Intelligent Robotics Exercise 3

'Call a meeting'

UNIVERSITY OF BIRMINGHAM

Team Leonard

Matthew Flint - 1247903 - mxf203@bham.ac.uk
Laurence Stokes - 0942846 - lls046@cs.bham.ac.uk
Robert Minford - 1189213 - rxm113@cs.bham.ac.uk
Alexander Bor - 1241885 - ahb285@cs.bham.ac.uk

This work was conducted as part of the requirements of the Module 06-13520 Intelligent Robotics of the Computer Science department at the University of Birmingham, UK, and is submitted in accordance with the regulations of the University's code of conduct.

January 7, 2015

1 Abstract

Mobile robots are regularly used in many applications: from aiding disaster recovery efforts in mines and after earthquakes, to military uses such as roadside bomb detection. Recently, they have even been employed to explore the surface of Mars. This paper presents the design and development of a Mobile Robot inspired by the AAAI Mobile Robotics Competition, 1996: Call a Meeting, which stressed navigation and planning. For our solution to the competition, we present an autonomous Mobile Robot that (a) searches for and finds an empty room (in terms of human presence) b) detects a human in a different room c) directs the human to the empty room. Our setup included a Pioneer 3-DX mobile robot, a Hokuyo URG-04LX laser sensor and a Toshiba i3 / 2GB RAM laptop. Notable features of our solution include a behavioural-based hybrid control architecture, ACML localization and a laser-based leg detection system for identifying persons.

Contents

2 Introduction

According to the Ronald Arkin, the author of Behaviour-Based Robotics, an Intelligent Robot is a machine able to extract information from its environment and use knowledge about its world to move safely in a meaningful and purposive manner [1]. Mobile robots (robots that are capable of locomotion), are one such example of an intelligent robot. They are regularly used in many applications: from aiding disaster recovery efforts in mines and after earthquakes, to military uses such as roadside bomb detection. Recently, mobile robots have been developed for consumer applications, such as the Roomba, and wheeled mobile robots have even been developed to explore the surface of Mars and are poised to return to the moon [2].

The ecological approach to Robot design is one in which the Robots environment and goals are the guiding principle of the design process [1]. This paper presents an ecological approach to the design and analysis of a mobile robot to solve the problems outlined by the AAAI Mobile Robotics Competition,1996: Call a Meeting.

2.1 Background and Prerequisites

This paper assumes very basic familiarity with the principles of robotics; key elements will be elucidated in laymans terms as appropriate. To the extent that it is needed to describe necessary algorithms, mathematical notation (to the level of an undergraduate engineering or science course) will also be presented.

2.2 Outline of the paper

Section 2 presents a review of relevant literature as it pertains to robot control architectures, localisation methods, motion planning, and feature recognition; starting from the broad and converging to those areas specific to the problem domain. The design of the solution our team adopted is then detailed and analysed in Section 3 and experimentation and results are presented in Section 4. Section 5 concludes with a brief concluding discussion of the task, directions for future teams, and acknowledgements to those individuals whose guidance and advice was invaluable during the design process.

---

3 Literature Review

3.1 The History of Robotic Control

Many different techniques and approaches for robotic control have been developed (Arkin, 1998). The robotic control system is, in fact, a continuous spectrum that ranges from purely deliberative reasoning at one extreme to purely reactive control at the other, with the myriad existing architectures fitting somewhere on the spectrum.

3.1.1 Hierarchical Paradigm

The foremost paradigm of robotic control, the deliberative/hierarchical paradigm, focuses primarily on the planning stage of the behavioural cycle. Robots developed with a hierarchical architecture are essentially reflex agents [3], selecting actions from rule matches on the current perceptions from sensory input. This approach typically employs a top-down view to design; an end-goal is realised and a sequence of modules are developed. These modules work to read values from the sensors available to gauge the proximate environment (perception), devise strategies to perform the desired behaviours given said perception (cognition), and then compose the signals that control the actuators to execute these behaviours to achieve the end goal (action).

3.1.2 Reactive Paradigm

The initial dominance enjoyed by the hierarchical paradigm was largely a result of the sphere of influence traditional Artificial Intelligence (AI) exerted over the subdiscipline of Robotics. The prevailing zeitgeist in AI at the time was the notion that knowledge, and knowledge representation, are central to intelligence [1]. Reactive control was born out of a shift against these traditions. Stellar to the reactive paradigm is the idea that the world is fundamentally unknown and changing, and that explicit symbolic representations of the world are simply unattainable. Arkin (1998, p. 24) describes reactive control as a technique for tightly coupling perception and action, typically in the context of motor behaviours, to produce timely and robotic response in dynamic and unstructured worlds [1].

The highlights of the reactive paradigm include its basic building block - a behavior (a direct mapping of sensory inputs to the corresponding motor actions) and the removal of the planning stage from the Sense-Plan-Act triad [1]. In reactive systems, complex behaviours emerge as a consequence of the interaction of many smaller, individual behaviours, all working towards a global maxima (known as emergent behaviour). These behaviours can occur sequentially or concurrently, and each sensor is mapped to its own behaviour: essentially a stimulus/response pair. The set of behaviours to be executed are determined by the Robots immediate environmental context and as a result of the modularity of the behaviours, reactive systems are inherently modular from a software development perspective.

3.1.3 Behaviour-Based Robotics

The advent of the reactive paradigm also coincided with the birth of behaviour-based robotics. The behaviour based approach draws inspiration towards robot control from the disciplines of neuroscience, psychology and ethology (the science of animal behaviour), and to fully appreciate behavior-based robotics some background in biological behaviour is necessary.

Biological behaviors have been studied in three broad categories: reflexive behaviors, reactive behaviors and conscious behaviors [14]. Reflexive behaviors are direct stimulus-response mappings in which the response to a particular sensory input is directly wired with a response and carried out without any cognitive involvement; an example in humans being the kneejerk reaction to touching a hot plate. The second category concerns behaviors that are learned over time, such as learning to ride a bicycle or learning to walk in early childhood. Every other behavior that requires conscious thought throughout the time of the action is categorized as conscious behavior.

