Fiducial Marker Navigation for Mobile Robots

Luke Ross
Department of Computer Science
Rhodes University
Grahamstown, South Africa
g09r5654@campus.ru.ac.za

Dr Karen Bradshaw
Department of Computer Science
Rhodes University
Grahamstown, South Africa
k.bradshaw@ru.ac.za

***Abstract*—Regarding mobile robots, navigation using fiducial markers has already been achieved and obstacle avoidance using search algorithms is common. However, the combination of these two ideas is relatively uncommon, and therefore, is the focus of this study. Owing to the mobility of these robots, good performance of the system is of utmost importance. Furthermore, since odometers on mobile robots are often innaccurate, and since other means of localization can prove complex and thus reduce performance, this system does not rely on previous map information. The relevant options available for the creation of this hybrid system are investigated, and an efficient system, capable of meeting its objectives, is designed and implemented on the WifibotLab LIDAR mobile robot. Under testing, the system showed good performance and proved accurate in terms of marker detection and the routes determined. Although various complicated scenarios were not feasible, the system was usable in practical scenarios, and therefore viable for integration and use on other similarly equipped mobile robots.**

***Index Terms*—Mobile robot motion-planning, Robot vision systems, Pattern recognition, Algorithms.**

I. Introduction

Fiducial marker navigation on mobile robots has been implemented successfully by researchers in various applications such as monitoring underwater domains using amphibious robots [1]. However, many implementations of these kinds of systems do not optimize the paths that the mobile robots travel. In most robot navigation systems in which minimal paths are calculated whilst avoiding obstacles, target tracking is not done using fiducial markers; instead methods based on GPS coordinates are more commonly used.

This project aims to add to the areas of mobile robot navigation and obstacle avoidance techniques. Since most mobile robots have low to moderate processing power owing to battery life constraints, significant emphasis is placed on the efficiency of the system built.

First, an investigation into the tools available for marker detection and the search algorithms, to calculate paths avoiding obstacles, suited to this study is conducted. Using this contextual information, decisions regarding the design of the system are discussed, followed by details of the implementation of the system. This system must satisfy certain criteria and therefore the objectives for this research are as follows:

- Detect fiducial markers accurately using a standard web camera.
- Navigate a mobile robot using a fiducial marker in a clear and obstacle-filled environment.
- Ensure that the system performs well, and is therefore suitable for less powerful robots.

This paper is structured as follows: *Section II* discusses literature on fiducial marker detection and obstacle avoidance, previous research conducted in these areas and the various options available for designing the proposed system; *Section III* covers the design, the reasoning behind the design decisions made, and the implementation of the system; *Section IV* compares the accuracy of the system with its performance; and *Section V* concludes the paper and mentions possible further work.

II. Background

This section reviews literature on fiducial markers, the toolkits available for fiducial marker detection and various search algorithms suitable for robot path planning. Current systems that implement fiducial markers for navigational purposes as well as systems that implement efficient path planning are also discussed.

The chosen toolkit for marker detection needs to be reasonably accurate and, owing to the processing and memory constraints on many mobile robots, not too computationally demanding. In addition, the chosen search algorithm needs to be capable of calculating the optimal path from the robot to the marker, avoiding obstacles. Furthermore, since the algorithm is used in a replanning manner, the only other requirement is that it must perform quickly.

A. Fiducial Markers

Fiducial, or fiduciary markers are image like objects, which are designed to be detectable and which usually contain an interpretable meaning [2]. Fiducials are used in many fields, including augmented reality, robotics and medicine. Such markers offer performance, identification and localization improvements, and are more cost-effective when compared to other techniques used for path-planning in unknown environments [3]. According to Owen et al. [2], two-dimensional, barcoded, square fiducial markers are the best and most popular form of marker to use.

B. Libraries for Marker Detection

Two libraries that can be used for marker detection are ARToolKit [4] and ArUco [5].

1) ARToolKit: is a successful, robust marker based system, commonly used in the augmented reality field. ARToolKit is open source for non-commercial use and widely used [6], meaning that there is extensive research material available. Furthermore, according to the feature list on the ARToolKit home page [4], ARToolKit supports multiple platforms and has real-time detection for two-dimensional markers. Hornecker and Psik [7] point out some disadvantages of using ARToolKit, one of which is that it does not recognize a marker if the full marker is not in the camera's view and Hirzer [8] claims that ARToolKit has a high false positive rate.

ARToolKit has been used extensively in marker tracking and augmented reality and, according to the "Projects" tab on its homepage, is used globally by over 300 researchers for a wide variety of purposes. Past applications that use ARToolKit vary from creating virtual environments in which ancient artifacts can be restored and viewed [9], to tracking human body movements [10].

2) ArUco - Based on OpenCv: is a basic C++ library used for the detection of fiducial markers and intended for augmented reality purposes [11]. It is based on OpenCv (Open Source Computer Vision), a vision based library, which is the most popular library in the computer vision field [11]. According to ArUco's homepage [5], with ArUco, markers can be detected using one line of C++ code; 1024 different markers are available; marker detection is fast and reliable since based on OpenCv; multiple platforms are supported; examples and sample code are available; and subject to possession of a BSD license, detection can be achieved at low-cost.

