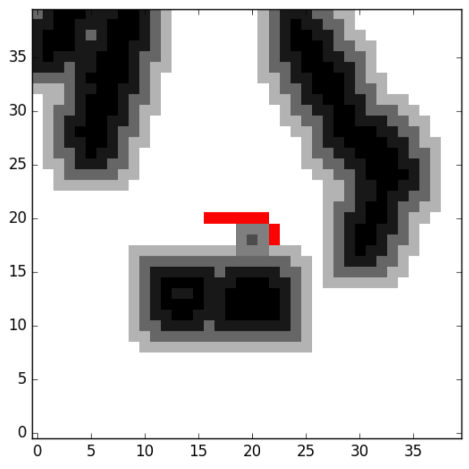


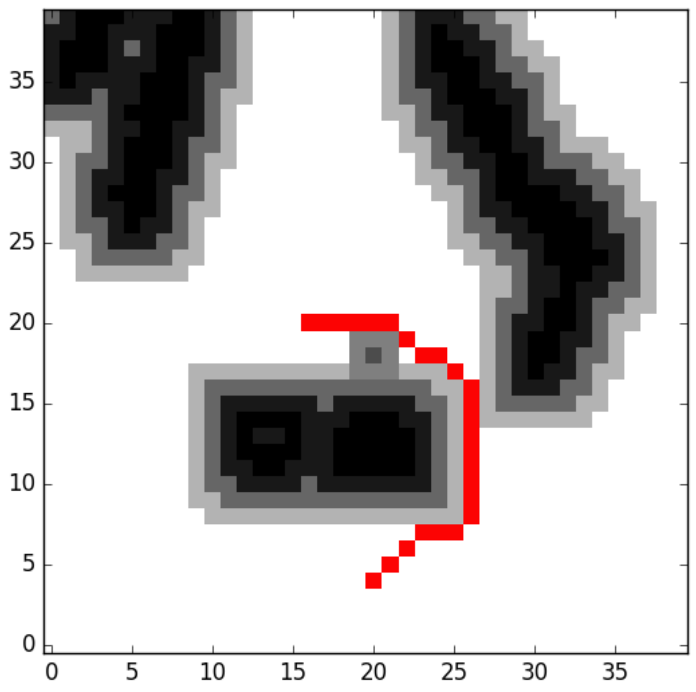
Figure 7.1.: Example test setup with route to a static target beyond the obstacles



(a) Initial, planned route



(b) New route after obstacle encountered



(c) Return route avoiding newly mapped obstacle

Figure 7.2.: Images from routing program showing collision recovery and map augmentation

7.2. Camera Localisation and Driving Speed

To demonstrate the effectiveness of the camera updates a test environment is set up so that the robot follows a planned, clear route both with and without updates from the camera. The position of the robot as detected by the camera and its internal position estimate are logged and compared.

For consecutive runs on a set path it is found that the error between the camera and robot position is consistently smaller when camera updating is included. The error is of the order of hundreds of mm without camera updates and of the order of tens of mm with updates. In Fig. 7.4 the robot travelled a set path with the target at (-1500mm, 2500mm). In the first case the robots internal coordinate logging recorded that the robot had reached the target, when in fact errors had built up and the camera recorded a final position much further out at (-1750,3000). On the second run the camera updates are enabled and the robots internal position estimate is kept much closer to the camera reading.

A test environment is then set up so that the robot follows a route to a human user standing at a consistent target position (Fig. 7.5. A narrow path between two obstacles is chosen so that the robots position accuracy is important for successful delivery. The error between the robots arrival point and the target is measured as is the travel time. Failure to deliver is defined as stopping at a greater error than 400 mm or taking longer than 30s to arrive. The test is run at two speeds, the robots top speed of 500 mm/sec and a lower speed of 250 mm/sec.