Behaviour-based robotics is principally concerned with reflexive behaviours, which in robotics nomenclature are labelled reactive. One of the pioneering behavioural-based architectures is subsumption, a reactive approach presented by Rodney Brooks in 1987 [15]. Subsumption builds on layers of competencies (or simply, a collection of behaviors to achieve a specific task) that are vertically arranged in the order of their sophistication. Basic behaviors, such as collision avoidance, are aligned in the lowest layer, while more cognitive behaviors, such as path mapping, are aligned in the higher levels [14]. The higher layers have the ability to inhibit or suppress the behaviors of the lower layers, hence the name subsumption. The layers of behaviour are designed to be ignorant of one another; each layer executes a dedicated behavior and this avoids the need for them to know the complete scenario they are trying to solve.

3.1.4 Hybrid Paradigm

Whilst the reactive paradigm emerged as the solution for the rigidities put forth by the hierarchical paradigm, the need for planning in more complex tasks began to show over time. Hence, roboticists looked into new ways of combining a planning process with reactive behaviors. Thus, a new breed of architecture was born, the Hybrid Deliberative/Reactive paradigm.

The hybrid paradigm altered the Sense-Plan-Act trilogy by allowing the Plan stage to be executed separately from the Sense-Act sequence. Arkin (1998) mentions three ways in which planning and reaction can be integrated [1]:

1. Hierarchical integration of planning and reaction - The planning and reaction layer are aligned vertically with the planning layer providing the information on which the reaction layer acts.
2. Planning to guide reaction - Planning configures and sets parameters for the reaction control system. Execution occurs solely under the reactive system's auspices, with planning occurring both prior to and concurrent with execution.
3. Coupled planning-reaction - Planning and reaction are concurrent activities, each guiding the other.

A 1995 workshop indicated that a multi-layered hybrid architecture comprising of a top-layer planning system and a lower level reactive system is the emerging architecture of choice in robot control design [1].

3.2 Localisation

Localisation, as applied to Robotics, is the difficult task for an autonomous robot to construct or use a map (or floor plan) and correctly position itself within it. There are several methods and technologies available to the end of achieving localisation including, but not limited to, GPS, odometry (dead reckoning[1]) and laser and sonar. In each approach however, improvements in accuracy come at the cost of increasingly expensive hardware and additional processing power.

A robot has two types of information sources in regards to localization: the idiothetic and the allothetic, and most biologically-inspired models for robots use a combination of both [5]. Tracking the number of wheel revolutions of the Robot is an idiothetic source, and whilst it can give the absolute position of the Robot, it can become subject to cumulative error which grows rapidly. Allothetic sources, by contrast, are the sensors of the robot (such as laser and sonar). The problem with allothetic sources is "perceptual aliasing", which is where two different places can be perceived as the same. In an office building, for example, it may be impossible to correctly localise from sensor information alone if all corridors are identical (and thus providing identical or near-identical sensory feedback).

The general localization problem has a number of increasingly difficult problem instances [6]. The first of these is position tracking and in this problem the robot knows its initial position. With position tracking the goal of localization is to maintain knowledge of the robots position whilst it navigates through the environment. Techniques that solve this problem are called tracking or local techniques [7].

The wake-up robot or global positioning problem is more difficult than the position tracking problem, since the robot does not know its initial position and instead has to localize itself from scratch. Thus, it needs to be able to deal with multiple ideas about its possible location. Methods that solve this problem are called global techniques [7].

[1]The process of calculating one's current position by using a previously determined position, or fix, and advancing that position based upon known or estimated speeds over elapsed time and course; for example using the number of wheel revolutions of the robot.

An even harder problem to solve is the kidnapped robot problem, wherein the robot knows where it is localized but is transferred to another location without knowing. The robot thus has to detect that it has been kidnapped and then re-localize. Techniques that solve this problem can also be used to solve the wake-up robot problem, which can be considered a special instance of the kidnapped robot problem [7].

3.2.1 Markov Localisation

The localization problem can itself be described as a bayesian estimation problem: the robot has a belief about where it currently is and the problem consists of computing the probability density over the space of all possible locations. A bayesian framework for so doing is the Markov Localisation framework originally developed by Thrun et al [7]. The Markov framework assigns weights (a probability) to points on the map from sensory and odometry data and updates these as new data is received, forming a combined overall belief in the location of the robot. Markov Localisation techniques often assume the environment is static (the markovian assumption) [8]. This assumption postulates that the robot's location is the only state in the environment which systematically affects sensor readings. More formally, it states that the sensor measurements observed at time t are independent of all other measurements, given that the current state Lt of the world is known. In the case of localization in densely populated environments, this independence assumption is clearly violated when using a static model of the world. [8]

3.2.2 Kalman Filters

Another Bayesian localization algorithm is the Kalman Filter. Kalman Filters operate recursively on streams of noisy input data to produce a statistically optimal estimate of the underlying system state. The state of a system refers to a vector, x, consisting of n variables describing some interesting properties of the system. An example of a state is the location of a robot, consisting of the x and y coordinates, and the orientation O, of a robot.

Kalman-filter based techniques have proven to be robust and accurate for keeping track of a robots position [9,10]. Because of its concise representation (the mean and covariance matrix suffice to describe the entire density) it is also a particularly efficient algorithm [11]. However, in its pure form, the Kalman filter does not correctly handle non-linear or non-Gaussian motion and measurement models, is unable to recover from tracking failures, and can not deal with multi-modal densities as encountered during global localization. Whilst non-linearities, tracking failure and even multi-modal densities can be accommodated using non-optimal extensions of the Kalman filter, most of these difficulties stem from the the restricted Gaussian density assumption inherent in the Kalman filter [11].

3.2.3 Monte Carlo Localization

A final technique to discuss is the (Adaptive) Monte Carlo localization approach (as described by Dieter Fox), which is included in the ROS Navigation framework [12]. This is also the localization approach we adopted for our Robot. The Monte Carlo approach is an technique to determine a Robots position and orientation (pose) using a particle filter, wherein each particle represents a possible pose of the Robot. The rationale for using a particle filter to solve the localization problem is clear. Unlike Kalman Filters, particle filters can approximate a wide variety of probability distributions, not just gaussian distributions. They are also computationally efficient, as they focus resources on regions in state space with high likelihood [13].

The basic principle of particle filters is having a perceptual model, p(zt — xt) that describes the likelihood of making the observation zt, given that the robot is at location xt, for each particle in the state space s. As new sensory and odometry readings are obtained, the model is updated by recursively replacing the particles using a method of probabilistic sampling [16].