Speers et al. [1] reported using an amphibious, autonomous robot to monitor underwater sensors. In this system, fiducial markers are displayed on sessile sensors and are used to communicate with the robot using the ArUco library. ArUco has also been used in various unpublished projects, some of which are listed on the ArUco homepage. These projects include: the Soldamatic Project, the purpose of which is to aid in the training of welders by using augmented reality to create welding simulations, as well as OpenSpace3D, which is used for developing interactive, real-time 3D projects.

C. Search Algorithms

Search algorithms have been used extensively in the fields of Computer Science and Robotics in a wide variety of applications. Examples are: computer games [12] such as computer chess[1], Google's famous search engine[2], and robot path planning.

The robot in this project needs to be able to travel towards the fiducial marker in an environment that may contain obstacles. Since the robot should travel along the shortest path, a suitable search algorithm needs to be implemented such that the optimal path, avoiding obstacles, from the robot's current position to the marker is chosen. Furthermore, an efficient search algorithm needs to be implemented to keep computation to a minimum. Since the robot's environment is unknown, map information needs to be acquired using its LIDAR (Light Detection And Ranging) sensor and a map representation constructed.

1) Representing the Environment: A specific robot configuration can be represented using a state, and the distance between two states can be represented using an arc. Therefore, a graph of these states, joined together by arcs, can represent a robot's environment. The robot's environment, or map information, needs to be stored internally in a structure on the robot. This will allow the robot to store map information whenever it is acquired from its sensors, plan an optimal path and know where to move. Two common structures that can be used to store map information are metric and topological maps.

Metric maps are usually implemented using occupancy grids. According to Thrun and Bucken [13], in the area of mobile robotics, occupancy grids are the most successful environmental representations. Occupancy maps are easily implemented and simple to use [14][15]. A disadvantage of using grid-based maps such as occupancy grids is that, when a high resolution grid is required, memory requirements can become an issue [14].

Topological maps represent the environment using a graph [13]. This graph consists of a collection of nodes connected by arcs. The nodes correspond to distinctive landmarks while the arcs correspond to the distances between these landmarks. Unlike occupancy grids, the resolution of a topological map is purely dependent on the complexity of the environment. This allows for faster planning and less memory requirements than when using an occupancy grid [15]. Topological maps have disadvantages in that they are difficult to implement and maintain in large evironments and that similar landmarks are often ambiguously recognized, resulting in inaccurate map information [15][13].

2) The A Search Algorithm:* This is an extension of Dijkstra's algorithm [12], and is one of the most widely used search algorithms in the field of Artificial Intelligence [16]. A* performs faster than Dijkstra's algorithm by using heuristics [12]. According to Sun et al. [16], the moving target search problem can be solved using A* in a dynamic environment to calculate the path with the lowest cost between the current start and goal states whenever a change in the environment or a deviation of the goal state from the current path occurs. This is computationally expensive as the new path has to be planned from scratch each time a change occurs [16].

III. System Design and Implementation

After the criteria for the system were outlined and various options available for the construction thereof were investigated, design decisions concerning these options were made. The system was then implemented using C++ on the robot itself.

The first section covers the detection and tracking of the marker, after which algorithms for avoiding obstacles and searching for the marker, are discussed. Finally, the overall

[1] http://verhelst.home.xs4all.nl/chess/search.html

[2] http://www.techi.com/2012/03/googles-search-algorithm-changes-1998-2012/

picture detailing how the different parts of the system fit together, is presented.

A. Fiducial Marker Tracking

The two libraries that were considered for tracking fiducial markers were ARToolKit and ArUco. Both of these libraries seemed likely candidates as they are both open source, cross-platform and, even though little information was found on the use of ArUco, both libraries are well documented. Although ARToolKit is robust, popular and widely used, technical issues are common concerning its setup and it is outdated. Since ArUco seemed just as competent and is up-to-date, it was chosen instead. Using the combination of ArUco with OpenCv made streaming from the web camera and processing the received frames simple and neat.

To keep track of the marker, the robot's web camera must pan, keeping the marker in its view. Therefore, after the marker has been detected, the center needs to be determined. If the center of the marker is not within two vertical thresholds, the camera must pan to keep the marker's center within these thresholds. These two thresholds should be set to 40% and 60% of the resolution width. *Figure 1* shows the marker being detected using ArUco with the thresholds edited in.