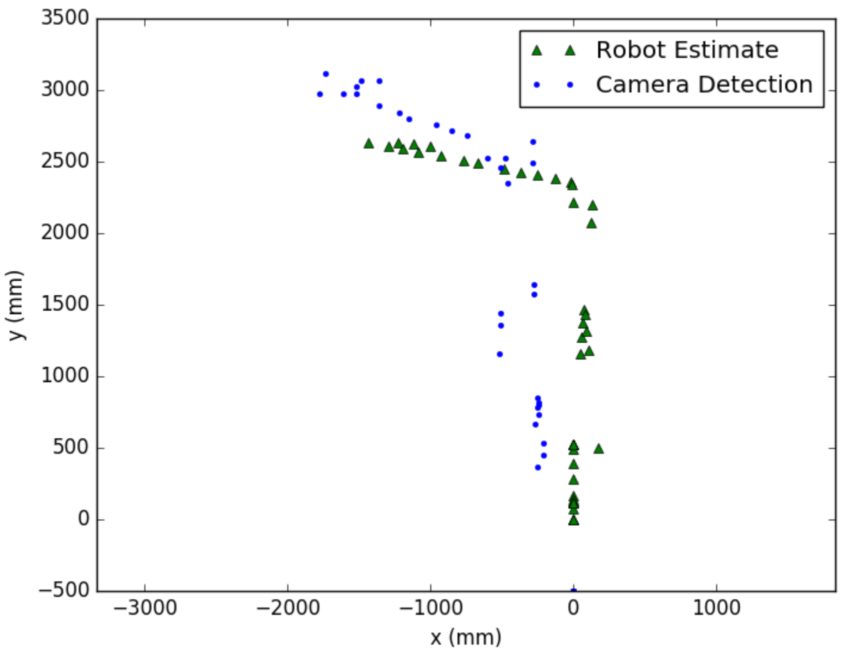
The performance in the case without camera updates is variable, which is to be expected as the error in the robots internal position estimate does not follow a consistent pattern. Factors such as the angle at which the robot leaves the dock can greatly impact the accuracy of the internal logging. The success rate for the tests without updates is 25%, while the rate for the tests with updates is 100%. At higher speeds the failure rate

Trial	Updates	Speed (mm/sec)	Time to deliver (sec)	Error (mm)
1	On	250	15	300
2	On	250	23	200
3	On	500	16	250
4	On	500	13	310
5	Off	250	Failed to deliver	n/a
6	Off	250	21	200
7	Off	500	Failed to deliver	n/a
8	Off	500	Failed to deliver	n/a

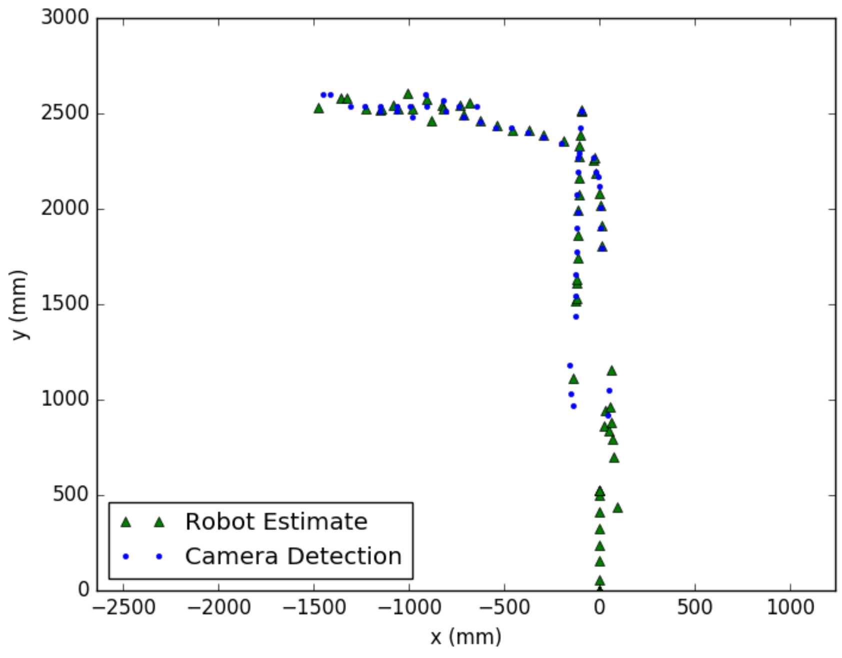
Figure 7.3.: Localisation test results

for the tests without updates is 100%, and the robot was observed to repeatedly collide with the obstacles in the test environment. Collision recovery via re routing was largely unsuccessful due to the incorrect position reported by the robot. This demonstrates that the camera updates are key to ensuring consistent performance of the robot.

For the tests with camera updates enabled the time to deliver decreases and the performance in terms of error remains consistent between higher and lower speeds. At high speed the robot was observed to occasionally collide with the test environment obstacles, indicating that its position estimate was not entirely accurate. However in these cases the robot was still able to recover from the collision and complete delivery.