The Adaptive extension to Monte Carlo Localization, developed by Dieter Fox, uses KLD-sampling to dynamically update the size of sample sets [17]. The key idea of the KLD-sampling method is to bound the approximation error introduced by the sample-based representation of the particle filter, and is so named because the approximation error is measured by the Kullback-Leibler distance[2] [17]. During the initial global localization, a robot may be highly uncertain of its pose, and thus a large number of samples required to accurately represent its belief. Once it has positioned itself, however, only a small sample set is required for the easier localization problem of position tracking [16]. KLD-sampling adapts the number of samples during the localization process, choosing large sample sets during global localization and small sample sets for position tracking [16].

3.3 Motion Planning

The basic motion planning problem in robotics is the challenge of moving a robot from one position to another whilst avoiding obstacles. It is also referred to as the navigation problem or piano movers problem and many mobile robot navigation problems can be expressed as instances of the basic motion planning problem [19]. There exist a large number of techniques to solve the basic motion planning problem, however not all of them solve the problem in its full generality [19].

In motion planning, a configuration describes the pose of the robot, and the configuration space C is the set of all possible configurations. Every obstacle in the configuration space can be mapped to a subset of C, $C_{obstacle}$. The complement of $C_{obstacle}$ is the free space and labelled C_{free}. The basic motion planning problem is solved by creating a

[2] A measure of how different two probability distributions (over the same event space) are [18].

path between two configurations, qinit and q_{goal}, through a connected region of C_{free} [19].

The first extension to the basic planning problem, and the one that concerns our solution, is the presence of moving obstacles. In this instance, the problem can not be solved by simply constructing a geometric path through C_{free}[19]. Rather, a continuous function of time must be calculated specifying the robots configuration at each instant [19]. This is achieved by adding one dimension representing time, C_T, to the robots configuration space [19].

Grid-Based searches are one technique for solving motion planning problems, and the ROS navigation stack adopts a grid-based search algorithm [20]. Grid-based approaches overlay a grid on the configuration space, and assume each configuration is identified with a grid point. At each grid point, the robot is allowed to move to adjacent grid points as long as the line between them is completely contained within C_{free}.

The specific technique the ROS Navigation stack uses is a grid-based costmap, wherein each cell in the map has one of three states: occupied, free, or unknown, and is assigned a cost (hence the term costmap). Occupied cells are assigned a lethal (infinite) cost, meaning that no part of the robot can step inside that cell. The global planner is given the obstacle and cost information contained in the Costmap, information from the robots localization system, and a goal in the world. From these, it creates a high-level plan for the robot to follow to reach the goal location using the lowest cost route. For speed of processing, the navigation uses an A* algorithm [27]. As the PR2 navigation system (the ROS Navigation system) successfully completed 26.2 miles of autonomous navigation in an office environment, without human intervention [27], it is provenly suitable to solve the Call a Meeting challenge.

3.4 Person Detection

In recent years, the social aspect of service robots has made it clear that they are not only expected to navigate within their environment but also to interact with people so as to provide useful services. Thus, they need to be aware of the presence and position of the people with whom they are interacting [28]. This is, however, a very challenging task; peoples behaviour is often completely unpredictable [28]. Critical to the success of this task is the ability of the robot to identify and interact with people, and, thus, a robust method of so doing is required.

The most common devices used for people identification and tracking are laser sensors and cameras [29]. The solution we adopted is laser-based leg detection system that uses the front-mounted laser scanner and was taken from the thesis work of one of the module demonstrators, Marco Becerra, and is available (in Spanish) at [29]. Marcos work is itself based on the work of Nicola Bellotto and Huosheng Hu, available at [28].

The basic premise of the detection algorithm is to identify typical patterns (relative to particular leg postures) that, in most of the cases, are well distinguishable from the other objects in the environment. These patterns correspond to the following three typical situations: two legs apart (LA), forward straddle (FS), and two legs together or (SL). The first pattern is usually very common in case the person is standing in front of the robot. The second, however, is most likely to happen when the person is walking. The last pattern covers most of the remaining possible postures. The results of the laser scan are preprocessed into a string pattern, and then checked for likeness to any of the typical leg patterns (subject to standard human dimensions - e.g. a leg cannot be 10 meters wide) through regular expression matching.

---

4 Design

This section details and analyzes the design choices we took as a team. All of our choices were mediated through experimentation, reference to the background literature, as well as availability and feasibility.

4.1 Hardware and Software

The Pioneer 3DX (P3-DX) is one of the most popular research robots [21] and the model we used. It is a compact differential-drive mobile robot and is fully assembled with an embedded controller, motors with 500-tick encoders, 19cm wheels, a tough aluminum body, 8 forward-facing ultrasonic (sonar) sensors, 8 optional rear-facing sonar, and 1, 2 or 3 hot-swappable batteries [22]. Because of its modest and balanced size combined with reasonable hardware, it is best suited for indoor navigation [21]. The robot was additionally equipped with a Hokuyo URG-04LX laser sensor, which can report obstacles in ranges from 20mm to 5600mm (with a 1 mm resolution) in a 240°arc (with a 0.36°angular resolution). It has a scanning time of 100ms/scan [23].

The software we used was the Robot Operating System (ROS), a Linux based software framework for operating robots [24]. It is open source and multi-lingual [24], and has language support for C++, Python, Octave, and LISP, and experimental ports for Java and Lua [25]. We used python (2.7) for our programming needs.

The core concepts of ROS are packages, nodes, topics, messages and services. A node is an executable program that takes data from the robot's sensors and passes it to other nodes. Nodes communicate with each other through strictly typed data structures called messages. Messages always travel via special ports called topics, which are strings such as odometry or map. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics [24]. All re-

lated nodes are combined in one package that can easily be compiled and ported to other systems. The packages are necessary to build a complete ROS-based autonomous robot control system [21].

4.2 Control Architecture

For the overall control design we followed a behaviourally inspired hybrid approach. We felt this best fit the problem scope; we knew exactly what tasks we needed to execute (and in what order), and we were dealing with a semi-structured environment (shown in Figure 2). Our challenge was thus to integrate the a-priori world knowledge with reactive behaviours such as obstacle avoidance.