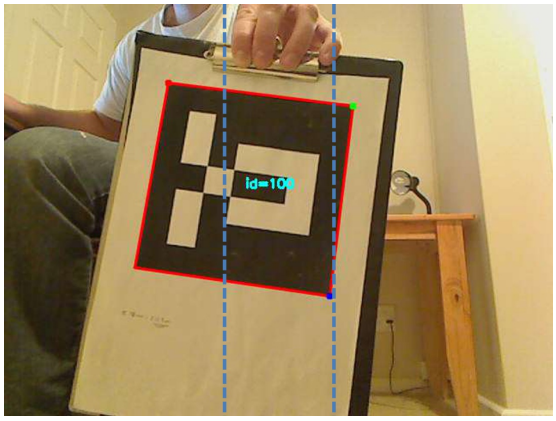


Fig. 1: Detection with ArUco

Since the robot needs to follow the marker and not just the camera, once the camera has panned a certain amount the robot should turn towards the marker and while the robot is turning, the camera should remain centered on the marker. The robot should then keep turning until its front has lined up with the camera's front.

Forward movement commands are issued to the robot when the marker is detected and the robot is not turning. This means that if the marker is moved to the left or right in the robot's view, the robot continues moving forward while the web camera pans. When the camera pans past one of the web camera thresholds, however, the robot stops moving forward and turns in the correct direction until it is facing the marker directly. The marker following process then continues. All this is done in a loop that iterates as fast as it can process frames, and therefore, a lower frame resolution may be crucial to ensure acceptable responsiveness of the system as a whole. While following the marker, at some stage the robot could catch up to the marker. If this happens, a stop command is sent to the robot when the distance between it and the marker is less than a certain threshold (750 mm for the purpose of this study).

B. Obstacle Avoidance

Owing to inaccurate odometry being a common issue with many inexpensive robots, this project's aim was to create a system that would allow a mobile robot, the Wifibot robot in this case, to follow a fiducial marker, optimally avoiding obstacles, whilst not relying on odometry at all. A new obstacle avoidance algorithm was therefore designed.

1) Algorithm Description: This algorithm uses A* in a similar manner to when it is used in a dynamic environment as discussed in *Section II*. However, instead of replanning whenever a discrepancy in the map is found or the goal moves, the path is replanned every few hundred milliseconds. Therefore a path is calculated only to determine in which direction the robot should move next. The robot is then issued the appropriate command and a new path is calculated. Since no information regarding the previous path calculated is reusable, using an incremental search algorithm would not result in improved performance. This is the reason that the A* search algorithm was chosen. The pseudo code for this algorithm is as follows:

```
    while true:
1.      if marker detected directly in front of robot AND path
        ahead of robot is clear:
            break out of loop and follow marker normally
        else if marker lost:
            break out of loop and search for marker
2.      lidar scans for obstacles
3.      obstacle data received from lidar is converted to
        occupancy map coordinates through conversion process
4.      obstacles are placed in the occupancy map using these
        coordinates
5.      fixed position of robot is placed in occupancy map
6.      goal coordinates are calculated using distance from the
        robot to marker and angle between direction that
        robot faces and marker
7.      goal placed in occupancy map using goal coordinates
8.      A* search algorithm used to calculate optimal path
        from robot to marker
9.      path calculated is analysed to determine robot action
10.     robot given the appropriate command
```

According to this pseudo code, after A* is used to calculate the path, the path is analysed to determine which move command the robot should be given. This is done by looking at the first cell of the occupancy map that the calculated path occupies - the cell that the robot needs to move to first. The robot is then issued a turn left, turn right, or move forward command based on this cell before the process is repeated.

2) Constructing the Map Representation: As mentioned in *Section II*, two-dimensional occupancy maps are the most successful environmental representations in the area of mobile robotics. For this reason, and owing to their simplicity, which makes for easy understanding and implementation, this representation was chosen over the more complicated topological map representation to represent the robot's environment in this project.

The URG-04LX-UG01 LIDAR[3] from Hokuyo is the LIDAR attached to the robot used in this research. When scanning the robot's surroundings, the data returned from this LIDAR is stored in an array containing the distances (in mm) to obstacles from the LIDAR. To construct the occupancy map, after the LIDAR has returned this array, the data in the array is converted into (x,y)-coordinates and then into occupancy grid coordinates. The obstacles are then placed in the occupancy grid. After a path has been calculated from the robot to the goal, avoiding the obstacles, and the robot has been issued a move command, the above process is repeated continuously to acquire new map data, thereby updating the path to the goal.

If the amount of area that each cell represents is smaller than the area that is occupied by the robot, additional "fake" obstacles are inserted into the occupancy grid along with each actual obstacle so that each obstacle makes up an area of 440 mm by 440 mm, an area slightly larger than the robot occupies. This eliminates the possibility of calculating paths that are too narrow for the robot to traverse. Since the LIDAR scans in an arc shape and the occupancy grid is a square, a ring of "fake" obstacles, two obstacles deep, is placed surrounding the robot's scan area. Two lines of "fake" obstacles, once again two obstacles deep, are also placed from directly behind the robot to the ring of obstacles along the 120 degree and -120 degree scan line from the robot's LIDAR. The ring and the two lines are placed such that paths that are out of the scan area specified by the user or in the dead zone of the LIDAR cannot be calculated. These additional "fake" obstacles are illustrated by *Figure 8*.

