An Autonomous Meeting Assistant

951926   1001231   1024072   1028907

Abstract—**In this report, we present a system which performs the 1996 AAAI Mobile Robotics Competition task "Call a Meeting"[1]. The robot was tasked with finding a meeting room, ensuring it is empty and bringing a certain number of people to the room within a 15-minute time limit. To complete the task, we use probabilistic roadmapping, Monte Carlo localisation and a face detection module using the ROS framework, installed on a laptop, mounted on a Pioneer 3-DX platform modified with a tripod-mounted Kinect.**

I. INTRODUCTION

Solutions to complex tasks often require the use of multiple techniques to solve different sub-problems, and this task is no exception. We were required to implement a localisation algorithm to determine the position of the robot, a probabilistic roadmap to allow paths to be planned through the space, a navigation algorithm for path planning, a method of exploring the space, and some way of detecting people. We then had to implement a system which would combine all of these separate modules into a single system that would complete the task. In the next section we present some background information about techniques that are often used to solve these sorts of problems. We then provide a detailed description of our system, before evaluating its performance and discussing experimental results. Finally, we conclude and suggest areas for improvement.

II. BACKGROUND

In this section, we discuss briefly the areas of robotics that are relevant to solving the task that has been set. We mention specific algorithms and techniques that we considered using, as well as describing briefly the benefits and drawbacks of some of the techniques.

A. Localisation

The aim of localisation is to obtain an estimate of the position of a mobile robot using sensor data [2]. While it is possible to use odometry data to do so, this data is inherently noisy and prone to error. In particular, odometry errors can arise from changes in the surface the robot is travelling across and the robot's weight. An important point to note is that localisation can be performed with a pre-defined map, as well as by generating a map in real-time. The much more complex latter problem is called simultaneous localisation and mapping (SLAM), which we will not discuss here. See [3] for an introduction to the SLAM problem.

1) Bayes Filter: Many advanced localisation techniques are based on the Bayes filter, shown in Algorithm 1, which uses a belief distribution $bel(x_t)$ to represent the state x_t. The calculation of $bel(x_t)$ at time t is dependent on $bel(x_{t-1})$ at time $t-1$, the last action u_t, and the last measurement z_t. In the first step, called the prediction step, the prior belief $\overline{bel}(x_t)$ is calculated. This step merges two probability distributions; the

prior belief over the previous state x_{t-1}, and the probability of transitioning from that state to the posterior state x_t given that the action u_t was taken. This step does not take into account any measurement taken in the posterior state, predicting using only the knowledge of the action taken. In the second step, the measurement update, the posterior belief is calculated by multiplying the prior belief with the probability of being in the posterior state given that the measurement z_t was observed. The result of this multiplication is generally not a probability, and therefore requires normalisation using some constant α. As there are usually multiple posterior states, x_t is usually a state vector rather than a single state, and so the two steps will be applied multiple times in order to update the belief for each state being considered. The filter is recursive, requiring some idea of the initial belief $bel(x_0)$ at time $t = 0$. The initial belief is either a distribution centred on x_0, in the case where the initial position is known, or a uniform distribution over the space otherwise. As the Bayes filter is not restricted

---

Algorithm 1 Bayes Filter [4]

1: **Algorithm bayesFilter**$(bel(x_{t-1}), u_t, z_t)$
2: **for** all x_t **do**
3: $\quad \overline{bel}(x_t) = \int P(x_t \mid u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$
4: $\quad bel(x_t) = \alpha P(x_t \mid z_t) \overline{bel}(x_t)$
5: **end for**
6: **return** $bel(x_t)$

---

to finite state spaces, it is not possible to implement it for anything other than very simple problems. There are two families of algorithms for localisation, known as *recursive state estimators*, with various properties that permit the use of the Bayes filter in more complex estimation tasks [4].

2) Gaussian Filters: The basic principle of the family of Gaussian filters is the use of multivariate normal distributions, which can be formulated from a mean μ and covariance Σ, to represent belief. As a result, the assumption that the system is a linear Gaussian system is made; the initial belief must be a Gaussian, and both the state transition function and measurement probability must be linear functions. Although Gaussian filters can be extended to non-linear systems, they perform best when the system meets the assumptions made. One of the main advantages of such filters is the computational complexity, which is polynomial with respect to the dimensionality of the state space. The main disadvantage is that Gaussians are unimodal and therefore cannot represent situations in which there are multiple hypotheses; a situation that is often encountered in robotics. Examples include the Kalman filter and the information filter, which are derived from two different ways of representing Gaussians. Both of these filters can be extended to non-linear systems by using a Taylor expansion to produce linear approximations of non-linear functions. Mixtures of Gaussians can also be used to extend the Kalman filter to encompass situations in which

multiple hypotheses are required, but each extension increases the complexity of the algorithm. This extensibility is one of the reasons for the popularity of the Kalman filter in state estimation problems. With some extension, the information filter is particularly suited to multi-robot systems where information from multiple sources must be integrated, but issues with complexity have resulted in the Kalman filter becoming the more popular of the two for the majority of problems [4].

Algorithm 2 Basic Monte Carlo Localisation [4]

1: **Algorithm MCL**$(\mathcal{X}_{t-1}, u_t, z_t, map)$
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: **for** $m = 1$ to M **do**
4:     $x_t^{[m]} = \textbf{sampleMotionModel}(u_t, x_{t-1}^{[m]})$
5:     $w_t^{[m]} = \textbf{sensorModel}(z_t, x_t^{[m]}, map)$
6:     $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: **end for**
8: **for** $m = 1$ to M **do**
9:     draw i with probability $\propto w_t^{[m]}$
10:    add $x_t^{[i]}$ to $\mathcal{X}_t$
11: **end for**
12: **return** $\mathcal{X}_t$

---

3) Nonparametric Filters: In contrast to Gaussian filters, nonparametric filters discretise the probability distribution and do not require assumptions of linearity and Gaussian belief distributions. Instead, the state is approximated by a finite number of values which are taken from the belief state at any given time. The number of these values can be varied, and nonparametric filters converge to the correct state as the number tends towards infinity. Because this family of filters does not impose restrictions on the posterior density, they are useful for problems such as global localisation, which require the state to be represented in a complex form. Global localisation is the problem of determining the position of the robot without knowing its initial position, resulting in global uncertainty and the need for a multimodal belief distribution. Although as a result nonparametric filters are more computationally expensive than Gaussian filters, it is possible to vary the number of values used to represent the belief to suit the problem using *adaptive* techniques. Examples of these types of filters are the histogram and particle filters. The histogram filter decomposes a continuous space into some finite number of regions, each of which is assigned a probability based on the belief. Extensions include using dynamic decomposition techniques to use more coarse representations in regions with lower probability, and the use of *selective updating*, which updates only the parts of the space which are deemed important. Particle filters approximate the belief by drawing a number of hypotheses called *particles* randomly from the belief distribution. The most important part of the particle filter is the resampling step, which selects particles from the initial set proportional to an importance factor. This has the effect of concentrating particles in areas of high likelihood, reducing computational power spent in areas which are not relevant. Improving the sampling method and adapting the number of particles based on time and uncertainty lead to more effective and error-resistant particle filters. A property which is particularly useful to us is that particle filters are very easy to implement [4].

4) The Markov Assumption: One of the reasons that these filters are so efficient stems from the assumption that the Markov property holds. While this may not actually be the case, the Bayes filter that forms the base of these filters is robust to the violation of some of the requirements of the property. The Markov property states that the future states of the system do not depend on past states. In other words, considering all previous states of the system provides no additional information for predicting future states; all prediction can be done using only the current state. Previous states do not have to be included in calculations, and as a result do not need to be stored, which leads to faster computation and lower memory usage.

5) Sensor & Motion Models: To use any of these filters to solve a localisation problem, it is necessary to model the sensors and motion of the robot. These models include some parameters representing the uncertainty attached to performing a specific action or receiving a certain measurement from a sensor. The motion model is used in the prediction step to determine the state transition probability $P(x_t \mid x_{t-1}, u_t)$; the probability of going from state x_{t-1} to x_t given that the action u_t was performed. The sensor model is used to calculate $P(z_t \mid x_t, map)$, which represents the likelihood of the measurement z_t being received in the state x_t on a given map. The model incorporates knowledge about the noise parameters of the operating environment and the sensor being used in the form of probability densities. For example, a range sensor could include densities for correct measurements, measurements of unexpected obstacles, sensor failures and random measurements [4].