Our hierarchical planner was structured into a series of sequential steps as per the task requirements. These are, in order:

1. Localise in respect to the map
2. Navigate to the hard-coded coordinates of the first room and position to scan the room.
3. Scan the room checking for the presence of people
4. IF person detected and confirmed, goto 6
5. ELSE set first room as the meeting room and goto 9.
6. Navigate to the coordinates of the second room and position to scan the room.
7. IF person detected and confirmed, goto 9.
8. ELSE set second room as the meeting room and goto 2.
9. Roam the area searching for people.
10. Upon detecting a person ask them if they would like to attend the meeting.
11. IF person confirms then drive back to the meeting room then goto 9
12. ELSE goto 9 (continue roaming)

During execution of the plan steps, appropriate reactive behaviours and responses are selected according to the robots peripheral environment as reported by the sensors. The navigation algorithm, for example, selects a point-to-point path consistent with the plan specification and a perceptually running obstacle avoidance algorithm results in the robot avoiding any local and unforeseen obstacles.

4.3 Localization

As a team we decided to adopt the ACML localisation implementation included as part of the ROS Navigation Stack package over a bespoke MCL implementation we had previously developed. As discussed in the background literature, the rationale of using a Monte Carlo approach to localisation is clear: they dont possess the independence assumption of Markov localisation techniques (in their basic implementations), and unlike Kalman filters (again, in their basic implementations), particle filters can approximate a wide variety of probability distributions, not just gaussian distributions. They are also computationally efficient, as they focus resources on regions in state space with high likelihood [13].

The immediate advantage of the ACML implementation over our MCL approach is that it uses adaptive sampling (adjusting the sample size during the localisation process). Experimental results (See 5.2) also show the ROS ACML implementation being much quicker and more robust than our bespoke MCL algorithm. Experimental data (5.2) also informed design decisions on the respective parameter configurations for the odometry drift, rotation, and translation noise.

4.4 Navigation and Exploration

For Navigation, we used the ROS Navigation Stack, described as a 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base [26]. It uses ACML for localisation and position tracking (when supplied with an a-priori map), and a grid-based costmap for motion planning [27]. It was developed for an autonomous mobile robot, PR2, to robustly navigate through a real-world dynamic office environment. PR2 achieved 26.2 miles of autonomous navigation [27]. The Navigation Stack also boasts several recovery strategies: when a robot finds itself entrapped by obstacles, for example, a number of increasingly aggressive recovery behaviors are executed to attempt to clear out space [27].

Considering its demonstrated robustness and the problem domain the ROS navigation stack was developed for (which is very similar to ours), there was no need to research other navigation methods at the expense of losing time to solve the other sub-tasks.

We tested two possible exploration strategies for our robot. One strategy was allowing the robot to free roam and the other was setting a fixed path through hard-coded points. Through qualitative experimentation (presented in 4.3) we found the fixed point approach to be more suited to the task. Whilst it covered less area in absolute terms than the random free roaming navigation strategy, the areas covered were more sensible to the problem scope (such as corridors as opposed to corners and dead-ends) and also covered significantly faster.

4.5 Obstacle Avoidance

Obstacle avoidance is included as default in the ROS Navigation Stack and has been proven to be robust (which is corroborated by our experiment data in Section 5). Obstacles are inferred strictly from geometric information from range sensors that can produce point clouds [27]. In our case, this was the front-mounted Hokuyo laser.

4.6 Person Detection

As per the sub-task of identifying persons, we adopted a laser based leg-detection system as developed by Macro Becerra, a demonstrator for this module. The algorithm is detailed thoroughly in the Literature Review section 3.4.

Whilst we implemented the basic algorithm as described in in section 3.4 in our robot, there are several improvements we could have considered. One such improvement would be integrating face detection with leg detection to arrive at a more accurate conclusion on the presence of a person. Another (more difficult) improvement would have been to track the person/s in real time. This would have the immediate advantage of eliminating some false positives from the leg and face detection (i.e. a person can't just appear/disappear in the middle of the scene). Tracking a walking person, however, is not a trivial task. There are many factors which may influence the human trajectories, such as the environment, the number of people, and the interactions among them. In fact, the unpredictability of such behaviors have been modeled by some researchers as Brownian motion [29]. Typically, Kalman Filters are used to track persons, as they are fundamentally designed to predict a processs state and then use measurements to correct or update these predictions. In this regard, their purpose and function in tracking persons is not at all dissimilar to that in motion tracking.

---

5 Experimental Results

In order measure performance and to best guide the design choices available, we conducted several experiments on the key components of the task. This section details and discusses the methods, results, and implications of the testing.

5.1 Odometry Noise

The Pioneer P3-DX robot is subject to marginal noise in its odometry readings. Discrepancies in the reported position of the robot from the localisation algorithm versus the reported position from the odometer give an indication of the level of noise to expect (and thus to account for). It should be noted, however, that our testing was subject to human error. Eliminating this error vector would have demanded more complex testing methods stretching beyond the scope of the exercise.

The motor of the Robot has a short delay before starting. We tested this to be consistent at 2 seconds (+- 0.1) and accounted for it in any movement-based tests by initializing the motor (by sending a spurious non-movement command) prior to sending a movement command.

5.1.1 Translation Noise

The translation noise was measured by driving the robot forwards with a set velocity and time. The results of the 30 test instances indicated that the translation noise was on average 0.7%, with this value fairly consistent across a variety of time values when using a value of 0.25 for speed.

Figure 2: Translation noise test 1

Figure 3: Translation noise test 2

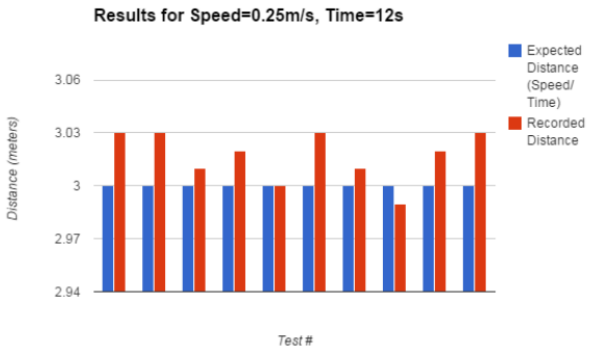


Figure 4: Translation noise test 3

5.1.2 Drift Noise

Drift noise levels were calculated by instructing the Robot to drive 1, 2, or 3 meters on a predefined straight path, and then measuring the x-axis deviation from that path, repeating the process 20 times for each distance.