C. Searching

When the marker is lost, the robot and its camera remain stationary for 5 s, thus allowing sufficient time for the user to display the marker in the camera's view. If the marker has not been detected by the time that this period has elapsed, the marker is searched for by panning the web camera all the way in one direction and then all the way in the other direction. The direction in which the web camera initially pans is the direction in which the marker was last detected. If the marker has still not been detected, the robot performs a turn of roughly 360 degrees (roughly, since the odometry is not measured). The direction of this turn is towards the direction in which the marker was last detected.

At any time during this process, if the marker is detected, the process terminates and the system resumes following the marker. If the marker has not been detected after these three stages have been performed, the searching process is repeated.

[3]http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html

D. Putting it all together

The state of the robot at any time is either: following the marker normally when no obstacles are in its path, avoiding obstacles in an attempt to follow the marker normally, or searching in an attempt to find a lost marker. This means that the robot can be in one of three different modes at any time. After all the devices have been initialized and all the startup processes performed, the system enters the main execution loop. When entering the loop, mode 3 - the searching mode - is the default and thus this is the robot's initial mode. The searching process is carried out until the marker is detected. On detection of the marker, the robot switches to mode 1 - following the marker normally, and begins following the marker. The robot remains in mode 1 provided that no obstacles are detected in the robot's path, meaning that no obstacles are detected within a rectangle of 0.44 m (marginally wider than the robot's width) by 1 m directly ahead of the robot. If an obstacle is detected inside this rectangle, the system switches into mode 2, obstacle avoidance mode. For a transformation from mode 2 back to mode 1, the robot needs to have successfully avoided any obstacles in its path to the marker. Then, the system changes back into mode 1 if either the marker is detected directly ahead of the robot and the path ahead of the robot is clear, or the marker is detected and its distance from the robot is 750 mm or less. If the marker is lost in either mode 1 or mode 2, the robot switches to mode 3. Whenever a switch to mode 3 occurs, the mode that is being switched from is recorded so that when the marker is detected again, the previous mode can be reinstated. The switching between modes is represented visually using a state diagram in *Figure 2*.

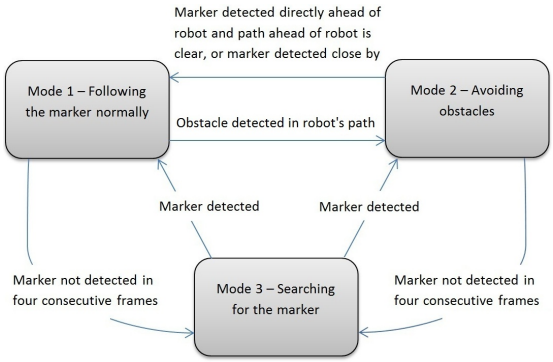


Fig. 2: State diagram of mode switching

In order for this system to be implemented successfully, three loops are required, two of which are run in separate threads that are spawned at startup. The first of these is the main loop for the system. This loop receives frames from the camera, detects the marker in the frames, pans the web camera, sets a flag to indicate that the robot's motors should be issued specific commands provided the system is in mode 1 or mode 3, and displays the video feed showing the detected marker. Note that in this loop, a flag indicating that the robot should

move in a certain direction is set. This flag is checked in the second loop, which is where the actual commands to the robot's motors are issued. If the flag is set to move forward and is not changed for the next few seconds, the move forward command is continuously issued to the motors. This is because commands must be issued to the robot at least every 250 ms to prevent a timeout occurring which resets the speed to zero, and for this reason, this process needs to be implemented using a simultaneously executing loop. This loop runs in its own thread, the *robotmove* thread.

The third loop is used to read data from the LIDAR and indicate to the main loop whether the path ahead of the robot is clear. When the system is in mode 2, this loop also handles the obstacle avoidance process and the display of the optimal path, calculated to avoid the obstacles. When calculating the optimal path avoiding the obstacles in mode 2, this third loop, like the main loop, sets a flag instructing the robot's next move. The search algorithm is therefore called from this loop. Since data from the LIDAR must be read continuously, the loop has been implemented to run in its own thread, which is called the *readlidar* thread.

These three loops, along with some of the system's main functionality, are shown in *Figure 3*.

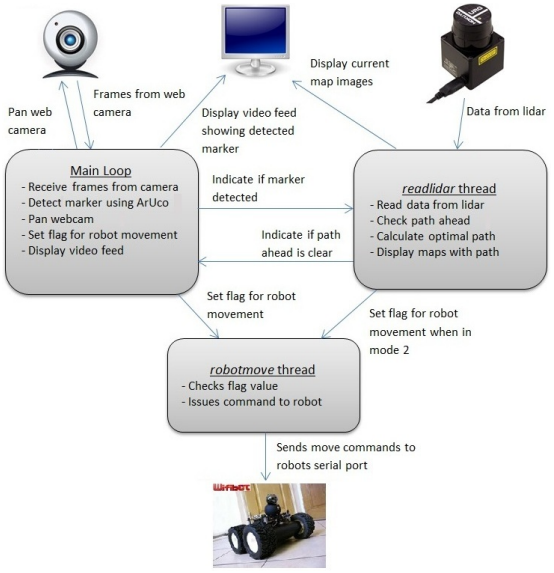


Fig. 3: Information flow between looping threads

IV. Results and Analysis

In this section, experiments measuring the accuracy and the performance of the system while adjusting various variables are discussed, the results presented and an analysis of these results given. Decisions made regarding the final system configuration are then mentioned, after which the system's capabilities are discussed.

