Quick Goal Seeking Algorithm for Frontier based Robotic Navigation

Vaisakh V P
Department of Computer Science & Engineering
Amrita School of Engineering
Amrita Vishwa Vidyapeetham, Coimbatore-641 112

ABSTRACT

There arises situations where an autonomous robot needs to navigate to a target location and no information is available about the terrain. Frontier based navigation is one of the most efficient methods of exploration and navigation for such situations. In a frontier based strategy, the robot navigates to the target location by detecting intermediate frontier regions, which are points lying on the boundary separating the explored region from the unexplored. In this paper, a new frontier based robotic navigation algorithm called the Quick Goal Seeking (QGS) algorithm is proposed. The QGS algorithm is tested in a real-time environment and its performance is compared with two major frontier based navigation algorithms; Modified Goal Seeking (MGS) and Fast Frontier Detector (FFD). The QGS algorithm uses heuristic informed search for path planning. It consists of a highly optimized and efficient scanning function which minimizes the search space. The performance of these three algorithms is compared based on the total time taken to reach the destination. It has been found out that the QGS algorithm performs better than the MGS and FFD algorithms in almost all the cases.

General Terms:

Frontier based navigation, heuristic path planning, occupancy grid

Keywords:

Frontier cells, path planning, robotics, navigation, IR range sensors

1. INTRODUCTION

A variety of frontier based algorithms have been designed and developed since the year 1988. But not much work has been done in comparing the performance of these algorithms among each other in a real-time environment. In most of the research works, the developed algorithms have been tested only under a simulated environment. Practical implementation of these algorithms has always been tedious as a wide range of problems must be addressed while interfacing the hardware. For example, the sensor output data must be calibrated and averaged to reduce the inaccuracies and errors in the readings. The communication delays and inaccuracies in sensor readings considerably reduced the performance of these algorithms. Unlike in a virtual simulation, the practical implementation of the frontier based navigation algorithms exhibits a variety of problems which require in-depth study and testing.

1.1 Motivation

The development and implementation of the modified goal seeking (MGS) algorithm and fast frontier detector (FFD) algorithm provided means to understand the drawbacks and faults associated with it. The motive behind designing and developing a new frontier based algorithm is to overcome these drawbacks and to provide a faster navigation strategy. The algorithm incorporates an innovative alignment function and a modified environment scanning technique to increases the overall efficiency of the frontier based search.

1.2 Problem Definition

The problem currently defined is to design and implement a new frontier based navigation algorithm in a real-time environment and to compare its performance with the MGS and FFD algorithms, based on the time taken to reach the target location. Given a mobile robot, it must be able to generate a path between two specified locations, the start node and the target node. The path should be free of collision and must satisfy certain optimization criteria (least time consuming path). The only information available to the robot is its current position and the location of the target in the grid map [4]. The robot has to continuously move from the current position until it reaches the target by avoiding the obstacles detected on route. Occupancy grids [4] are used for the representation of the environment.
Here each cell in the grid contains information about its state, which is calculated depending on the probability of occupancy [4] of that particular cell. Hence a cell which is occupied by an obstacle will have a very high probability of occupancy value returned by the sensor, which makes it unavailable for the robot to traverse. Frontier based heuristic search algorithm is the main domain of the problem. It is useful in situations where no prior planning is feasible and all decisions are taken at real-time.

1.3 Organization of the paper

This paper is organized into five sections. Section-2 briefly explains the various frontier based navigation algorithms developed so far. The problem scenario, the proposed system architecture and the implementation details are explained in Section-3. Results of the

comparative study are discussed in Section-4. Section-5 concludes the findings and spreads light on the scope for future work.

2. RELATED WORKS

A variety of work has been done so far in the field of frontier based navigation. The concept was proposed by Brian Yamauchi in Frontier Based Exploration Using Multiple Robots [1]. Later V.R Jisha [2] proposed two frontier based navigation algorithms namely Basic Goal Seeking (BGS) algorithm and a better version of it called the Modified Goal Seeking (MGS) algorithm [2]. All the frontier based navigation algorithms made use of proximity sensors to detect and identify the frontier regions during the exploration. These algorithms were time consuming and had to process enormous amounts of data from various cells in the grid map. Later Matan Keidar and Gal A Kaminka [3] proposed four different sets of frontier based navigation algorithms; namely Fast Frontier Detector (FFD), Wave front Frontier Detector (WFD), Incremental Wave front Frontier Detector (WFD-INC) and Incremental Parallel Wave front Frontier Detector (WFD-IP) which were more efficient and faster than the state-of-the-art frontier based navigation strategies [3]. A brief description of these algorithms is given in the following section.