Our results showed an average drift per meter of 2.9159 cm/m for the 60 tests at the 3 different distances (2.7125, 2.99 and 3.045 for 3 meters, 2 meters, and 1 meter respectively). This means that for each meter the robot drives forwards, it will on average drift 2.9159cm. The results also suggest that the drift noise decreases (in relative terms) as distance increases. Further testing would be required to test this hypothesis, however, as our results could be specific to the distances we tested (i.e. the phenomena is only observable over small distances ¡= 3 meters), or simply anomalous to our small data set of 60 test instances. Figures 4.2 shows the results.

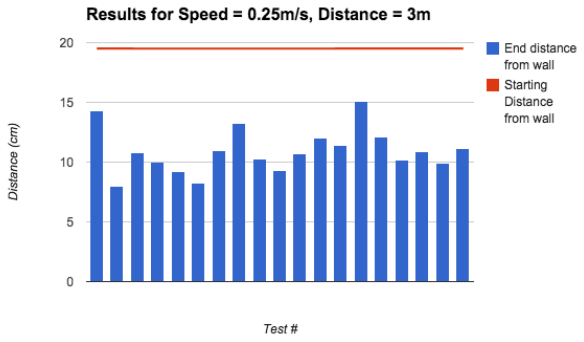


Figure 5: Drift noise test 1

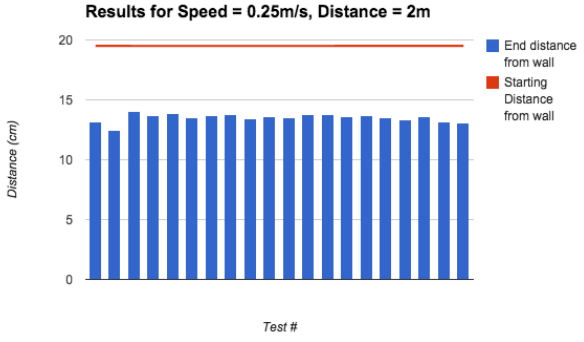


Figure 6: Drift noise test 2

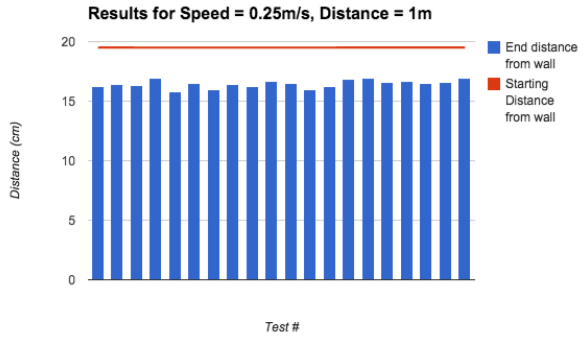


Figure 7: Drift noise test 3

5.1.3 Rotation Noise

To test rotation noise we set the Robot to turn a set value, pi radians (180 degrees), at a set angular.z velocity (turning speed) of 4/pi radians/second (meaning it was turning at 45 degrees/second). The actual number of degrees turned over the 4 seconds was then measured and the difference compared. Over 30 test instances the average difference was found to be 00.55% with a variance of +-4 degrees.

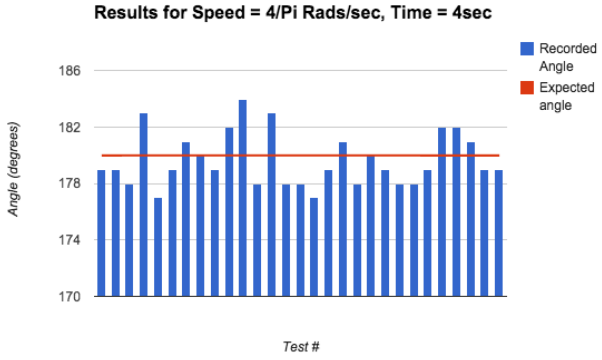


Figure 8: Rotation noise test 1

5.2 Localisation Performance

Whilst we adopted the ACML localisation as part of the ROS package over a bespoke MCL implementation we had previously developed, we still tested our bespoke implementation for its robustness, and whether it could indeed be successfully used in a non-trivial application. Both localisation methods were analysed by the speed at which they could localise the robot as it travelled to a pre-set point. For the sake of brevity, the testing was conducted in the stage simulation.

Our results (Figure 4.4) show that the ACML implementation outperforms our MCL implementation significantly (35% quicker in achieving localisation) and with a smaller

deviation in the results (a standard deviation of 4.06 versus 5.81). Our MCL implementation, however, always succeeded in localising the robot, and time constraints notwithstanding, could be used in place of the ACML implementation.

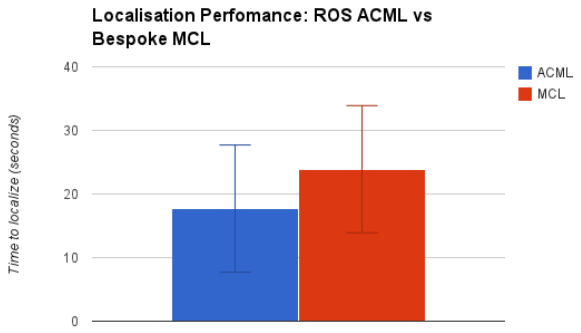


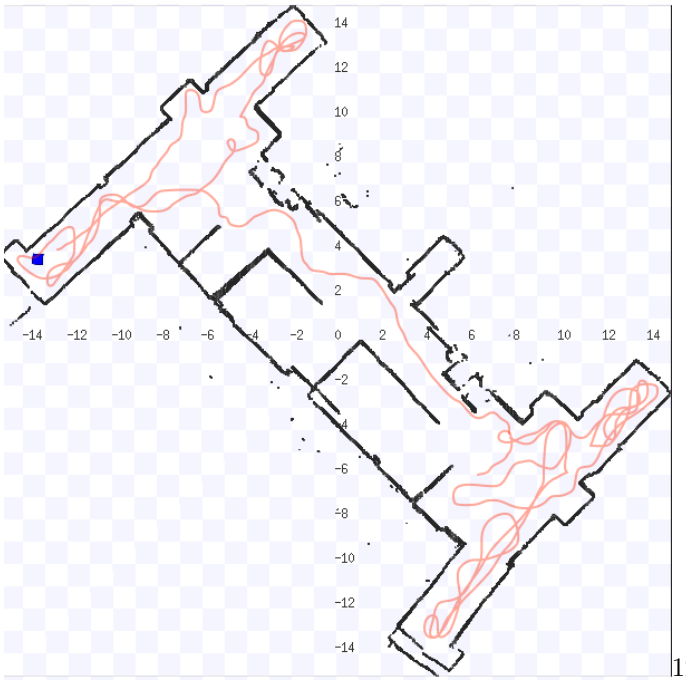
Figure 9: Performance of ROS ACML vs our bespoke ACML implementation

5.3 Exploration

We tested two possible exploration strategies for our robot. One strategy was allowing the robot to free roam and the other was setting a fixed path through hard-coded points.