A. System Accuracy and Performance

1) Marker Detection using ArUco: Regarding the design of the fiducial marker, the size of the marker was varied in an attempt to gain accuracy by increasing the size, or to reduce the amount of visual distraction by reducing the size. Accuracy can also be gained by increasing the resolution of the web camera, or performance increased by decreasing this resolution. Three different sized markers, with side lengths measuring 130 mm, 160 mm and 200 mm, were printed out for testing with three different resolutions, 160x120, 320x240, and 640x480. The maximum distance along the ground from the web camera, from which each of the markers was detected perfectly, without being lost in any frames, was then measured for each resolution. These tests were performed in an artificially well lit environment.

A graph showing the results obtained from this procedure is given in *Figure 4*. This graph clearly shows that by increasing the marker size, the maximum distance at which the marker can be detected, increases in a reasonably linear manner. Furthermore, it can be seen that when setting the camera to a resolution of 320x240, as compared to 160x120, a large increase in accuracy is observed. However, when further increasing the resolution to 640x480, accuracy increases only marginally. The rate at which the maximum distance increases as the resolution increases does not seem to be linear, but rather logarithmic.

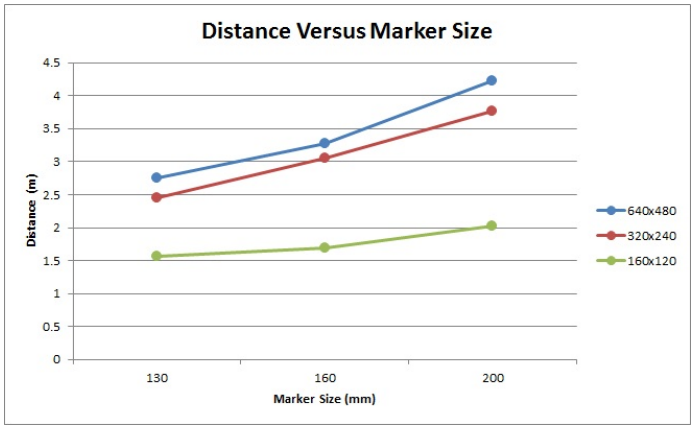


Fig. 4: Distance versus marker size for different resolutions

On the Wifibot robot, when using a resolution of 320x240, frames were processed at their maximum speed which was capped by the frame rate of the web camera, an increase of 171.4% from that when using a resolution of 640x480.

2) Occupancy Grid Design - Area Represented: As with the marker design, there were several options concerning the design of the occupancy map. These options affect the optimality of the paths calculated and the time taken to calculate them. It is important that the paths calculated are as close to optimal as possible as unnecessary distance traversed is wasteful when considering that most mobile robots operate on battery power. Owing to the nature in which the A* search algorithm is used, these paths need to be calculated quickly, otherwise move commands may be given to the robot when they are no longer appropriate, causing the robot to behave

erratically and possibly make contact with the obstacles.

The first choice regarding the design of the occupancy grid, is how much of the environment surrounding the robot should be represented. This is easily done by ignoring data received from the LIDAR that is further away than a specified distance, when building up the occupancy grid. Consider the example where the goal is placed directly in front of the robot, at a distance of 5.6 m (the maximum range that the LIDAR can scan). Therefore, when the LIDAR range is limited, the goal is contained in the cell directly in front of the robot, but at a distance further than that which the occupancy map can represent. When the robot begins to move towards the goal, it remains on the outskirts of the grid until it is closer than the range to which the LIDAR data is limited. Keeping this example in mind, limiting how much environmental data is represented by the grid, results in the occupancy grid having fewer cells. If the grid has fewer cells, when the A* algorithm is used to calculate a path from the robot to the goal, fewer cells need to be considered, thus boosting performance. This comes at a cost though. Not only might the path that the robot takes to the goal be unnecessarily longer than the optimal path, but incorrect decisions regarding which direction the robot should move in to avoid obstacles could be made, perhaps causing the robot to traverse into a dead-end.

Three different sized occupancy grids were tested. These grids represent the area within 1 m, 3 m and 5.6 m, respectively, from the LIDAR at all angles that the LIDAR is capable of scanning. These distances are hereafter referred to as the LIDAR's scan range. Tests on each grid were performed and the system's performance measured. For each run of the experiment, the goal was set 5.6 m away from the robot. Since the performance of the system differs according to different environmental layouts, all runs were performed keeping this layout unchanged.