Integrating the sensor and motion models into the particle filter results in an algorithm known as Monte Carlo localisation (MCL). The most basic version of MCL is shown in Algorithm 2, in which $\mathcal{X}_{t-1}$ represents the particles from the previous time step, M the total number of particles, $x_t^{[m]}$ the state of the mth particle with the action u_t applied to it via the motion model, $w_t^{[m]}$ the importance weight of the mth particle, and $\bar{\mathcal{X}}_t$ the resampled set of particles. We use an adaptive implementation of MCL provided by the ROS system.

B. Route Planning

1) Probabilistic Roadmaps: A probabilistic roadmap (PRM) is a multi-query motion planning algorithm for robots in static workspaces [5] which solves the problem of determining a path between start and goal configurations while avoiding collisions with obstacles in the workspace. The roadmap algorithm consists of two phases; the learning phase and the query phase.

In the first step of the learning phase, the construction step, a graph G with vertices V and edges E is constructed. Free configurations are repeatedly sampled from the workspace and added to V. A free configuration is one where the robot does not intersect with walls or other obstacles in the workspace. Once a specified number of configurations have been sampled, the vertices in V are connected. Attempts to connect two vertices v and n are only made if they are within a neighbourhood specified by a maximum distance D, and a connection is only made if the path between the two vertices is also valid, i.e. there is no configuration on the path where

Fig. 1: The Pioneer 3-DX fitted with tripod mounted Kinect and Hokuyo laser rangefinder.

the robot intersects with walls or obstacles. If a connection is made, the edge from v to n is added to E. The second step of the learning phase is the expansion step, which is intended to improve the connectivity of the graph. This step can be skipped if a sampling algorithm, which takes into account difficult (narrow) areas of the workspace, is used in the construction step.

Once the graph has been constructed, the query phase is entered. In this phase, the roadmap is used to find paths between the start and goal locations s and g. In order to do this, s and g must first be connected to the graph using the same connection strategy used in the construction step. A graph search algorithm, such as A*, can then be used to find the shortest path between the two newly added vertices.

There are a number of ways in which the basic PRM algorithm can be improved. Most of these relate to the construction of the graph, as the graph is the part of the algorithm which most influences subsequent performance—if vertex placement is bad, then even with a good graph search algorithm, it is possible that a path will not be found. The sampling method is particularly important, and many different techniques have been proposed to improve the coverage of the PRM. Roughly speaking, the techniques can be separated into uniform and advanced techniques. Uniform techniques include random sampling, sampling points on the intersection points of a grid, and splitting the workspace into cells and sampling some number of points from each of those subspaces. Uniform methods provide a simple but effective way of sampling, but may experience a drop in performance for more difficult workspaces, for example those with narrow corridors. In these cases, using advanced techniques such as medial axis sampling, which generates points at the midpoints between obstacles, can be effective [6]. The strategy used to connect vertices can also affect the structure of the graph, which in turn

affects the query phase. Improvements to the query phase can be made by using different graph search algorithms, but unless the graph is extremely dense, the query time is negligible. As an example, our implementation of Dijkstra's algorithm took 964 milliseconds to find a path between two points on opposite sides of the map with a roadmap populated with 5000 vertices. A path smoothing technique (our implementation can be found in Algorithm 4) can also be used on the path found by the graph search algorithm to improve it further. In some cases, a path may not be found with the graph being used, and although expensive, re-generating the graph can sometimes remedy this problem.

2) Rapidly Exploring Random Trees: A Rapidly Exploring Random Tree (RRT) is a randomised data structure and algorithm that is also used to solve path planning problems. RRTs are designed to handle nonholonomic constraints and high degrees of freedom [7]. Instead of sampling points and then attempting to connect them as in PRMs, RRTs are biased towards moving into unexplored areas of the state space by sampling points and being "pulled" towards them [8]. There are several planners for RRTs, including a bidirectional planner where the tree grows from the start and goal vertices and uses an aggressive heuristic to connect the trees. In contrast to the multi-query PRM, an RRT is a single query algorithm; it is rerun every time a query is made. The properties of RRTs make them particularly suited to problems involving robot arms and robots with nonholonomic kinematics. As our platform can be considered holonomic, we mention RRTs for completeness.

C. Exploration

The problem of exploration involves taking action to maximise information gain, given the current knowledge of the world. What is meant by information gain depends on the problem being solved. In many cases this is discovering the locations of walls and obstacles, but can also extend to more complex objectives such as determining the locations of specific objects, rooms and people. The most basic form of exploration is the random walk, which ensures that any finite sequence of actions will be executed eventually [9]. However, this is a drawback as it is an exhaustive search, and many other improved exploration techniques have been proposed. Greedy techniques move the robot from its current location to the closest location that has not yet been explored [10]. Frontier-based exploration moves to boundaries between open space and uncharted territory in order to maximise its information gain [11]. Distributed approaches using coordinated multi-robot systems have become an active area of research, and novel techniques drawing on concepts from economics have been proposed [12], [13]. As with many solutions using distributed computing, the use of multiple "processors" allows the task to be completed in a shorter time, but requires the implementation of algorithms that can make good use of these increased capabilities.

D. Face Detection

Face detection is concerned with finding whether or not there are any faces in a given image (usually in gray scale) and, if present, return the image location and content of each face [14]. Research in this field can be dated back to the early 1970s, when a number of techniques such as simple heuristics

and anthropometric techniques were proposed [15]. However, these techniques were not very adaptable to changes in the environment, requiring either a lot of tweaking or complete system redesigns. Face detection techniques can be broken down into two separate groups, feature-based and image-based [15]. Feature-based approaches tend to make use of facial features such as facial geometry and skin colour to detect faces. An example of a feature-based approach are Haar-like features, which use contrast variances between adjacent rectangular groups of pixels to classify objects and faces by combining them to make features. Image-based face detection does not have knowledge regarding features presented to it, instead using training and mapping techniques to build detection systems. Neural networks, which use training samples to build up a classification system to identify faces, are an example of an image-based approach. For our system, we considered two particular methods of face detection. One of the methods is known as blob detection which uses a method to extract adjacent regions of the same colour to identify as an object. The other method, which we eventually chose to use, was Haar cascades, face detection using Haar-like features. The way in which this classifier works is by using the change in contrast values between adjacent rectangular groups of pixels, as opposed to using intensity values of a pixel [16]. Contrast variances between pixel groups allow relative light and dark areas to be determined, where two or three adjacent groups with a relative contrast variance form a Haar-like feature. Due to the large number of features which can be derived, a system can be trained to find the best set of features which can then be combined in order to produce a classifier cascade [17]. In the detection phase for a face, a window of a specific size is moved over the input image. A classifier cascade containing all of the Haar-like features is applied to each region of the image which falls within the window, and successful classifications within this window are used to determine whether a face is present in the region.

III. Design

In this section we explain in detail the techniques, algorithms and hardware that we used to solve the task.

A. Hardware & Software

1) Robotics Platform: For the task, we used a Pioneer 3-DX fitted with a Hokuyo URG-04LX laser rangefinder. The Pioneer is 455mm long, 381mm wide, 237mm high, and weighs 9kg, making it a relatively compact platform. It is fitted with six front-mounted and two side-mounted sonar sensors. The robot is powered by three batteries which provide a run time of 8-10 hours, assuming no accessories are attached. The robot is equipped with a differential drive system which has a 0cm turn radius, so it can be considered holonomic. The maximum translation and rotation speeds are 1.2m/s and 300°/s respectively [18].

The Hokuyo laser is mounted on the front of the robot. It has a scan area of $240°$ with a maximum radius of 4m and an angular resolution of $0.351°$. A scan is completed by the laser every 100ms. The laser is accurate to ±20mm at distances between 20 and 1000mm, and $\pm1\%$ of the measurement distance at 1000-4000mm [19].

The Microsoft Kinect is a video device which produces colour images much like a normal camera, but has the additional ability to return a distance (depth) value for each pixel in the image. As such, the device is similar to a 3D camera. The Kinect camera has a $57°$ horizontal and $43°$ vertical field of view with a nominal depth of 3.5m. Images from the camera have a resolution of 640x480 [20]. Interfacing the Kinect with the ROS framework is achieved by using the OpenNI SDK, an open source framework for 3D sensing development.