5.3.1 Random Exploration

To test the feasibility of the random navigation strategy we allowed the robot to freely roam across the floor. In free roaming it discovered more areas, and in more detail, than the specific point navigation strategy. However, as seen from Figure 9, the exploration is stochastic and inefficient: the robot often looped back on itself, at times circling for several minutes.



5.3.2 Specific Point Exploration

In the specific point exploration strategy we gave our robot 6 points in which to roam. Though significantly reducing the overall exploration range, it did make identifying people much faster, as only those map regions with high likelihood to have a person were explored and areas such as corners ignored. Additionally, the robot did not loop back on itself or otherwise behave inefficiently. We selected this strategy in our implementation.

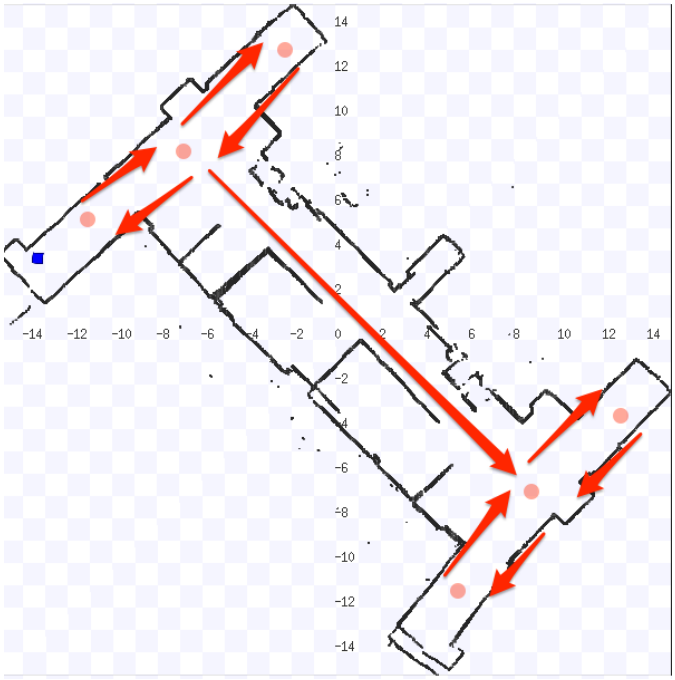


Figure 11: Plotted exploration route

5.4 Human Detection Testing

5.4.1 Leg Detector Average

To measure the best value for the leg detector average we let the robot roam along a set path using the navigation stack as the driver, recording the averages calculated from the number of legs detected/the number of readings taken. This provided an average value to increase the effectiveness of our leg detection. We measured and modified the amount of readings taken before calculating the average and the threshold value to cause the Robot to believe a person was present to find the most suitable parameters for the task.

Test	Average number	Amount of Readings taken	Observed Result
1	0.5	50000	Many False Positives :Robot thinks people are there when they are not.
2	0.5	100000	Many False Positives: Robot thinks people are there when they are no. Average is lower.
3	0.5	150000	Many False Positives: Robot thinks people are there when they are not, but average is lower again. Average appears to be too low.
4	0.75	50000	Less false positives, but average does not differ too much when someone is in front of the Robot
5	0.75	100000	Very similar to test #4, but more responsive when used with more readings. Average may need to be increased.
6	0.75	150000	Again very similar to test #4, but taking too long between each assumption.
7	1	50000	Detected the legs and low positives. Average consistently higher due to low number of readings taken
8	1	100000	Timely and accurate assumption.
9	1	150000	Low false positives but took too long between readings and would sometimes overshoot people while driving.
10	1.25	50000	Higher average decreased the false positives but meant that if no false positives were detected then the average would not be above the threshold.
11	1.25	100000	A higher number of readings seemed to decrease the average number further, no detections if there are no false positives.
12	1.25	150000	A higher number of readings seemed to decrease the average number further, no detections if there are no false positives.
13	1.5	50000	Significantly less false positives, higher average seems to lead to less humans being falsely detected.
14	1.5	100000	The higher number of readings meant the average was lower. It rarely reached over the threshold.
15	1.5	150000	The higher number of readings meant the average was lower. It rarely reached over the threshold.

Based on the experimental data we chose to use 100,000 readings with an average of 1. This gave us the most accurate detection rate that someone was present whilst also performing in a reasonable amount of time. Using more readings, whilst producing a more accurate result, would not have yielded a timely enough result. The average of 1 means that the robot has to have seen at least one person during the entirety of its scan. This dramatically reduced the false positive rate while exploring; whilst the leg detector might produce false positives at a given point it is unlikely to detect it over its entire scan.

5.4.2 Leg Detector Parameters

To determine the best parameters for the Leg Detection algorithm (largely to eliminate the large number of false positives which it originally suffered), we placed the robot in a fixed, stationary position, looking in to a corridor (the most controlled setting we could find). We then ran our

tests, first seeing if the values detected any false positives. We then changed the values to try and reduce these false positives and then introduced a person into the scanners range to check that they were accurately detected.