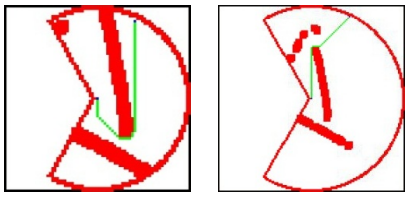
To record the times taken for calculating paths on different sized grids, the robot needed to be kept stationary and so commands to the robot's motors were stopped. For each of the three tests, the time taken for the path, from the robot to the manually placed goal avoiding obstacles, to be calculated was recorded. These tests were repeated 20 times to achieve more accurate results for each of the occupancy grids, and the average times calculated. The results obtained from these tests are presented in *Table I*.

TABLE I: Scan range effect on performance

Distance (*mm*)	**Total Cells**	**Ave Time (*ms*)**
5588	131x131=17161	145.1
3036	73x73=5329	33.8
1012	27x27=729	1.6

Since the amount of area that the grid represents is proportional to the total number of cells in the grid, these are included in the table. Even though many of the cells in the grid are never considered owing to the ring and two lines of "fake" obstacles placed on the grid, this does not affect the relationship between the number of cells in the grid and the performance of the two tests. As can be seen in the table, the relationship between the total number of cells and the time taken for the path to be calculated, is far from linear, but rather as the number of cells increase, the performance decreases at a much faster rate.

Although the performance is significantly reduced when a larger scan range is used, the overall paths calculated are closer to being optimal. This is because more of the environment is considered each time paths are calculated and therefore, more informed routes are chosen. Under testing, this proved to be the case. In certain scenarios, shorter paths are chosen when larger scan ranges are used. One can see this by considering the scenario illustrated in *Figures 5(a)* and *5(b)*. In this scenario the goal has been set to the left of the robot, 5.6 m distant. The thin line shows the robot's path.



(a) 3 m (b) 5.6 m

Fig. 5: Exaggerated maps using different scan ranges

In *Figure 5(a)*, the scan range has been limited to 3 m. Here an incorrect path to the goal has been calculated since crucial information regarding the environment has been ignored. In *Figure 5(b)*, the scan range is set to 5.6 m, and therefore more information regarding the robot's surroundings is taken into consideration, allowing the shortest path to the goal to be calculated.

Although the above scenario shows a case where it is preferable to have a larger scan range, in practice, these scenarios are not common and it is most often unnecessary. This is because the LIDAR cannot detect obstacles behind obstacles, causing many openings and blocked off areas further away, to remain undetected.

3) Occupancy Grid Design - Grid Resolution: The second decision regarding the occupancy grid's design, is how fine the resolution of the grid should be. This is done by varying how much of the environment each cell in the grid represents. As the resolution of the occupancy grid is increased, the number of cells in the grid increases (cell size reduces), thus representing the environment more accurately. This allows for smaller openings to be traversed and tighter corners around obstacles to be taken, thus reducing the distance to the goal. However, as when increasing the scan range, when increasing the resolution, performance decreases. When the resolution decreases, the opposite occurs - an increase in performance is noted, but the representation of the robot's environment is less detailed. Openings in the environment that could be traversed are often not detected and when traversing around obstacles, extra distance is travelled.

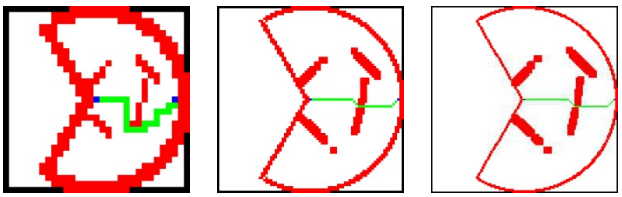
For this experiment, the side length (cell size) of each cell in the occupancy grid was set to represent distances of 88 mm, 148 mm and 440 mm. To measure how the system performs when the cell size is varied, the same test as when varying the scan range, was repeated, but this time with the LIDAR scan range set back to 5.6 m and the cell size varied. The same environmental layout was used. *Table II* shows the results obtained from this experiment.

TABLE II: Grid resolution effect on performance

Cell Size (*mm*)	Total Cells	Ave Time (*ms*)
88	131x131=17161	145.1
148	79x79=6241	34.4
440	29x29=841	1.6

Once again, as the number of cells in the grid increases, the performance decreases rapidly. Since the number of cells in the grid depends on the size of the cell, making adjustments to the scan range or the cell size has the same impact on performance.

Figures 6(a), *6(b)* and *6(c)* show the differences in grid accuracy when adjusting the cell size, and therefore the optimality of the paths calculated. In this scenario, two long obstacles were placed close to one another, creating an opening that the robot could fit through fairly easily.



(a) 440 mm (b) 148 mm (c) 88 mm

Fig. 6: Exaggerated maps using different cell sizes

When using a cell size of 440 mm, the opening was not detected and an alternate route was calculated. The opening was still detected when using the other, higher resolution occupancy grids. However, in most scenarios, if openings are reasonably wide, the same paths are calculated as when using finer resolutions. The reason for this is that when a higher resolution is used, each obstacle detected is exaggerated to represent an area of 440 mm by 440 mm of the environment, and therefore the only difference when using a higher resolution grid is that obstacles are pin pointed onto the map with more precision than when a lower resolution grid is used. Note that when using a cell size of 440 mm, diagonal movement is prohibited as no extra ”fake” obstacles are placed, resulting in incorrect paths being chosen. However, in the results shown in *Table II*, diagonal movement was not prohibited for this case and so the timings shown are comparable with the other results.