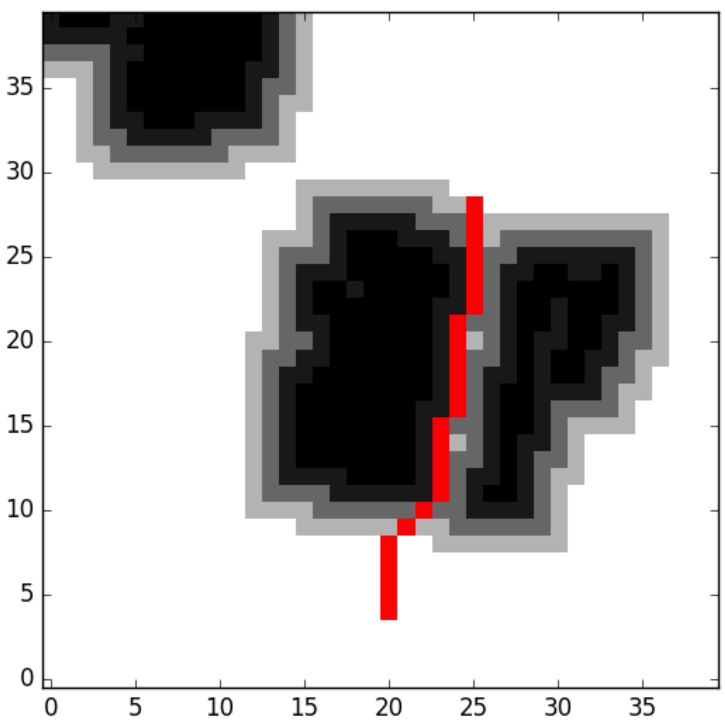


(a) Without camera updates the error is clear



(b) With updates the two measurements are consistently closer together

Figure 7.4.: Robot Localisation with and without camera updates



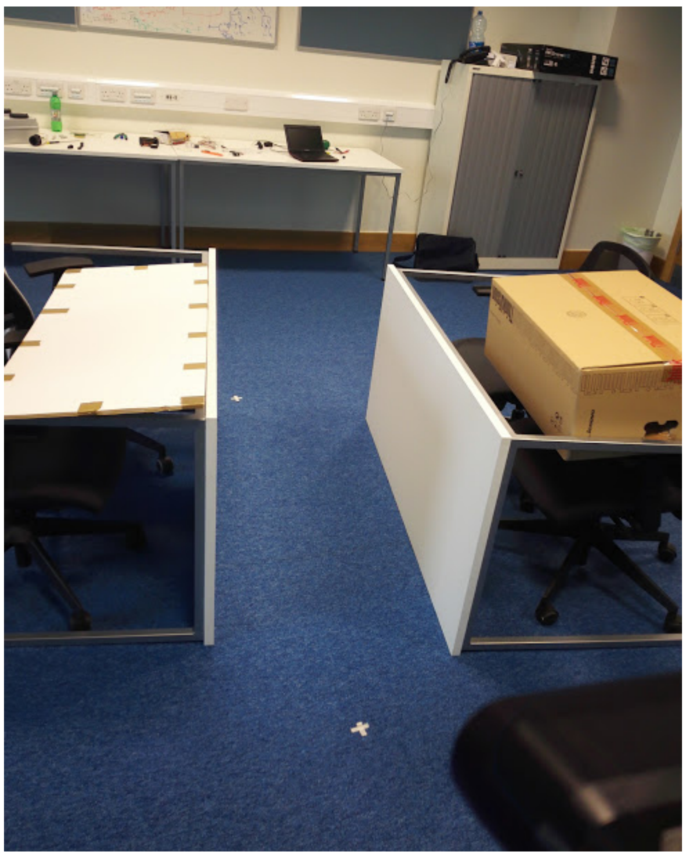


Figure 7.5.: Setup and route for localisation tests

Trial	Updates	Speed (mm/sec)	Strategy	Time to deliver (sec)	Error (mm)
1	On	250	Risk Averse	18	200
2	On	250	Risk Averse	19	250
3	On	250	Risk Averse	20	80
4	On	250	Fast	15	290
5	On	250	Fast	17	150
6	On	250	Fast	23	200