AMin	AMax	BMin	BMax	CMin	CMax	Observed Result
0.1	0.2	0	0.4	0.1	0.4	Robot seemed very sensitive, was set off by many obstacles.
0.1	0.15	0	0.4	0.1	0.4	This seemed to have no effect, still very sensitive.
0.1	0.3	0	0.4	0.1	0.4	This seemed to increase the number of false positives detected.
0.05	0.2	0	0.4	0.1	0.4	This increased the number of false positives detected.
0	0.2	0	0.4	0.1	0.4	This dramatically increased the number of false positives detected
0.1	0.2	0	0.4	0.1	0.4	These values seem to yield the best results.
0.1	0.2	0.2	0.4	0.1	0.4	This did not reduce the false positives at all.
0.1	0.2	0.1	0.4	0.1	0.4	This did not reduce the false positives at all.
0.1	0.2	0	0.3	0.1	0.4	Reduction in false positives, specifically the doorway posts.
0.1	0.2	0	0.2	0.1	0.4	No longer detected people as well, missed people standing in a 'normal' pose.
0.1	0.2	0	0.3	0.1	0.4	These values seem to yield the best results.
0.1	0.2	0	0.3	0.2	0.4	Did not seem to affect the number of false positives.
0.1	0.2	0	0.3	0.3	0.4	Did not seem to affect the number of false positives.
0.1	0.2	0	0.3	0.3	0.5	Did not seem to affect the number of false positives.
0.1	0.2	0	0.3	0.35	0.4	Did not seem to affect the number of false positives.
0.1	0.2	0	0.3	1000	1000	This is to stop it having an effect at all. These seems to make the robot a lot more reliable.

The parameters we chose are highlighted in bold with qualitative justification presented in the observed result column. After our tests were complete we ran the robot through the full program to check that these results behaved as expected in the viva environment.

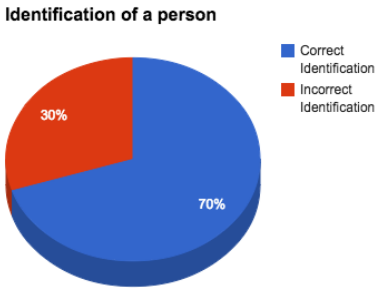


Figure 12: Leg detector detection rate of person

5.4.3 Leg Detector detection rate

The leg detection rate was a binary pass:fail test. We set the robot at a static point looking into a room and then have it scan for a person in that room, measuring the number of correct (detecting a person when a person is present or vice versa) and incorrect results.

The leg detection yielded an accuracy rate of 70 percent. We feel this rate is acceptable given the environment of the room and the way the leg detector algorithm works. Nevertheless, we decided that we needed human confirmation (a button press) that a human was genuinely in the room.

5.5 Obstacle Detection

To test the accuracy of the obstacle detection we ran the robot through a set route, placing a wooden board across the path at different points.

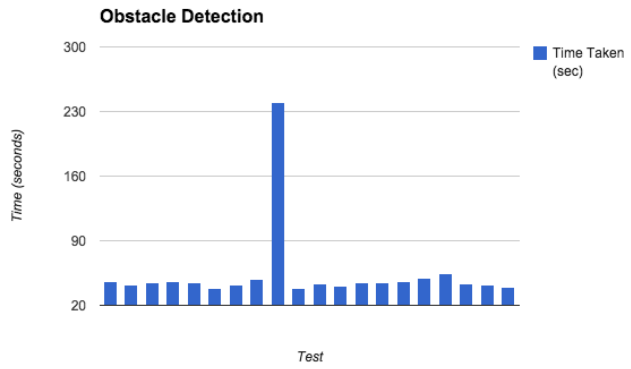


Figure 13: Leg detector detection rate of person

As evidenced by Figure x, 98% of the time the robot avoided the board. Difficulties in avoidance arose when the robot became 'trapped' in a corner created by the board and walls.

5.6 Overall Performance Testing

The overall performance tests were full-run tests scored by a binary pass:fail metric. Over 10 test instances the Robot robustly succeeded in the full run 8 times.

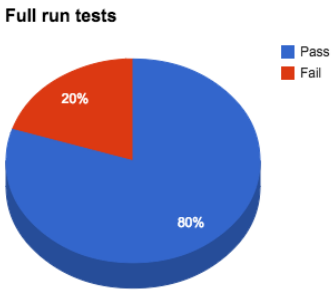


Figure 14: Leg detector detection rate of person

The feedback from the full-run tests was reassuring; the tests completed successfully 80% of the time. The reasons for the tests failing were one instance of an object entering the domain and the robot failing to stop in time as it was going too fast (subsequently fixed). The second time the robot got 'stuck' in a corner which, given enough time, we believe it may have eventually worked its self free.

6 Discussion

6.1 Conclusions and Future Work

We developed a robot that can demonstrably and robustly fulfill all the criteria as set by the task, Call a Meeting. All of our design choices were mediated through experimentation, reference to background literature, as well as availability and feasibility of resources. The success of the project was the result of adopting and integrating relevant and well-developed modules from the ROS framework in combination with an overall planning scheme to solve the complete task.

Nonetheless, there are several areas the project could have been improved, including: More thorough person detection through combining the leg detection module with a slower, facial recognition method (once the leg detector records a possible person) to reduce false positives and increase certainty of the presence of a person. In fact, we did research and consider this option as a team (and had a working facial recognition module) but did not achieve integration within sufficient time. We felt that it was better to leave it out and maintain the existing (and robust) solution we had that used only leg detection. Confirmation of the presence of a person (or lack thereof) could have been achieved using speech recognition as opposed to a keypress from the person (such as say yes if you wish to come to the meeting). The robot could have also scanned the room/s more thoroughly for the presence of a person; in its current implementation it enters the room, and then scans a 180 degree arc from an angle. In so doing, the corners behind the robot are missed. We rationalized as a team that it would be extremely unlikely for someone to be stood in the corner of a room behind where we set the robot to scan (at an angle from the entrance door), but adding a more thorough room scan is definitely a strong consideration for potential future improvements.

6.2 Acknowledgements

The Authors of this paper would like to thank the ROS development team for providing a stable platform for robotics development. Also, we thank all the module convenors and demonstrators of the Intelligent Robotics Module at the University of Birmingham, UK, for their hard work and guidance throughout the entire module and this task in particular.