2) Robot Operating System: The Robot Operating System (ROS) is an open source software framework developed by Willow Garage and the robotics community that provides libraries and tools for use in robot applications. The system comes with a series of *stacks*; user-contributed packages which can be used to perform SLAM, localisation and various other common tasks, which means that development can be focused on solving the problem at hand. ROS development can be done in a variety of programming languages, including C++, Python and Java, and can be run on a large number of platforms, with support for humanoid robots, arms, quadrotors and many mobile robots.

The system has a graph-like structure and makes use of data streams for communication between nodes, and as a result is robust to node failures. The use of data streams for communication allows for a high degree of modularity—a node written for one task can be used in the system for another task with little or no modification. Each node can have an arbitrary number of *publishers* for sending messages to *topics* and *subscribers* for receiving them. One of the major strengths of the system is that when a message is sent, it does not have to be sent to a specific node; once it is published, the message is available to any node which is subscribed to the topic on which it is published. A master node provides information about the state of the whole system, including node status and the currently active topics.

B. System Structure

Our implementation attempts to make the most of the ROS system, with separate nodes for each subsystem. The actions of the system are controlled by the control node which is based on a finite state machine. This node then hands off to other nodes when tasks need to be executed. The navigation node uses data from the localisation and PRM nodes to move between points in the map. Vision processing is done in a node written in C++ for speed, and the resulting data is used by the control node to make decisions about which actions to take.

1) Localisation Node: We independently developed two localisation nodes which implemented Monte Carlo localisation (MCL), but which were not sufficiently error free that we were confident in using them for the combined task. Instead, we made the decision to use the ROS implementation of AMCL which comes with the software installation. By using a tried and tested implementation, we reduce the likelihood of bugs causing problems with the localisation. This is very important as, to move safely through the space, it is necessary to have an accurate position estimate. We had to tweak some parameters in the algorithm that we deemed necessary in the testing of our own algorithms while running in tandem with some of the other nodes that were to make up the final system. The

AMCL defaults for filter updates are 0.2 metres for translation and 0.52 radians for rotation. We decreased both of these to 0.1 metres and 0.1 radians respectively in order to receive filter updates more often. A comparison of the tracking error of the three localisation algorithms can be found in Figure 6.

Algorithm 3 Probabilistic Roadmap Generation

1: **Algorithm generatePRM**(map)
2:   $V \leftarrow$ **sampleVertices**(map)
3:   **for** $v \in V$ **do**
4:     $a \leftarrow 0$
5:     **while** $a < A \wedge$ **nConns**(v) $< C$ **do**
6:       $n \leftarrow$ **getClosest**(V, v)
7:       **if** dist(v, n) $< R_n$ **then**
8:         **if** ¬**neighbour**(v, n) $\wedge$ **vPath**(map, v, n) **then**
9:           $E \leftarrow$ **connect**(v, n)
10:         **end if**
11:       **else if vPath**(map, v, n) **then**
12:         $E \leftarrow$ **connect**(v, n)
13:       **end if**
14:       $a \leftarrow a + 1$
15:     **end while**
16: **end for**

---

2) PRM Node: The PRM node is used to find paths to navigate from one point on the map to another. Whenever a goal point is received by the node, it finds the shortest path from the robot's current position to the goal position by using Dijkstra's algorithm to search the road map generated when the node is initialised. We will now walk through Algorithm 3, used for generating the roadmap. In line 2, points are sampled from the map using cell sampling. Cell sampling is a uniform sampling technique in which the map is split into a number of equally sized square *cells*, and a certain number of points are randomly sampled from within each cell. This sampling technique, although simple, provides very good coverage of the space, and is particularly good at sampling points in narrow corridors, as each cell is required to contain a specified number of points. We now iterate over the sampled points, hereafter called vertices, attempting to make connections between them. There are three connection strategies available. The *nearestN* strategy connects only to a certain number of the nearest nodes by Euclidean distance. The *threshold* strategy connects nodes to a certain number of randomly selected nodes which are within a threshold distance of the node. The *neighbourhood* connection strategy, which we have chosen to use after experimentation, is explained here. We connect the target vertex v to candidate vertices n in the graph in order of proximity to v. As some vertices may be unconnectable, we specify a maximum number of attempts A that should be made to connect a vertex to the graph. We also specify C, the maximum number of connections that a vertex can have. If the number of connection attempts a exceeds A, or the number of connections the target vertex has exceeds C, the algorithm will move on to the next vertex. When connecting vertices, the algorithm starts by selecting the vertex in the graph which is the shortest Euclidean distance from v. In line 7, a check is made to see if the candidate vertex is within the neighbourhood of the target vertex. Vertices within a Euclidean distance R_n of v are considered to be within its neighbourhood. If the candidate vertex is within the

neighbourhood, v and n are connected only if they are not already connected by a path that is contained entirely inside the neighbourhood, and there is a straight line between the two vertices which does not pass through any obstacles. If the candidate vertex is not in the neighbourhood of the target, then a connection is made so long as there is a free path between the the two. These checks are performed in lines 8 and 11. The benefit of this is that vertices are not connected multiple times to other vertices in close proximity, but connect to vertices further away which means that shortcuts can be made between neighbourhoods. A vertex is guaranteed to be connected to the closest accessible vertex in its neighbourhood.

When checking the validity of points and lines on the map, it is necessary to take into account the size of the robot. This can be done by checking a radius around each point for obstacles, but for the purposes of efficiency, we have taken a slightly different approach. Instead of checking the area around points, we increase the size of obstacles by the required safe distance from obstacles. In our case, this is slightly above half of the longest dimension of the robot. After applying this size increase, we receive an "inflated" map, an example of which can be seen in Figure 2, which we use when generating the graph and for path planning. When querying the PRM, a the
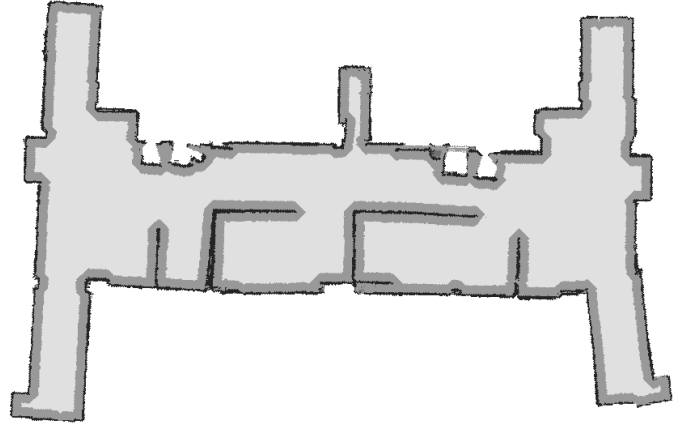

Fig. 2: Example of an inflated map used for PRM. Black pixels are obstacles. Grey pixels inside the map show the inflation applied to the obstacles.

robot's current position and the goal location are added to the graph and connected in the same way as other vertices. Dijkstra's algorithm is then run with the newly added vertices as parameters, and the shortest path between the vertices is returned. While using the raw path is viable, it can be further improved by smoothing. As seen in Algorithm 4, redundant intermediate vertices are removed from the path in an iterative fashion. I is the number of smoothing passes executed, and P is a vector of waypoints along the path. P_n represents the nth waypoint in the path. The intermediate waypoint b is removed only if there is a valid path between waypoints a and c. The algorithm terminates either when the requested number of iterations has completed or if the path cannot be smoothed further. The benefit of smoothing is twofold; the path length becomes closer to the optimum, and the number of vertices in the path is reduced. The fewer vertices there are in the path, the less the robot must rotate during travel, and this results in a faster travel time from start to goal.

When a path has been found, a message is sent to a topic

announcing that a path to the goal was found. In cases where a path cannot be found with the current graph that the PRM holds, the graph will be regenerated and searched again a number of times. If a path still cannot be found after the specified number of regenerations has occurred, the PRM will publish a message informing listening nodes of this fact so that they can handle the consequences.