Figure 7.6.: Results for routing tests

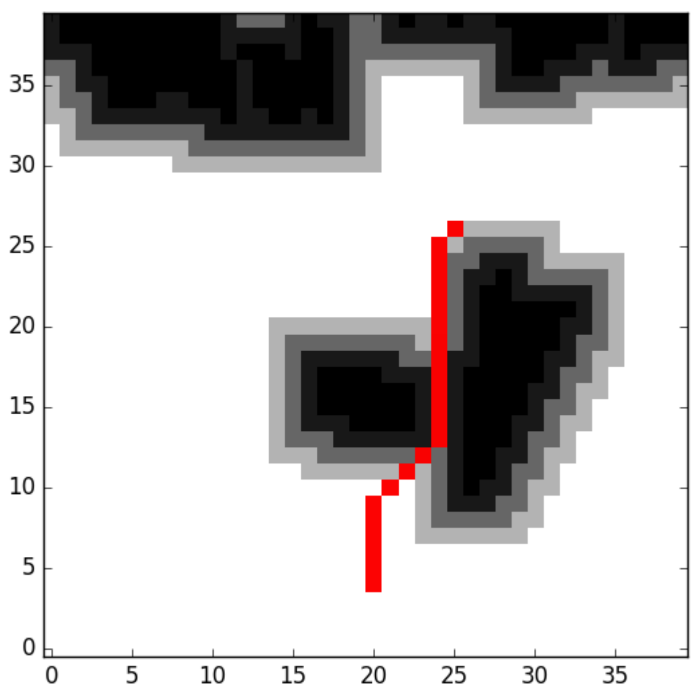
7.3. Routing Strategies and Recovery

To examine the performance of the robot for different routing strategies a test environment is laid out where both a short route between two obstacles and a safer route bypassing the obstacles are available (Fig. 7.7). The route taken depends on the importance given to risk avoidance which is a configurable parameter in the program.

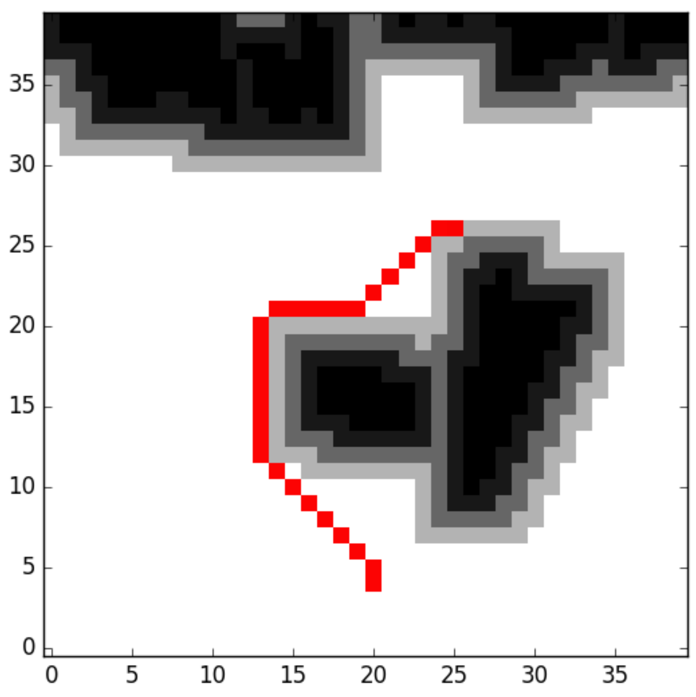
The robot makes several runs on each path with a consistent speed of 250 mm/sec. The time to delivery and error upon arrival is measured as for the localisation tests.

As could be expected the time to delivery is faster on average for the shorter route than for the risk averse strategy. On one trial of the faster route the robot collided with the obstacles on the narrower path, resulting in a higher time to delivery. The error is similar for both strategies, indicating that the longer route does not affect the accuracy.

Overall the results for the two strategies are comparable. The difference between the short and risk averse routing strategies may have become more apparent if a larger testing environment was used, however the test environment was designed to be at the scale of an environment where the system would normally operate. For applications such as the guide use case where collision avoidance is more important than speed the risk averse strategy may be more appropriate.



(a) Fast: shortest route passing between two obstacles



(b) Safe: longer path to give obstacles a wider margin

Figure 7.7.: Fast vs Safe routing strategies

8. Limitations and Future Work

This section examines the limitations of the system developed with consideration to the operation environment it was designed for. The potential for future development is outlined including addressing these limitations and the need for more thorough testing in a realistic environment.

8.1. Limitations

Basing the navigation on a wall mounted camera restricts the robot to a very small field of operation. The camera can only observe one room; a typical household has more than one room so multiple cameras would be needed. Large parts of the room may also be out of view behind furniture. Since it was demonstrated that the camera updates are essential to operation of the robot it would seem that the Asus camera is not a practical solution to the localisation problem. Choosing a camera with a wider field of view may be an improvement but will not solve the problem with blocked areas. Alternatively, mounting a camera on the ceiling would reduce the blocked areas but may be detrimental to the user tracking operation. In research it is commonplace to mount a depth sensing camera directly on top of the robot. It may be more useful to take this approach for robot localisation and use another method to track the user.

The robot wheel encoders have severe performance issues with large cumulative errors in the robots internal position estimate unless correction from the camera is available and consistent. Since the robot sometimes operates out of view of the camera these inaccuracies could lead to a highly inaccurate map in areas where the robot sensors are the only source of information. These errors may ultimately prove too great to reliably augment the map using the robot sensors. Again this suggests that a different approach to robot localisation may be required, such as a mounted camera.

The final major weakness of the project is that the robot has not yet been field tested. In field trials of other home service robots the reaction was ambivalent. Clearly user acceptance is vital if the platform is to have any commercial success, and so user trials need to be performed.