4) Path Calculation: This section deals with the design of the A* search algorithm itself. As with the design of the occupancy grid, when adjusting the search algorithm's design, a trade-off exists between the optimality of the paths calculated and the time taken to calculate them.

The optimality of the paths calculated and the performance of the system when restricting movement to only four directions (horizontal and vertical) and when allowing it in all eight directions (diagonals movement allowed) were tested. When diagonal movement is allowed, more cells are considered when calculating the path and therefore more calculations are performed, reducing performance severely. However, as with all of these design decisions, there is a trade-off. When movement is restricted to four directions, paths calculated are not optimal and are most often far from being optimal. When only horizontal and vertical movement is allowed, the Manhattan heuristic is used and when diagonal movement is allowed, the Chebyshev heuristic is used. The Euclidian heuristic was also tested in the case of movement in four directions to see how this heuristic affected the optimality of the paths calculated and the performance of these calculations. Once again, the same experiment was repeated.

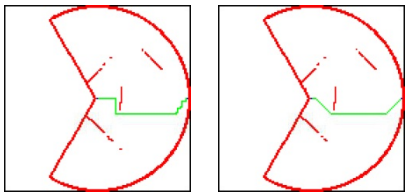
The results from the performance tests performed are presented in *Table III*.

TABLE III: Effect on performance based on the number of cells considered

Cells Considered	Heuristic	Ave Time (*ms*)
8	Chebyshev Distance	145.1
4	Manhattan Distance	19.2
4	Euclidean Distance	5.7

As expected, by limiting movement to four directions, performance increases dramatically. However, the 236.8% performance increase noted when changing from the Manhattan to the Euclidian heuristic was unexpected. This increase seems to be due to the smaller value calculated by the Euclidean heuristic, as well as the Chebyshev heuristic for that matter, whereas the Manhattan heuristic results in a larger value for each considered cell. Owing to the many mathematical calculations and comparisons that occur in the A* implementation, these larger values hinder the performance of the algorithm.

Concerning the optimality of the paths calculated, shorter, and less jagged routes are determined when diagonal movement is allowed, compared to when it is prohibited. The scenario illustrated in *Figure 7* shows an example of this.



(a) Four cells (b) Eight cells

Fig. 7: Different paths calculated when four and eight cells are considered

Not only is extra mileage added to the routes calculated,

when only considering movement in four directions, but in certain scenarios, openings detected cannot be passed though. This occurs in scenarios where the openings are fairly small, and at angles to the robot, and so a diagonal path needs to be calculated. Furthermore, when only considering movement in four directions, consecutive paths calculated differ quite drastically even when the map data and the goal are kept stationary, thus affecting the usability of the system. This is not the case when considering diagonal movement as paths rarely deviate from previously calculated paths.

B. Final Configuration and System Capabilities

After experimentation, the following decisions regarding the various parameters discussed were made:

- A marker size of 160 mm by 160 mm was used
- The camera resolution was set to 320x240 and all automatic lighting was disabled
- The occupancy grid was set to represent a limited range of 3 m from the LIDAR
- Each cell in the occupancy grid was set to represent an area of 148 mm by 148 mm of the environment
- Diagonal movement was allowed and therefore, the Chebyshev heuristic was used

Remarkable performance was noted when using the above configuration, while still allowing optimal, or close to optimal paths to be determined. If the system was implemented on a robot with lower specifications, these parameters could be set with lower values. Although accuracy regarding marker detection and/or the paths calculated would be reduced, a well chosen configuration of these settings should allow for sufficient accuracy.

Figure 8 shows a screenshot of the system captured while the robot is avoiding obstacles.

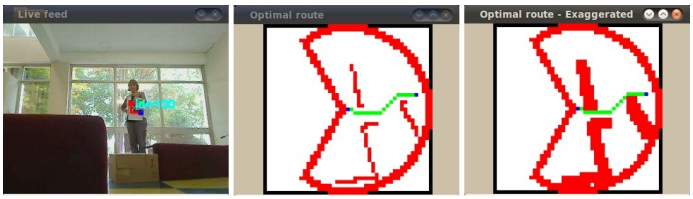


Fig. 8: System in action

Since previous map information could not be reused, the system was incapable of finding routes to the marker in certain complex scenarios. However, the system was successful in most practical scenarios.

V. Conclusions and Future Work

The objective of this study was the creation of a system that allows for the navigation of a mobile robot in open or busy environments using a fiducial marker. This needed to be achieved without localizing the robot and therefore previous map and path information was not reused. Furthermore, this system needs to perform efficiently since the system is targeted at mobile robots, many of which have low processing power. These objectives have been met as shown by the results presented in *Section IV*. Although not all layouts were traversable since previous map and path information was not reused, in most practical scenarios, navigation was possible and thus the system proved feasible.