Algorithm 4 Path Smoothing

1: **Algorithm smoothPath**(P, I, map)
2: **for** $i = 0$ **to** I **do**
3:     **for** $j = 0$ **to** $|P| - 2$ **do**
4:         $a \leftarrow P_i$
5:         $b \leftarrow P_{i+1}$
6:         $c \leftarrow P_{i+2}$
7:         **if vPath**(map, a, c) **then**
8:             **remove**(P, b)
9:         **end if**
10:     **end for**
11:     **if** path not modified **then**
12:         break
13:     **end if**
14: **end for**
15: **return** P

---

3) Navigation Node: The navigation node moves the robot along paths provided by the PRM, and uses the estimated pose provided by the localisation node as the robot's current position. When a path is received, the first point is removed and set as the current waypoint to navigate to. The first action taken by the navigation node is to rotate towards the point until the robot heading and the bearing of the point from the robot's current location are within 0.15 radians of each other. With this constraint, we can be sure that the robot's heading is very close to the bearing of its destination before any movement is made towards it. If the waypoint is a particularly long distance away, the error between the heading and bearing at the starting location gradually increases as the robot nears the waypoint. If the error is below the 0.15 radian threshold, then only translational movement is applied, but if the error exceeds the threshold, angular movement is also applied, and the robot moves in a smooth curved path towards the waypoint. Adjusting for the error while moving means that the robot does not have to stop to adjust, which decreases the time spent travelling to the goal. Each waypoint has a radius of 0.2 metres, within which the robot is considered to have arrived at the point. When an intermediate waypoint is reached, the next point is removed from the path and set as the destination. The path is gradually consumed in this way until the goal is reached, at which point message is published informing any listening nodes of the fact.

The navigation also performs obstacle detection, by using laser data. An area of 90° directly in front of the robot is split into 10 sectors, from each of which a median distance reading is drawn. The median is used to reduce the effect of noise and erroneous readings. If the median of any sector is within a safe distance of 0.5 metres, movement commands will no longer be sent to the motors. The position of the detected obstacle is calculated using trigonometry and placed onto a temporary map, which is passed into the PRM. The obstacle is detected as a point on the map, which is then inflated in the same way

as the map used in the PRM. The PRM then prunes vertices which fall within the obstacle, as well as connections between vertices that pass through the obstacle. When this process is complete, the navigator re-sends the goal to the PRM which generates a new route, avoiding the obstacle. Obstacles are removed from the map when the robot is more than 4 metres from their location, or when it arrives at its goal. When an obstacle is removed, the graph is reconnected to ensure that the connectivity of the graph is not permanently affected.

4) Vision Node: For face detection, we decided to make use of camera data from the Kinect. Face detection in ROS necessitates the use of a vision library. We decided to use OpenCV as it is a highly popular tool in the research field, and is written in part by the ROS developers, suggesting easier integration. Communication between ROS and OpenCV requires the cv_bridge stack, which is only compatible with C++ and Python. We decided that C++ was a better candidate than Python as face detection is very computationally expensive, and C++ is faster in these types of applications. The OpenNI SDK publishes many ROS topics containing a
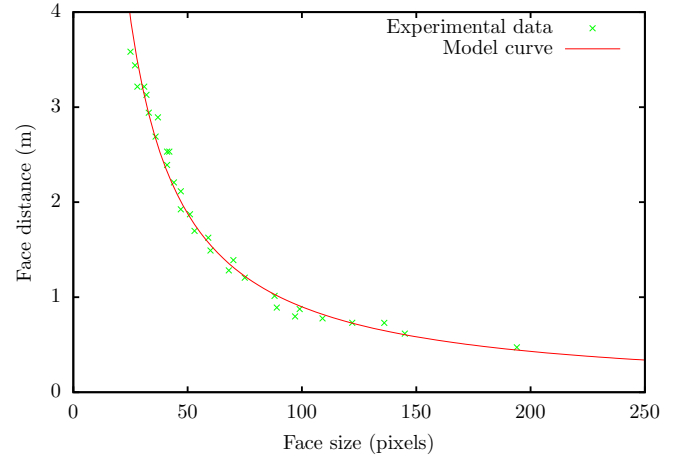

Fig. 3: Depth model showing correlation between face size and distance to face.

variety of data from the Kinect. We made use of two of these topics, one providing depth and one RGB image data from the Kinect camera. These two topics are subscribed to by the C++ node that we created to allow the use of OpenCV with the Kinect image data. The output from OpenCV is then used in conjunction with the Kinect depth data to gain further confidence in whether a face has been detected. The C++ node publishes the location of the detected face along with the depth (i.e. distance to the face). This data is subscribed to by the control node which then uses the data accordingly.

The face detection system using OpenCV worked as expected, but gave many false positives, detecting cabinets and clothing as faces, even after we tuned the parameters that were available to us. We decided to use the Kinect's depth information to decide which detections were actually faces. To do this, we considered the fact that many of the false positives were much larger or much smaller than actual human faces, and that human faces do not tend to vary much in size. Using the size of the rectangle representing the potential face in the image along with the depth of the potential face, we can determine whether the rectangle is a real face by

comparing it against the expected depth and rectangle size. To obtain these values, we created a face size model by recording measurements of real faces. We had two human participants, one tall and one short, stand 0.3m from the camera and take small steps backwards until they were 3.5m away, as we recorded the measurements of depth and rectangle size. Once we had these values, we plotted them on a graph and programmatically found a quadratic line of best fit, shown in Figure 3. The line of best fit is our model, which we can use to compute an expected value for either depth or face size, as long as we have the other. Once this model was included in the node, rectangle sizes from detections were used to determine expected depth and compared with the depth data from the Kinect. When they were not within a set tolerance, the detections were discarded as false positives.

Algorithm 5 Face Detection Node

1: **Algorithm detectFace**(image)
2: face_rects ← **haarDetect**(image, haar_cascade)
3: **for** face in face_rects **do**
4:     **if** ¬**validFace**(face, depth) **then**
5:         **remove**(face_rects, face)
6:     **end if**
7: **end for**
8: **publish** face_rects

Algorithm 5 shows the algorithm in the face detection node. In line 1, a Haar feature detector is run on the image received from the Kinect video feed, from which a series of rectangles representing possible faces are received. Non-faces are removed from the array in line 4. If valid faces remain after the array has been checked, they are published to the network. Face validity is determined by Algorithm 6. In line 2, depth values from a smaller rectangle of 20% width and height of the original rectangle, in the centre of the original rectangle are retrieved. We use the median depth of this smaller area as the actual depth of the potential face. If the median of the area exceeds a threshold of 3.5m as per the effective range of the face detection algorithm, the face is rejected. Otherwise, in lines 7–10, the expected depth and actual depth are compared. The tolerance value α, set to 15%, is applied to the value expected by the model. If the actual depth is within this tolerance range, the face is considered valid.

Algorithm 6 Face Validity Check

1: **Algorithm validFace**(face_rect, depth)
2: depth_pixels ← **getCentreRegion**(face_rect)
3: median ← **getMedian**(depth_pixels)
4: **if** median > threshold **then**
5:     **return** False
6: **else**
7:     expected ← **expectedDepth**(face_rect)
8:     tolerance ← α·expected
9:     diff ← |median − expected|
10:     **if** diff > tolerance **then**
11:         **return** False
12:     **end if**
13: **end if**
14: **return** True

With our first implementation of this model, referred to as depth model 1, we noticed that the expected depth values were causing the model to discard valid faces in the far left and far right of the field of view of the camera. Although the Kinect sensor manufacturer claims to adjust for angle distortion, it is clear that there is some discrepancy at larger angles. We found that the model worked well for the central 40% of the image (roughly 95% true positive rate), while for the left and right 30% of the image there was a smaller true positive rate (roughly 60%). This led us to create a pair of models, one for the centre of the image, one for the left and right hand sides, to replace the previous model. We created these in the same way as the previous model. For each model (centre and angle), we collected data for two participants of varying height and combined them. The models are used conditionally—if the centre of the face rectangle is within the central 40% of the image, the centre model is used, otherwise the angle model is used.
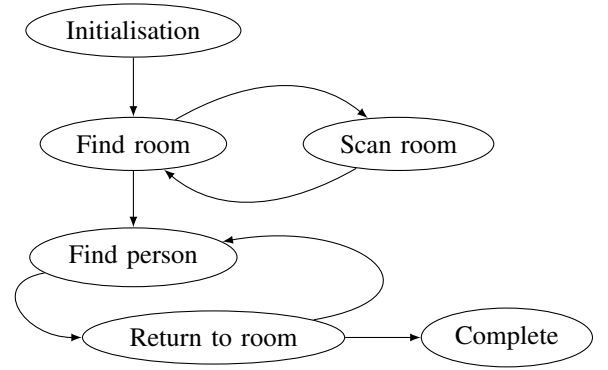

Fig. 4: Simplified version of the finite state machine used in the control node.