8.2. Future Work

There are some notable improvements to be made in the system. In testing it was observed that much of the error from the robots position is as a result of inaccurate turning manoeuvres. This is partly due to the fact that the robot performance varies according to the floor surface, for example it tends to turn too far on a smooth wood floor and not far enough on carpet. Due to this inconsistency the estimation of angle turned can never be completely accurate. A potential improvement would be to have some way of externally monitoring the robots orientation as well as its position. There are several ways this could be achieved: using the camera as already implemented in the system an identifiable pattern could be placed on either side of the robot and the position of each pattern tracked individually to determine the orientation. However this strategy would suffer from the same limitations as the position updates: the robot is often out of view of the camera and receives no correction at these times.

An alternative which would solve the problem of the robot being out of view would be to use a different system for robot localisation. Ultra Wide Band indoor localisation systems such as the Irish DecaWave have been tested in indoor environments to high degrees of accuracy and may be accurate enough to track the orientation by placing sensors on either side of the robot. The camera could remain in use for user tracking or the DecaWave technology could also be used to track users, although this would require the user to wear a tracking device. The Asus camera could also be mounted on the robot as previously described, and use a self-localisation method to determine its position and orientation within the mapped environment. The modular nature of the software means that compnents can be swapped out easily provided that the message format remains the same. This means that alternative tracking methods could be integrated and tested with minimum effort.

There are also improvements to be made in the mapping program. The approach of adding obstacles the robot encounters to the map may be problematic if the obstacles are temporary. A better approach would be to have the map update progressively over time, so that obstacles that are encountered only once will eventually be removed. This could be approached by having all objects encountered by the robot 'fade away' over time, or by having the robot visit the obstacle location to determine whether it is still occupied. Expanding on this idea another possibility would be to use a probabilistic approach where obstacles found by the robot are added to the map, decay over time, and estimates of position are reinforced or weakened each time the robot revisits the cell in question. Such a probabilistic approach could account for the fact that collisions may occur due to error in the robots position estimate, and take into account the proximity of already mapped obstacles to determine the probability that a new one has been encountered.

After these corrections have been implemented the obvious next step for this project is to trial the use cases in an appropriate environment and gather feedback from potential users. As seen in the literature it is often the case that a system is built to sophisticated specifications only to be rejected by potential users, so it would be unwise to continue the project further without user feedback.

9. Conclusion

A navigation strategy was successfully implemented for the robot which showed improved performance over the augmented dead reckoning strategy previously used. The effectiveness of the camera localisation was improved resulting in an overall increase in position accuracy. Different path planning strategies were compared and analysed in terms of their effectiveness for the proposed use cases. Some limitations of the localisation system were noted which may have an adverse impact on the success of the project if deployed in a user trial. For further development of the project, it is recommended that these limitations be addressed and that feedback is collected from potential users.

References

[1] United nations, department of Economic and Social Affairs, Population Division. (2015). World population prospects: The 2015 revision, custom data acquired via website., [Online]. Available: `http://esa.un.org/wpp/` (visited on Mar. 28, 2017).

[2] J. L. Wiles, A. Leibing, N. Guberman, *et al.*, "The meaning of "ageing in place" to older people," *The gerontologist*, gnr098, 2011.

[3] S. Coradeschi, A. Cesta, G. Cortellessa, *et al.*, "Giraffplus: Combining social interaction and long term monitoring for promoting independent living," in *2013 6th International Conference on Human System Interactions (HSI)*, Jun. 2013, pp. 578–585. DOI: `10.1109/HSI.2013.6577883`.

[4] B. Graf, U. Reiser, M. Hägele, *et al.*, "Robotic home assistant care-o-bot x00ae; 3 - product vision and innovation platform," in *2009 IEEE Workshop on Advanced Robotics and its Social Impacts*, Nov. 2009, pp. 139–144. DOI: `10.1109/ARSO.2009.5587059`.

[5] M. Bal, W. Shen, Q. Hao, *et al.*, "Collaborative smart home technologies for senior independent living: A review," in *Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, Jun. 2011, pp. 481–488. DOI: `10.1109/CSCWD.2011.5960116`.

[6] Philips lifeline medical alert product page, [Online]. Available: `https://www.lifeline.philips.com/` (visited on Jan. 11, 2017).

[7] Kinesis qtug - a proven tool for falls risk assessment, (visited on Jan. 10, 2017).

[8] Vivago, living at home longer and safer, [Online]. Available: `https://www.vivago.com/solutions/home-care--private-sector/` (visited on Jan. 14, 2017).