7 References

1. Arkin, R. C. (1998).
Behavior-based robotics. MIT press, p. 1, 15, 20, 24, 212-213

2. Mark W. Spong and Masayuki Fujita, 2004.
Control In Robotics. IEEE Control Systems Society.
Available: ieeecss.org/sites/ieeecss.org/files/documents/IoCT-Part1-04Robotics.pdf [Online]
(Accessed 11/11/2014)

3. Stuart J. Russell and Peter Norvig. 2003.
Artificial Intelligence: A Modern Approach (2 ed.). Pearson Education.

4. John Kelleher,
Robot Control Architectures, [DT211/1 Applied Computing, DIT School of Computing].
Available: http://www.comp.dit.ie/jkelleher/appliedcomputing/week8/RCA.pdf [Online]
(Accessed 11/11/2014)

5. Margaret E. Jefferies, Wai-Kiang Yeap,
Robotics and Cognitive Approaches to Spatial Mapping,
Volume 38 of Springer Tracts in Advanced Robotics, ISSN 1610-7438. p 175.

6. S. Thrun, D. Fox, W. Burgard, and F. Dellaert,
Robust monte carlo localization for mobile robots. Artificial Intelligence 128,1-2 (2001), 99141.
Available: http://citeseer.nj.nec.com/thrun01robust.html [Online]
(Accessed 20/11/2014)

7. S. Thrun, D. Fox, W. Burgard,
Markov localisation for mobile robots in dynamic environments. Journal of Artificial Intelligence Research 11 (1999), 391-427.
Available: http://citeseer.nj.nec.com/fox99markov.html [Online]
(Accessed 20/11/2014)

8. D. Fox,
Markov Localisation (1999).
Available at: cs.cmu.edu/afs/cs/project/jair/pub/volume11/fox99a-html/node2.html [Online]
(Accessed 25/11/2014)

9. J. J. Leonard and H. F. Durrant-Whyte,
Directed Sonar Sensing for Mobile Robot Navigation, Kluwer Academic, Boston, 1992.

10. B. Schiele and J. L. Crowley,
A comparison of position estimation techniques using occupancy grids, in Proceedings of IEEE Conference on Robotics and Automation (ICRA), vol. 2, pp. 1628 to 1634, 1994.

11. S. Thrun, D. Fox, W. Burgard, and F. Dellaert,
Monte carlo localization for mobile robots. Institute of Computer Science III, University of Bonn, D-53117 Bonn.
Available at: http://www.cc.gatech.edu/ dellaert/pub/Dellaert99icra.pdf [Online]
(Accessed 25/11/2014)

12. ROS.org,
AMCL Package Summary,
Available: http://wiki.ros.org/amcl [Online]
(Accessed 21/10/2014)

13. D. Fox et al,
"Particle Filters for Robot Localisation",
Available: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=547840CC2F7B99B [Online]
(Accessed 24/11/2014)

14. R. R. Murphy.
Introduction to AI Robotics. The MIT Press, Cambridge, Massachusetts, 2000.

15. L. De Silva and H. Ekanayake,
Behavior-based Robotics And The Reactive Paradigm - A Survey. University of Colombo School of Computing.
Proceedings of International Workshop on Data Mining and Artificial Intelligence (DMAI 08).

Available: http://www.cs.utah.edu/ãlnds/papers/behavior-robotics-2008.pdf [Online]
(Accessed 27/11/2014)

16. D. Fox,
Adapting the Sample Size in Particle Filters Through KLD-Sampling . Department of Computer Science, University of Washington.
Available: www.cs.washington.edu/robotics/mcl/postscripts/adaptive-ijrr-2003.pdf [Online]
(Accessed 27/11/2014)

17. D.Fox,
KLD-Sampling: Adaptive Particle Filters, Department of Computer Science, University of Washington.
Available: http://papers.nips.cc/paper/1998-kld-sampling-adaptive-particle-filters.pdf [Online]
(Accessed 27/11/2014).

18. D. Pinto,
The Kullbach-Liebler Distance, 2007.
Available: http://www.cs.buap.mx/ dpinto/research/CICLing07-1/Pinto06c/node2.html [Online]
(Accessed 27/11/2014)

19. Jean-Claude Latombe,
'Robot Motion Planning: Edition en anglais'. The Springer International Series in Engineering and Computer Science, Springer Science and Business Media, 1991 p.6, 22

20. D.V. Lu, D. Herschberger, W.D. Smart,
Layered Costmaps for Contest-Sensitive Navigation, Washington University in St. Louis, Department of Computer Science and Engineering.
Available: http://www.cse.wustl.edu/ dvl1/publications/LayeredCostmaps.pdf [Online]
(Accessed 02/12/2014)

21. Zaman, S.; Slany, W.; Steinbauer, G.,
"ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues," Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International , vol., no., pp.1,5, 24-26 April 2011 doi: 10.1109/SIECPC.2011.5876943.
Available: http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=5876943 [Online]
(Accessed 02/12/2014)

22. Pioneer P3-DX,
Adept Mobile Robots,
Available: http://mobilerobots.com/ResearchRobots/PioneerP3DX.aspx [Online]
(Accessed 02/12/2014)

23. Scanning Range Finger (sokuiki sensor),
hokuyo-apt.jpt,
Available: mhttps://www.hokuyo-aut.jp/02sensor/07scanner/urg-04lx.html [Online]
(Accessed 02/12/2014)

24. M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng,
ROS: an open-source Robot Operating System Computer Science Department, University of Southern California.
Available: http://ai.stanford.edu/ ang/papers/icraoss09-ROS.pd [Online]
(Accessed 02/12/2014)

25. What is ROS,
ROS.org,
Available: http://wiki.ros.org/ROS/Introduction [Online]
(Accessed 02/12/2014)

26. Navigation Package Summary,
ROS.org,
Available: http://wiki.ros.org/navigation [Online]
(Accessed 02/12/2014)

27. Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B. and Konolige, K. (2010).
The Office Marathon: Robust navigation in an indoor office environment. 2010 IEEE International Conference on Robotics and Automation.
Available: http://wiki.ros.org/Papers/ICRA2010-Marder-Eppstein?action=Att... [Online]
(Accessed 29/12/2014)

28. Bellotto, N. and Huosheng Hu, (2009).
Multisensor-Based Human Detection and Tracking for Mobile Service Robots. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39(1), pp.167-181.
Available: http://webpages.lincoln.ac.uk/nbellotto/doc/Bellotto2009.pdf [Online]
(Accessed 29/12/2014)

29. Beccera, M. (2012)
"Sistema De Seguimento de Personaspara un Robot Novil de Servicio". Universidad Nacional Autonoma De Mexico.
Available: https://dl.dropboxusercontent.com/u/53733773/web/file... [Online]
(Accessed 29/12/2014)