5) Control Node: The control node can be considered as a finite state machine, where there are several states in which the machine can be, and it transitions between these states until it reaches the completed state. A simplified diagram can be seen in Figure 4. The robot starts in the *initialisation* state, and waits to receive a map of the space that it is to work in. Once a map has been received and pose estimates are being received from the localisation node, the robot moves into the *finding room* state, in which it attempts to locate an empty room that it can take people to. Due to time constraints, the central points and bounding boxes of the two meeting rooms available on our map were hard coded into the system. The robot uses the PRM to navigate to the meeting room closest to its current location and, once inside, performs a 360° scan of the room to check whether it can identify any faces. When a message is received from the vision node containing rectangles representing faces in the field of view, the robot transitions into the *face check* state. In this state, the robot attempts to confirm whether what is being seen is really a face. A face is confirmed when the vision node sends three successive messages which all contain a face in the same area. When the first message is received, the rectangle with the lowest depth value is selected. When subsequent messages are received, the rectangle with the centre closest to the previous rectangle is selected, and a check is made to see if this rectangle's centre is within the bounds of the previous rectangle. If one of the subsequent messages contains no rectangles, or contains rectangles which are too far from the previous rectangle received, the robot returns to
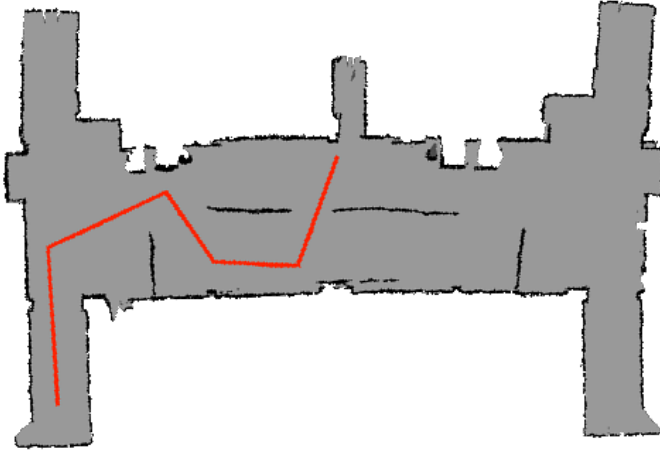
Fig. 5: Path used for tracking error experiment.

scanning the room. If a face is confirmed, the room is occupied and the robot moves to the next room on the list and performs the same scan. If this room is also occupied, the robot will continue to check rooms in the list until it has exhausted the list. If none of these is found to be free, the robot will return to the first room and start the list again. This will be repeated until an empty room is found.

Once an empty room is found, the *exploring* state is entered. To save time, we decided to adapt the sampling methods used for PRM to generate exploration paths. Using cell sampling, one point is sampled in each cell with a width of four metres. Points are not sampled within meeting rooms, as we are only concerned with people outside the meeting rooms. The robot's current location is connected to the point closest to the robot, and that point is connected to the next closest point to itself. Subsequent points are connected in the same way until a single path is constructed. The first point in this path is set as the goal location, and the robot moves to it using the PRM. Once the point has been reached, the next point is consumed, and so on until the end of the exploration path is reached. When this happens, points are once again sampled, but this time with a cell width of two metres. When the path generated from this is consumed, the cell size is set to one metre. After that, the cell size is once again set to four metres and the cycle continues. Decreasing the cell size means that more points are sampled into the map, and so places which may not have been explored with a larger cell size have a chance of being visited. If a face is seen while the robot is exploring, the robot will attempt to centre the face in its field of view and then confirm that it is a face in the same way as described above. If a face is confirmed, then the robot will set a goal to just in front of the expected location of the person calculated from the depth of the rectangle. When the goal is reached, a sound is played to alert the person, who is asked whether they would like to attend the meeting via a dialogue box displayed on the screen. If the person accepts the invitation, the robot guides the person back to the meeting room that it found was free in the finding room state. We assume that the person follows the robot once the invitation is accepted. If the invitation is declined or a 15 second timer expires, the robot continues exploring. The robot is provided with a number of persons that it must find and bring to the meeting room, and once that number of persons have been found and brought to the room, the task is complete

and the system is shut down.

We make several assumptions about how the world works. Due to limitations with our face detection, persons who we are trying to find are assumed to be facing the robot at all times and do not move once their face has been confirmed by the robot. We also assume that the person follows the robot to the meeting room, and that people who have been taken to the meeting room do not leave.

IV. EXPERIMENTATION

In this section we display the results of experiments carried out to determine the best algorithms, parameters or strategies to use in various parts of our system.

A. MCL Tracking Error

1) Aim: The aim of this experiment is to compare the tracking error of three different MCL implementations; the built in ROS AMCL and two of our own implementations.

2) Method: As each MCL implementation has many different variables, we will use the set of variables that resulted in the best performance in past experiments. To provide accurate results, we require a ground truth, which will be obtained from the Stage simulator. We will run the experiment three times for each implementation and record the tracking error every second while the robot moves, until it reaches the end of its path. The tracking error is defined as the Euclidean distance between the ground truth and the estimated pose provided by the localisation at each point at which a measurement is taken. The robot will start from the same initial position and follow the same path, seen in Figure 5, in each experiment.
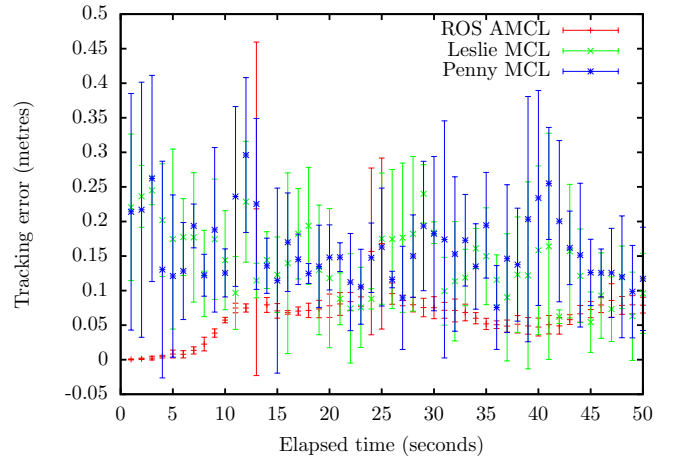

Fig. 6: Comparison of Tracking error of ROS AMCL and two independently developed MCL algorithms

3) Results: Figure 6 shows the tracking errors for each implementation, with error bars representing the standard deviation over three runs. Apart from two anomalous points at 13 and 25 seconds at which all implementations experience an increase in tracking error, the deviation of AMCL is significantly lower than the other two implementations. The average deviation over the three runs for AMCL is 0.019m, compared to averages of 0.071m and 0.085m for Penny and Leslie respectively. It is clear that to obtain accurate position estimates we should make use of ROS AMCL.
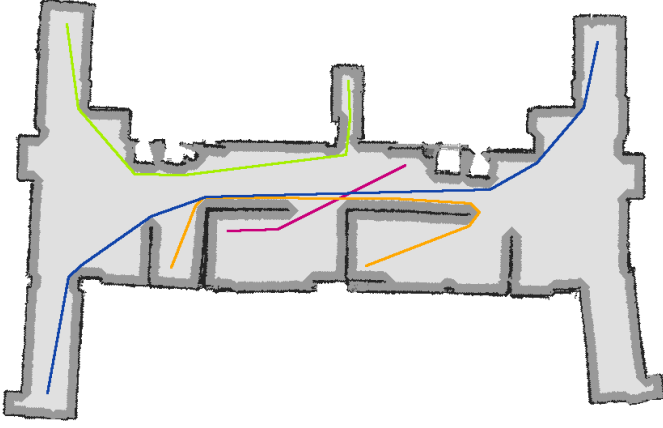
Fig. 7: Optimum paths used for comparison of connection strategies and sampling methods.