[9] X. H. B. Le, M. D. Mascolo, A. Gouin, *et al.*, "Health smart home for elders - a tool for automatic recognition of activities of daily living," in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug. 2008, pp. 3316–3319. DOI: `10.1109/IEMBS.2008.4649914`.

[10] M. S. Hossain, "Patient status monitoring for smart home healthcare," in *2016 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, Jul. 2016, pp. 1–6. DOI: `10.1109/ICMEW.2016.7574719`.

[11] H. R. Lee, H. Tan, and S. Šabanović, "That robot is not for me: Addressing stereotypes of aging in assistive robot design," in *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Aug. 2016, pp. 312–317. DOI: `10.1109/ROMAN.2016.7745148`.

[12] N. Pavón-Pulido, J. A. López-Riquelme, J. J. Pinuaga-Cascales, *et al.*, "Cybi: A smart companion robot for elderly people: Improving teleoperation and telepresence skills by combining cloud computing technologies and fuzzy logic," in *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*, Apr. 2015, pp. 198–203. DOI: `10.1109/ICARSC.2015.40`.

[13] M. Vincze, W. Zagler, L. Lammer, *et al.*, "Towards a robot for supporting older people to stay longer independent at home," in *ISR/Robotik 2014; 41st International Symposium on Robotics*, Jun. 2014, pp. 1–7.

[14] M. Vincze, D. Fischinger, M. Bajones, *et al.*, "What older adults would like a robot to do in their homes - first results from a user study in the homes of users," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, Jun. 2016, pp. 1–7.

[15] J. Bedkowski, M. Pełka, K. Majek, *et al.*, "Open source robotic 3d mapping framework with ros x2014; robot operating system, pcl x2014; point cloud library and cloud compare," in *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*, Aug. 2015, pp. 644–649. DOI: `10.1109/ICEEI.2015.7352578`.

[16] J. Biswas and M. Veloso, "Depth camera based indoor mobile robot localization and navigation," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 1697–1702. DOI: `10.1109/ICRA.2012.6224766`.

[17] P. Henry, M. Krainin, E. Herbst, *et al.*, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[18] D. Maier, A. Hornung, and M. Bennewitz, "Real-time navigation in 3d environments based on depth camera data," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, Nov. 2012, pp. 692–697. DOI: `10.1109/HUMANOIDS.2012.6651595`.

[19] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1–4. DOI: `10.1109/ICRA.2011.5980567`.

[20] R. Siegwart and I. R. Nourbaksh, *Introduction to autonomous mobile robots.* Massachusetts Institute of Technology, 2004.

[21] A. Patel. (2016). Introduction to a*, [Online]. Available: `http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html` (visited on Jan. 10, 2017).

[22] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, Jul. 1997, pp. 146–151. DOI: `10.1109/CIRA.1997.613851`.

[23] N. Correll, *Introduction to autonomous robots.* 2016. [Online]. Available: `http://correll.cs.colorado.edu/?p=965`.

[24] D. Ferguson and A. Stentz, "The field d* algorithm for improved path planning and replanning in uniform and non-uniform cost environments," 2005.

[25] A. M. Webb, "A semi-autonomous system to aid the elderly," University College Dublin, thesis, 2016.

[26] *iRobot Create 2 Open Interface (OI) Specification based on the iRobot Roomba 600*, 2015.

[27] M. Quigley, B. Gerkey, and W. D. Smart, *Programming robots with ros, A practical introduction to the robot operating system.* O'Reilly, 2015.

Appendices

A. Base station installations

The installation guides in this section assume a Linux Ubuntu distribution and the availability of the following programs:

1. GIT
2. CMake
3. GCC or similar compiler, included as standard with Ubuntu
4. Python 2.7.x, included as standard with Ubuntu

To confirm/install these packages:

```
~$ sudo apt-get install git cmake gcc python
```

*This guide is adapted from [25]

A.1. OpenNI/NiTE

First download the following libraries, available at `http://www.openni.ru/openni-sdk/openni-sdk-history-2/`

1. OpenNi 1.5.x
2. Primesense OpenNI compliant sensor drivers v5.1.6.6
3. NiTE 1.5.x

These libraries should be installed in the order listed. To verify functionality of the OpenNI framework connect the Asus or similar OpenNI compliant sensor, navigate to the directory OpenNI-Bin-Dev-Linux-x64-version $andenterthefollowingcommands$:

```
~$ cd Samples/Bin/x64-Release
~$ ./NiHandTracker
```

A window will appear showing the sensors depth camera output. If a subject walks in front of the camera and waves the camera should begin tracking their hand using a green trace of the hands position.

Note: these libraries have been deprecated as of Primesenses acquisition by Apple Inc. They are however since openly available either packaged with primesense cameras such as the Asus Xtion Pro Live or online for download.