Regarding future work, there are several extensions that can be made to the existing system. Firstly, since the web camera used, by panning, could only view 189 degrees of its surroundings, the marker could not be kept in view when avoiding obstacles in certain scenarios. This issue could be solved by either using a camera with a 360 degree panning capability, or at less cost, multiple standard web cameras.

Secondly, the issue of inaccurate odometer readings could be solved. An inexpensive solution is to add either two accurate trailing wheel encoders, or a single trailing wheel encoder and a digital compass, to the robot in use. Either of these options could provide the robot with accurate odometry as well as information regarding direction. Another, more interesting solution, would be to attach at least two optical mouse sensors, one on each side of the robot. Provided that the surface used is smooth, this would allow for the distance travelled as well as the direction of travel to be determined. After integrating one of these solutions, a search algorithm, perhaps Moving Target D* Lite [16] as it would be more suited, could be implemented in the manner that it is intended, thus resulting in a large increase in performance. A wandering algorithm could also then be implemented instead of the simple algorithm for searching discussed in *Section III*.

References

[1] A. Speers, A. Topol, J. Zacher, R. Codd-Downey, B. Verzijlenberg, and M. Jenkin, "Monitoring underwater sensors with an amphibious robot," in *Proceedings of the 2011 Canadian Conference on Computer and Robot Vision*, ser. CRV '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 153–159. [Online]. Available: http://dx.doi.org/10.1109/CRV.2011.27

[2] C. B. Owen, F. Xiao, and P. Middlin, "What is the best fiducial?" in *The First IEEE International Augmented Reality Toolkit Workshop*, Darmstadt, Germany, Sep. 2002, pp. 98–105.

[3] A. Mutka, D. Miklic, I. Draganjac, and S. Bogdan, "A low cost vision based localization system using fiducial markers," *World Congress*, vol. 17, pp. 9528–9533, 2008. [Online]. Available: http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac2008/data/papers/2280.pdf

[4] Artoolkit. ONLINE. [Online]. Available: http://www.hitl.washington.edu/artoolkit/

[5] ArUco: a minimal library for Augmented Reality applications based on OpenCv. ONLINE. [Online]. Available: http://www.uco.es/investiga/grupos/ava/node/26

[6] M. Fiala, "ARToolKit applied to panoramic vision for robotic navigation," in *Proceedings of the Vision Interface, Halifax, Nova Scotia, Canada, June 11-13, 2003*, 2003.

[7] E. Hornecker and T. Psik, "Using ARToolKit markers to build tangible prototypes and simulate other technologies," in *Proceedings of the 2005 IFIP TC13 international conference on Human-Computer Interaction*, ser. INTERACT'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 30–42. [Online]. Available: http://dx.doi.org/10.1007/11555261_6

[8] M. Hirzer, "Marker detection for augmented reality applications," *Image, Rochester, NY*, 2008. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Marker+Detection+for+Augmented+Reality+Applications#9

[9] F. Stanco, D. Tanasi, G. Gallo, M. Buffa, and B. Basile, "Augmented perception of the past. The case of Hellenistic Syracuse," *Journal of Multimedia*, vol. 7, no. 2, pp. 211–216, 2012. [Online]. Available: http://ojs.academypublisher.com/index.php/jmm/article/view/jmm0702211216

[10] A. C. Sementille, L. E. Lourenço, J. R. F. Brega, and I. Rodello, "A motion capture system using passive markers," in *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, ser. VRCAI '04. New York, NY, USA: ACM, 2004, pp. 440–447. [Online]. Available: http://doi.acm.org/10.1145/1044588.1044684

[11] J. Brunner. Aruco: Augmented reality library from the university of cordoba. ONLINE. [Online]. Available: http://www.ros.org/doc/api/aruco_pose/html/index.html

[12] M. Nosrati, R. Karimi, and H. A. Hasanvand, "Investigation of the * (star) search algorithms: Characteristics, methods and approaches," *World Applied Programming*, vol. 2, no. 4, pp. 251–256, April 2012.

[13] S. Thrun and A. Bucken, "Integrating grid-based and topological maps for mobile robot navigation," in *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 2*, ser. AAAI'96. AAAI Press, 1996, pp. 944–950. [Online]. Available: http://dl.acm.org/citation.cfm?id=1864519.1864527

[14] G. K. Kraetzschmar, G. Pagès Gassull, and K. Uhl, "Probabilistic quadtrees for variable-resolution mapping of large environments," in *Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, M. I. Ribeiro and J. Santos Victor, Eds. Lisbon, Portugal: Elsevier Science, July 2004. [Online]. Available: http://citeseer.ist.psu.edu/kraetzschmar04probabilistic.html

[15] S. Thrun and A. Bcken, "Learning maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, pp. 21–71, 1998.

[16] X. Sun, W. Yeoh, and S. Koenig, "Moving target D* Lite," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, ser. AAMAS '10. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 67–74. [Online]. Available: http://dl.acm.org/citation.cfm?id=1838206.1838216