B. PRM sampling methods and connection strategies

1) Aim: The aim of this experiment is to find the sampling method and connection strategy which, as a pair, produce the shortest path lengths on average.

2) Method: To find the best sampling method and connection strategy pair, we created four qualitatively different pairs of start and end points. We passed each point pair to the PRM, applied the path smoothing algorithm to the generated path and measured its length. We varied the maximum number of connections per node—using values of 5, 10, 20 and 30—and only chose the value which produced the best path length. In addition, we experimented with different values for variables specific to each sampling method. For grid sampling we varied the size of the x and y step, using values of 0.5m, 1m, 2m and 4m. For cell sampling we varied the number of points sampled in each cell, either 2 or 5, and used 1m, 2m, 4m and 6m cell sizes. For random sampling, we varied the total number of points sampled, using values of 25, 50, 100, 200, 400 and 800. We ran three trials for every combination of sampling method, connection strategy, maximum connections per node, and pair of start and end points as well as every combination of variables specific to each sampling method.

We chose four start and end point pairs which would provide qualitatively different paths, allowing us to compare the various techniques in different environments, ensuring we choose a combination which is able to generalise well. Point pairs were chosen to give examples of long, medium and short paths through the map. Finally, we added a pair in which the goal point is within a narrow corridor. This final pair should give us data about how well the sampling method deals with difficult areas of the map.

3) Results: The experiment produced four path length values for every combination mentioned above, one for each pair of start and end points. Sets of four values cannot be compared at once, so in order to compare combinations, these four path lengths must be combined into a score which indicates how good the paths are. To create such a score, we determined the theoretical optimum path length by manually tracing lines on the map and measuring their length. These are the lines shown in Figure 7. The score for a combination of variables and methods is calculated as the ratio between the computed path length and the theoretical optimum. For example, if the

optimum length were 10m and the computed length were 12m, the ratio as a decimal number is 1.2. To achieve a combined score for all four paths, we average the four scores. Figure 8 shows the best scores achieved by sampling method and connection strategy combinations.

The best score (lower values are better) is achieved by random sampling with the neighbourhood connection strategy. However, during the experiment the trials for this combination did not always compute a path. In particular, a path could not be computed where the end point was within the narrow corridor. This is due to the fact that with random sampling nodes are randomly distributed across the map with no regard for the fact that narrow areas have a smaller probability of having nodes generated within them. Since this combination could not compute a path in one of our test cases, we cannot use it.

The next best score was achieved by grid sampling and the neighbourhood connection strategy. This combination, similarly, consistently could not compute a path to the end point within the narrow corridor. This is easily explained by the nature of grid sampling; the nodes are placed across the map as evenly spaced points, meaning that narrow areas have a smaller probability of having nodes placed within them. Again, we cannot use this combination because it does not compute a path in our test scenario.

Sampling Method	Connection Strategy	Best Score
Cell	Neighbourhood	**1.03909**
	NearestN	1.09907
	Threshold	1.12969
Grid	Neighbourhood	**1.03693**
	NearestN	1.14463
	Threshold	1.05153
Random	Neighbourhood	**1.02578**
	NearestN	1.05171
	Threshold	1.06639

Fig. 8: Best scores for each combination of sampling method and connection strategy

The third best score was achieved by cell sampling and the neighbourhood connection strategy. This combination was consistently able to compute paths in all cases in the experiment. The reason is fairly clear; cell sampling is designed to ensure complete coverage of the free space in the map and provides a higher probability of node coverage in narrow areas containing walls or non-free space. We could not see any qualitative objections to using this combination, so this is chosen as the best combination. Specifically, this combination used 30 maximum connections per cell, a cell width of 1m and a target number of nodes per cell of 5.

C. Haar cascade comparison

1) Aim: The aim of this experiment is to find the Haar cascade most suited to perform face detection with the Kinect. We wish to find the cascade which has the highest specificity and sensitivity; all actual faces should be classified as faces, and anything which is not a face should be ignored.

2) Method: To test the different Haar cascades we captured a set of images using the Kinect camera, all of which included
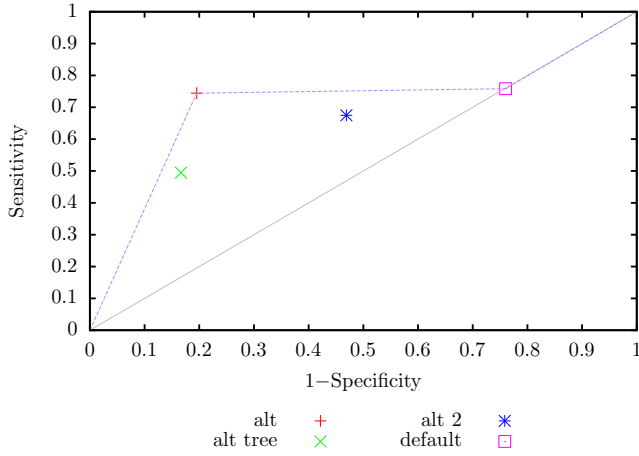
Fig. 9: ROC graph of frontalface Haar cascade results.

faces in different surroundings and at different angles. We then processed each image using our face detection node which works on still images, using the frontalface_default, frontalface_alt, frontalface_alt2 and frontalface_alt_tree cascades (obtained from the OpenCV package). We manually analysed the processed images, which contain rectangles indicating the areas detected as faces. In order to generate the data, we looked at the image output from the program for each cascade and analysed the number of faces that were correctly detected (true positives), the number of non-faces not detected (true negatives), the number of non-faces detected (false positives) and finally, the number of faces which were completely missed (false negatives). Once we had all these values for each image, we averaged the sensitivity for each Haar cascade to find their final value.

3) Results: From our results of the above experimentation it was obvious that none of the cascades performed well on profile faces. This was expected as the cascades we used were specifically trained on frontal face images. The default and alt cascades performed better on faces at an angle than the alt2 and alt tree. The alt and default cascades also performed well on frontal face images with various backgrounds. As is obvious from the ROC graph in Figure 9, the best performer was the alt cascade. Although its sensitivity value of 0.744 was slightly lower than the 0.758 of default, the alt cascade was more effective with regard to true negatives (i.e. not indicating that faces exist in places where other cascades picked up false positives). This result pushed the 1−specificity reading for the alt cascade towards the left, highlighting it as the best candidate. Although the default cascade performed better in the sensitivity reading, it also performed worst in the 1−specificity reading, frequently classifying non-faces as faces. It is clear that the cascade which performed best was frontalface_alt, classifying actual faces as faces with a good degree of consistency. As the cascade still detected false positives, we decided to attempt to reduce the frequency of these by combining depth information from the Kinect with the usage of Haar cascades on the images.

D. Adding the Kinect depth model

1) Aim: The previous experiment established the best Haar cascade to use in our vision node. However, there are assump-

tions we can make which were not made in the last experiment. The aim of this experiment is to determine the performance of face detection when making some assumptions about the world, as well as using a face size model.

2) Method: Since we are trying to improve detections for this cascade with our face size model, we performed this experiment in the areas where all the cascades generally performed badly (had many false positives). In this area, we used all of the cascades to find features of the scene which were falsely detected as faces. We noted these for reference as true negatives, which we can eliminate with our new depth model. To determine the sensitivity and specificity of the vision node with the new assumptions, both before and after the inclusion of both Kinect depth models (depth model 1 and depth model 2 as described in the Design section), we designed a single experiment which would produce results for all three at once.

We modified our vision node to show rectangles of different colours based on whether they would be accepted by model 1 or model 2. We had a participant move within the depth of field of the camera to 100 different positions and we took readings for true and false positives and negatives. Alongside the features of the scene we noted earlier which we hoped to eliminate as true negatives, this allows us to gain three points on the ROC graph to compare.
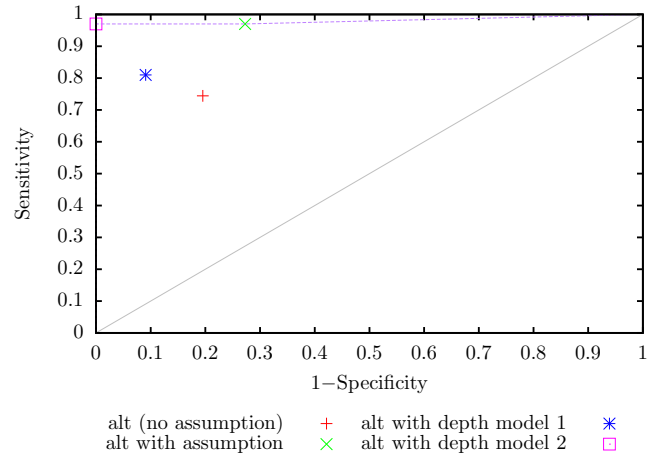

Fig. 10: ROC graph of frontal_face_alt cascade performance with assumption that subjects face the camera and with depth models applied.