Next PyOpenNI should be installed to allow for python controlled user tracking. As this module is written in C++ and is platform dependent installation is more involved than a normal python module. First execute the following command to install dependencies:

```
    ~$ sudo apt-get install cmake build-essential git-core python-dev libboost-pytho
```

Note: Boost python is quite large may take a long time to install.

Next use GIT and CMake to retrieve source code and build PyOpenNI:

```
    ~$ git clone https://github.com/jmendeth/PyOpenNI.git
    ~$ mkdir PyOpenNI-build
    ~$ cd PyOpenNI-build
    ~$ cmake ../PyOpenNI
    ~$ make
```

Upon completion the module should be stored at lib/openni. Copy this new module into your Python module library to use it. Verify the install by opening a python terminal and importing the module.

```
  >>> from openni import *
```

Details about functions and the module itself may be found using

```
>>> help(openni) #Help for the whole module
>>> help(openni.Context) #Help for a specific class
>>> help(openni.Context.init) #Help for a specific function
```

Additionally a number of sample scripts are provided in the “PyOpenNI/examples” directory.

A.2. OpenCV

First instal dependencies for OpenCV:

1. GIT, CMake -covered in previous step.
2. NumPy and SciPy python modules
3. - GTK+2.x or higher, including headers to display images on screen
4. libav development packages for processing video streams
5. Image I/O packages for handling various file formats

Install the packages by running the following lines of code. `~$ pip install numpy scipy`

```
    ~$ sudo apt-get install build-essential pkg-config
    ~$ sudo apt-get install libgtk2.0-dev
    ~$ sudo apt-get libavcodec-dev libavformat-dev libswscale-dev
    ~$ sudo apt-get install libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev
```

```
libjasper-dev libdc1394-22-dev
```

Next acquire the latest stable OpenCV version by cloning their GIT repository and the contributions repository. Ensure the same version is checked out in each case. ~$ cd

```
    ~$ git clone https://github.com/Itseez/opencv.git
    ~$ cd opencv
    ~$ git checkout 3.1.0
    ~$ cd
    ~$ git clone https://github.com/Itseez/opencv_contrib.git
    ~$ cd opencv_contrib
    ~$ git checkout 3.1.0
```

Next build OpenCV using CMake ~$ cd /opencv

```
    ~$ mkdir build && cd build
    ~$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D
OPENCV_EXTRA_MODULES_PATH= /opencv_contrib/modules -D BUILD_EXAMPLES=ON
    ~$ make {j4
```

Note:Set "-jx" where x is the number of cores available on your computer to speed up compiling. ~$ sudo make install

```
    ~$ sudo ldconfig
```

Once this completes without error OpenCV has been installed successfully. This can be verified by opening a python terminal and attempting to import it.

```
>>> import cv2
>>> cv2.__version__
>>> '3.1.0'
```

For further verification a number of example scripts can be found at http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html

B. Robot setup

B.1. Getting Started

To connect to the Edison board: `~$ sudo apt-get install screen`
Connect both microUSB ports to laptop or connect power supply to Roomba serial port and outer microUSB (port J3) to laptop. Then `~$ screen /dev/ttyUSB0 115200` and press enter to be prompted for a username and password. If connection fails run

```
~$ ls /dev/ttyUSB*
```

to confirm the port is correct.

To setup wifi: `root@roomba:~# configure_edison --wifi`

and follow instructions to connect to your local network.

Once connected to WiFi the board can be reached via ssh. From your screen session run `ifconfig` to determine the IP address. `~$ ssh root@[edison IP address]`

To setup from scratch:

Note: these steps will erase all data from the edison board.

Install latest copy of Yocto from the Intel website. Tutorial at `https://software.intel.com/en-us/flashing-firmware-on-your-intel-edison-board-linux`

Connect to edison via USB cable and screen as above and run `configure_edison` to setup user, password and WiFi connection. **Remember your password.** There is no password recovery available, in the case of a forgotten password the board will need to be flashed and all data will be erased.

Code for this project is available at `https://github.com/emmafoley2/roomba_navigation` the script `enable_serial.sh` is required to send commands via the serial port.

To load a script on boot: Navigate to the /etc/ directory and create a directory named "init.d". This is the directory the Edison will check for scripts to run at boot. Create the script and modify the permissions.

```
root@roomba:~/etc/init.d# vim newscript.sh
root@roomba:~/etc/init.d# chmod +x newscript.sh
```

Then enable this script to run on bootup by issuing the following command:

```
root@roomba:~/etc/init.d# update-rc.d newscript.sh defaults
```

Verify the script is functioning by rebooting your Edison. Wait for the startup process to begin and then SSH into your device. You should see your script running as a process.