3) Results: The resulting ROC graph is shown in Figure 10. The point labelled 'alt (no assumption)' is the point taken from the last experiment. We collected 11 scene features which were falsely identified by the cascades as faces. This means that the 1−specificity axis of the ROC graph is of a fairly low resolution. However, with 100 true faces, the sensitivity axis is of a high resolution.

We can see in the graph that sensitivity and the 1−specificity value have both increased. It appears that adding the assumption of faces being directed towards the camera increases sensitivity, while performing the experiment within an area where cascades tend to perform badly causes the 1−specificity value to increase.

Using depth model 1 we see that we have decreased both the sensitivity and the 1−specificity value. The decrease in sensitivity is clearly due to the wrongful discarding of real faces done by model 1 on the left and right hand sides of the image. However, the correct discarding of false positives has also decreased the 1−specificity value. Overall, this point is better in some respects and worse in others when compared with using no model at all.

When using depth model 2, we see that the performance improves greatly. This depth model discards none of the true positives given by the cascade, keeping its sensitivity identical to using no model at all. However, false positives are down to 0, making the 1−specificity value also 0. It is worth noting that, since there were only 11 'true negatives' being sought, this value being 0 may be slightly misleading and may well be higher with a larger number of true negatives. Using depth model 2 cannot improve on the sensitivity of using no model at all, but by eliminating all true negatives it is clearly the best point on the ROC graph and is certainly worth using.
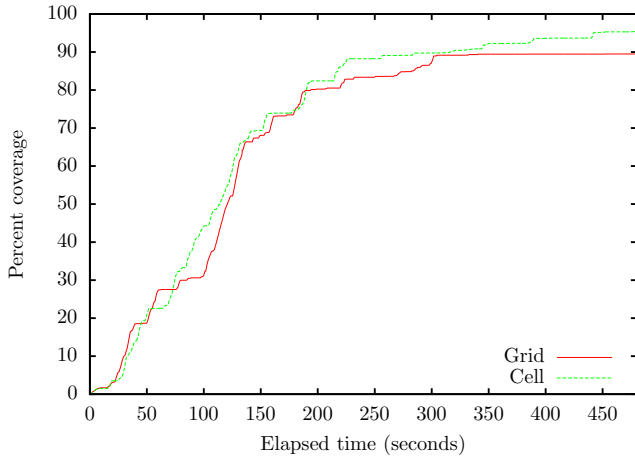

Fig. 11: Comparison of the best grid and cell based exploration strategies. The heatmap corresponding to the cell strategy run is found in Figure 13.

E. Exploration coverage

1) Aim: In this experiment, we aim to find the best method of sampling points on the map so that the robot can explore the most space in the shortest amount of time.

2) Method: We let the robot explore the map in simulation, as described in Section III-B5, for 8 minutes, using each of the sampling methods with four different strategies (8m-6m-4m, 6m-4m-2m, 4m-2m-1m and 2m-1m). We used ray tracing to project the field of view of the robot onto an occupancy grid. Each grid cell the ray passed through was counted as explored. Rays originated 0.3m in front of the robot and travelled outwards until the ray was obstructed by a wall or reached the maximum effective distance of the face detection node of 3.5m. The number of free cells remaining in the map was calculated each second. As we do not explore the meeting rooms, cells inside these areas were not counted in the total. Each experiment will be run for 8 minutes. The movement speed of the robot affects the speed of exploration. The maximum movement speed and rotation speed were set to 0.7m/s and 0.4rad/sec respectively in each experiment.

Strategy	Sampling method	
	Cell	Grid
2m→1m	91.3	**89.4**
4m→2m→1m	**94.0**	86.4
6m→4m→2m	93.5	83.5
8m→6m→4m	93.7	82.8

Fig. 12: Percentage coverage after eight minutes of exploration with different strategies and sampling methods.

3) Results: As can be seen in Figure 12, the cell based exploration consistently covered more of the map than the grid strategy. Figure 11 shows the best run of the best strategy for each method. After 300 seconds, the grid method stopped covering new areas of the map, while the cell based method continued to increase its coverage. The effectiveness of the exploration can also be shown as a heat map. Figure 13 shows the heat map at the end of the best cell method run, which shows that almost the whole map was explored, with relatively even coverage.

F. Locating meeting rooms

1) Aim: The aim of this experiment is to test how well our robot performs in finding an empty room. We wish to test the robot's ability to handle different events during the meeting room availability confirmation stage, as well as confirm that it correctly identifies and enters the closest room first.

2) Method: We tested three states of the world which could affect the robot's performance in the task. In the first state, neither of the rooms was occupied. The second state had the first room the robot entered occupied, and the second unoccupied. In the third state, both rooms were occupied. The robot was started from two positions in opposite corners of the map to ensure that the closest room to the robot was checked first. We recorded the time elapsed from the start until checkpoints of when the robot entered the first room, starting and finishing scanning of the room, leaving the first room, and the same conditions for the second room. In the first world state, we stopped timing when the robot finished scanning the first room, in the second state when it finished scanning the second room, and in the third state when the person in the
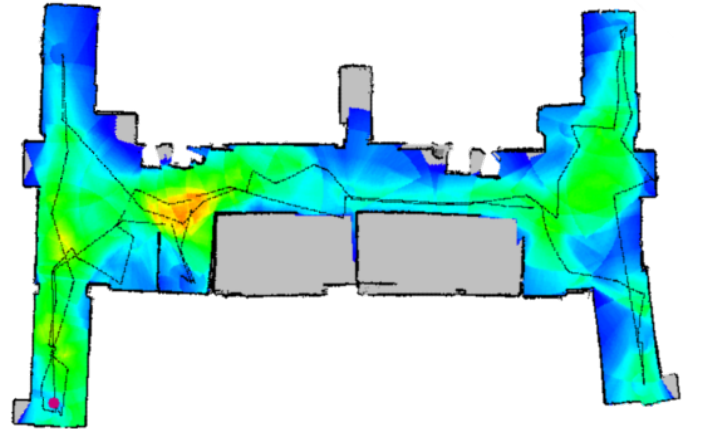

Fig. 13: Heat map showing coverage over an 8-minute run using the best cell-based strategy. The colour gradient goes from red (high) to blue (low). The pink dot shows the starting location. The black line is the path taken by the robot.
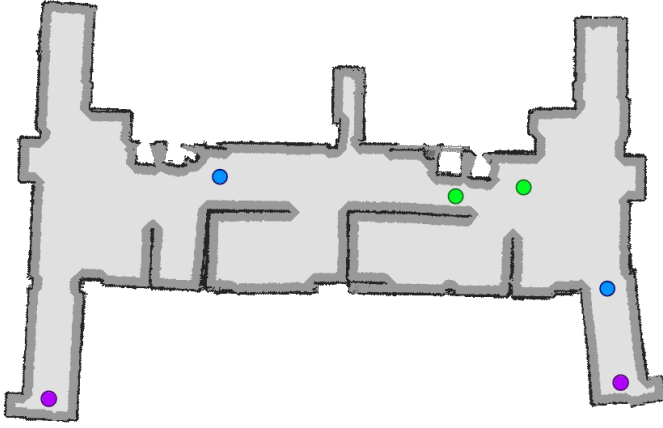
Fig. 14: Map showing attendee locations in the final experiment.

second room was detected. We ran the experiment once for each of the three states, from both of the starting locations, and averaged the time taken when starting from the two different locations to obtain an estimate of the time it would take to find an empty meeting room.

3) Results: State two is the worst case scenario since the robot must do a full scan of both rooms, whereas in state three the presence of a face in the first room means it does not do a full scan. For this state, it took on average 2 minutes and 22 seconds for the robot to finish scanning the second room. In state one, the robot took on average 1 minute and 25 seconds. In state three, the average time was 2 minutes and 22 seconds. The robot successfully identified the nearest room correctly from both locations. The robot also successfully planned routes to the centre of rooms and also routes between the two rooms in order to check them.

G. Gathering meeting attendees

1) Aim: The aim of this experiment is to test the robot's ability to gather meeting attendees and take them back to the meeting room within the 15-minute time limit. This experiment and the previous experiment are the most important as they allow us to analyse the robot's performance for the whole task.

2) Method: We came up with three test cases to check the system's ability to deal with different states of the world. The attendees were placed at short, medium and long distances from the door of the meeting room, as shown in Figure 14. The medium and long distance locations were chosen to make sure that the exploration strategy found people in different areas of the map. We ran the experiment until two runs without human intervention were completed for each state of the world. We recorded the average time of the experiment, the number of times it missed attendees during successful runs and finally the number of failed attempts—experiments which required us to intervene.

3) Results: Figure 15 shows the results of the trials. Even with the worst case scenario time, averaging 11 minutes and 55 seconds, adding the 2 minutes and 22 seconds worst case time to locate a meeting room does not exceed the 15-minute time limit. However, people in view were sometimes not correctly detected. This is usually caused by fewer than

the three frames required to confirm a face being received. During the experiment we had many issues related to AMCL localising to an incorrect position, which caused the robot to collide with walls and require intervention. If this happened, the experiment was abandoned. We were unable to identify the reason for AMCL losing its position frequently because it was not our implementation. When there were few or no issues with AMCL, the system worked well. Empty meeting rooms were found correctly each time, and attendees who were not detected the first time they were passed were found when the exploration returned to that location. Most importantly, we remained within the 15-minute time limit required by the task.

	Short	Medium	Long
Average time	4:24	9:19	11:55
Missed persons (avg)	1	2.5	2
Failed attempts	4	5	10

Fig. 15: Data for locating attendees and taking them back to meeting rooms.

V. Discussion

Once our experiments provided the optimal configuration for the variables in our system, our full system tests showed that the robot completed the task within the 15-minute time limit even in our worst case scenario. The robot stayed within the task specification, finding empty rooms and attendees and taking attendees to the meeting rooms. However, the task was not always completed because the robot sometimes collided with walls and other objects. This was mainly due to issues with AMCL, as discussed in Section IV-G3.

When implementing individual components of the system, we found that the ROS publisher/subscriber model was sometimes helpful while at other times a hindrance. For example, this structure helped when developing the vision node since the main system was written in Java while the vision node was written in C++. Using the standard ROS architecture, there was minimal complexity in passing data from the C++ node to the Java node, whereas usually this is more complex.

We found during our implementation that generating intelligent behaviour in a robot is more difficult than it first seemed. This is due in part to the fact that the environment is non-deterministic and dynamic, and the sensors used to gather data are prone to noise. Furthermore, there are a very large number of scenarios—some of which seem obvious or trivial at first—that need to be considered during both design and experimentation, to increase the system's reliability and performance. For example, although we designed our system to avoid obstacles and plan around them, in the real world environment, obstacles which were detected, such as humans, would often disappear later and this had to be accounted for.

While the individual components and techniques we used in our robot were chosen using sound experimental analysis, there were some drawbacks to using them. Results showed that AMCL performed better for localisation than our own implementations. However, since we did not implement AMCL ourselves, we could not find or correct issues with it when they occurred. By using our own PRM implementation we had the advantage of full control over the code, as well as

full understanding of how it worked. On the other hand, the obstacle avoidance part of the navigation code needed improvement and due to time constraints we could not perform this improvement. This had a negative effect on our system's overall performance and was responsible for failed runs in some of our experiments. Using OpenCV for face detection worked well, but caused some problems. Firstly, processing images from the Kinect's camera was very slow, typically taking 300–400ms to detect faces using a single cascade on a single image. This meant that when the robot was moving, the face's detected position would vary greatly between detections. Another issue was the high rate of false positives for all of the Haar cascades. However, these issues did not prevent us from creating a highly successful vision node when coupled with the Kinect's depth data. Using the Kinect vastly improved our vision node and was easy to set up and use. The main disadvantage of the Kinect was the low resolution of the camera, but our vision node did not suffer from this.

A. Potential Improvements

Although the vast majority of bugs were fixed, we were unable to determine the root of some issues which occasionally caused the system to crash. For a robot to be truly autonomous, it should have sufficient error handling capability to prevent minor issues causing the system to stop. Improvements could be made to the exploration path generation by using heuristics. The current method occasionally generates paths which are inefficient, requiring a lot of rotation, which slows down the exploration. A more advanced sampling method such as medial axis sampling could be used in the PRM in order to generate paths which are less likely to result in collisions. To further decrease time taken to gather attendees, any person seen while returning to the meeting room could be noted, and instead of resuming exploration, locations where people might be could be checked first. The use of multiple robots could further increase the efficiency of both exploration and attendee collection, although this would increase the complexity of the system. Improved human-robot interaction could also be considered. Rather than communicating with attendees using a small set of sounds, making announcements in natural language—perhaps using text-to-speech—would give the attendees more information about what the robot is doing.

VI. CONCLUSION

In this report we have presented our system which locates an empty meeting room, finds people in the vicinity and guides them to a meeting using a map, meeting room locations and a starting position. Although our system is not devoid of issues, our experiments show that it completes the task within the required time limit, and we have suggested some ways in which the implementation could be improved. Our system makes use of the Kinect, a relatively new piece of hardware, displaying one application in which the functionality it provides can be used to design improved vision systems. We have experienced for ourselves the difficulties of implementing a robust real-time system which must interact with a dynamic world, and have become familiar with many fundamental techniques in robotics including localisation, probabilistic roadmapping and face detection. What we have learned from implementing this system will surely be useful in other areas of computer science, and stands us in good stead for the future.

REFERENCES

[1] D. Kortenkamp, I. Nourbakhsh, and D. Hinkle, "The 1996 aaai mobile robot competition and exhibition," in *AI Magazine*, vol. 18, 1997.

[2] W. Burgard, D. Fox, and S. Thrun, "Active mobile robot localization by entropy minimization," in *Advanced Mobile Robots, 1997. Proceedings., Second EUROMICRO workshop on*, pp. 155–162, IEEE, 1997.

[3] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99–110, 2006.

[4] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents Series, MIT Press, 2005.

[5] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[6] R. Geraerts and M. Overmars, "Sampling and node adding in probabilistic roadmap planners," *Robotics and Autonomous Systems*, vol. 54, no. 2, pp. 165–173, 2006.

[7] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," tech. rep., Iowa State University, 1998.

[8] S. M. Lavalle, J. J. Kuffner, and Jr., "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2000.

[9] S. Thrun, "Exploration in active learning," *Handbook of Brain Science and Neural Networks*, pp. 381–384, 1995.

[10] S. Koenig, C. Tovey, and W. Halliburton, "Greedy mapping of terrain," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 4, pp. 3594–3599, IEEE, 2001.

[11] 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*, pp. 146–151, IEEE, 1997.

[12] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, "Coordinated multi-robot exploration," *Robotics, IEEE Transactions on*, vol. 21, no. 3, pp. 376–386, 2005.

[13] R. Zlot, A. Stentz, M. Dias, and S. Thayer, "Multi-robot exploration controlled by a market economy," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 3, pp. 3016–3023, IEEE, 2002.

[14] M. Yang, "Face detection," *Encyclopedia of Biometrics (eds. SZ Li)*, 2008.

[15] E. Hjelmås and B. Low, "Face detection: A survey," *Computer Vision and Image Understanding*, vol. 83, pp. 236–274, 2001.

[16] P. Wilson and J. Fernandez, "Facial feature detection using haar classifiers," *Journal of Computing Sciences in Colleges*, vol. 21, no. 4, pp. 127–133, 2006.

[17] P. Viola and M. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.

[18] Adept Technology, Inc., "Pioneer 3-DX datasheet," 2011.

[19] L. Hokuyo Automatic Co, "Scanning laser range finder URG-04LX specifications," 2008.

[20] M. Andersen, T. Jensen, P. Lisouski, A. Mortensen, M. Hansen, T. Gregersen, and P. Ahrendt, "Kinect depth sensor evaluation for computer vision applications," tech. rep., Department of Engineering Electrical and Computer Engineering, Aarhus University, 2012.