B.2. Pyroomba

Pyroomba is a user made library which provides a python execution of most functions detailed in the Create2 Open Interface. It is available for free download at `http://jon.github.io/pyroomba/`

```
    ~$ git clone git://github.com/jon/pyroomba
```

Once the library is downloaded navigate to the "pyroomba" directory containing setup.py and enter the following install command:

```
    ~$ python setup.py install
```

Assuming there are no errors you should be able to import pyroomba as normal via a python terminal.

```
  import pyroomba
```

For practical verification connect a Roomba platform via serial port and run katamari.py. If there are no problems and the correct serial port is entered a chip-tune should be played by the Roomba. Note: The device name for the Edison serial port is "/dev/ttyMFD1". Modify the test scripts as necessary.

B.3. ROS

Instructions for installing ROS on the base system can be found at `http://wiki.ros.org/kinetic/Installation/Ubuntu`.

Installation on the Edison is more involved as Yocto is not officially recognised by ROS. A minimal installation is detailed here as adapted from the tutorial at `https://github.com/moriarty/ros-ev3/blob/master/brickstrap-build-status.md`, intended for Debian Jessie.

Update repositories and install dependencies:

```
    root@roomba:~# opkg update
    root@roomba:~# opkg install unzip bzip2 lz4 lz4-dev libtinyxml-dev
```

Upgrade pip:

```
    root@roomba:~# python -m pip install -U pip
```

Install dependencies:

```
    root@roomba:~# pip install empy
```

Install ROS packages:

```
    root@roomba:~# pip install -U rosdep rosinstall_generator wstool rosinstall
catkin_pkg rospkg
```

Download and install sbcl:

```
    root@roomba:~# wget http://netcologne.dl.sourceforge.net/project/sbcl/
..  sbcl/1.2.7/sbcl-1.2.7-armel-linux-binary.tar.bz2
    root@roomba:~# tar -xjf sbcl-1.2.7-armel-linux-binary.tar.bz2
    root@roomba:~# cd sbcl-1.2.7-armel-linux
    root@roomba:~sbcl-1.2.7-armel-linux# INSTALL_ROOT=/usr/local sh install.sh
    root@roomba:~sbcl-1.2.7-armel-linux# cd ..
```
Set OS environment variable as ROS will not detect Yocto automatically.
```
    root@roomba:~# export ROS_OS_OVERRIDE=3.10.98-poky-edison+
```
Initialise rosdep:
```
    root@roomba:~# rosdep init
```
Add this repository to the list of sources. Open:
```
    root@roomba:~# vim /etc/ros/rosdep/sources.list.d/20-default.list
```
Add the following line in the os-specific listing section
```
yaml https://raw.githubusercontent.com/moriarty/ros-ev3/master/ev3dev.yaml
```
Update rosdep
```
    root@roomba:~# rosdep update
```
Create and switch to directory ros_comm; then create the rosinstall file, initialize the ros workspace, and check the ros dependencies are all met. (Install is for ros_comm,common_msgs and rospy packages).
```
    root@roomba:~# mkdir ros_comm && cd ros_comm
    root@roomba:~ros_comm# rosinstall_generator ros_comm common_msgs rospy
--rosdistro indigo --deps --wet-only --tar > indigo-ros_comm-wet.rosinstall
    root@roomba:~ros_comm# wstool init src indigo-ros_comm-wet.rosinstall
    root@roomba:~ros_comm# rosdep check --from-paths src --ignore-src --rosdistro
indigo -y
```
Install libconsole-bridge
```
    root@roomba:~ros_comm# git clone https://github.com/ros/console_bridge
    root@roomba:~ros_comm# cd console_bridge
    root@roomba:~ros_comm\console_bridge# cmake src
    root@roomba:~ros_comm\console_bridge# make install
    root@roomba:~ros_comm\console_bridge# export CMAKE_PREFIX_PATH=/home/
..  root/ros_comm/console_bridge
```
Now install ROS: (may take some time!)
```
    root@roomba:~ros_comm# ./src/catkin/bin/catkin_make_isolated --install
--install-space /opt/ros/indigo -DCMAKE_BUILD_TYPE=Release
```
finally,
```
    root@roomba:~# source /opt/ros/indigo/setup.bash
```
Now your ROS installation is ready for testing. On both robot and host machine, run
```
    root@roomba:~# export ROS_MASTER_URI=http://[hostname or host IP]:11311
```
Any port can be used but 11311 is common. To avoid having to do this every time you may find it easier to add this line to your .bashrc file. Simple example talker and listener programs can be found in [27] to test your system; if this fails check your firewall settings and try the troubleshooting steps at the ros wiki.

[27] and the ros documentation at http://wiki.ros.org are great resources on the usage of ROS.