2.1 Basic Goal Seeking Algorithm

There are two goal seeking algorithms as proposed in Frontier Based Goal Seeking for Robots in Unknown Environment [2]. The first one is known as Basic Goal Seeking algorithms or BGS and the other one, which is a modified version of the former, is known as the Modified Goal Seeking algorithm or MGS [2]. During each execution cycle of the BGS algorithm, the robot performs a full round scan of its environment and updates the occupancy value (Po) of its four adjacent cells, one in each direction: top, right, bottom and left. These four cells detected on every scan are termed as the cells in the current sensing region [2]. As explained before, a frontier cell is a cell explored by the robot which is having at least one unexplored cell as its neighbouring cell. After each scanning operation, the newly detected frontier cells are assigned heuristic cost value known as Goal Seeking Index (GSI) [2]. The cost of moving to a cell (x, y) is found as the product of its occupancy value Po and the distance of the cell (x, y) from the current position of the robot [5]. Calculating the cost based on occupancy value is explained well in [5] . The GSI for each frontier cell is found out with the help of the equation given below.

$$GSIxy = ((D_{max} - D_{xy,target}) - C_{xy,current}) \qquad (1)$$

D_{max} - largest distance possible between any cell in the grid and the target location

$D_{xy,target}$ - distance between the given frontier cell (x,y) and the target cell

$C_{xy,current}$ - product of the occupancy value (Po) of the frontier cell (x,y) and its distance from the current location of the robot.

As the distance between the frontier cell and the target cell decreases, the GSI value keeps increasing. Hence after every loop execution, the algorithm chooses a FC with the greatest value of GSI. However it was found out that the performance of the BGS algorithm deteriorated when wall like obstacles were introduced in the grid. In order to overcome this problem, a new algorithm called the MGS algorithm was introduced.

2.2 Modified Goal Seeking Algorithm

The MGS algorithm is a modified and more efficient form of the BGS algorithm. The efficiency lies in the way in which frontier cells are chosen during the execution. In a grid filled with wall like obstacles, it was found out that the BGS algorithm was unreliable. The robot tends to get trapped in locations similar to a closed room with a narrow exit. In such situations the robot wasted large amount of time in exploring cells which were previously explored and hence efficient ways to get out of a trap situation was highly essential. This lead to the development of a more efficient MGS algorithm.
The MGS algorithm resolved this problem by defining the trap [2] situation clearly. After a complete scan of the environment, if no FC are detected in the current sensing region, the situation is called a trap or a local maximum. Once a trap situation is detected, the algorithm checks for the availability of previously detected FC which lie outside the current sensing region. If no such cells are available then it fails to reach the target cell and quits. But if such FC are available, then a best one among them is chosen based on certain strategies. Frontier cells which lie next to an obstacle, a FC which is closer to the target cell or a FC which is close to the current robot position are given more preference over the rest of the frontier cells. If multiple such cells are available, any one FC is chosen and the robot moves towards it trying to escape from the trap. The pseudo code of the MGS algorithm is given below.

WHILE the goal is not reached
  Identify the FC within the current sensing region
  IF FC present inside the current sensing region **THEN**
    Determine the Goal Seeking index of all the current FC and choose the one with the largest value
  ELSE IF FC present outside the current sensing region **THEN**
    Choose a FC which is near to an obstacle, closer to the target or closer from the current robot position
  ELSE
    All area has been covered and the goal is not present in the area. STOP.
  END IF
  Move towards the best FC chosen
END WHILE

Algorithm 1: Modified Goal Seeking algorithm pseudo code [2].

2.3 Wave front Frontier Detector

WFD is a graph-search-based method which uses breadth-first search (BFS) for frontier detection. The main feature of the WFD algorithm is that it does not search the entire map data during its each execution. Instead it only processes those locations which have already been scanned by the robot and its sensor and hence unknown cells are removed from the search space [3]. Initially the current robot position is pushed into a queue which determines the search order and in order to prevent the re-detection of the same frontiers, four special lists are used. Map-Open-List and Map-Close-List are used to contain cells that have already been en-queued and de-queued respectively by the outermost BFS where as Frontier-Open-List and Frontier-Close-List contains cells that have already been en-queued and de-queued respectively by the frontier extraction BFS [3].

2.4 Fast Frontier Detector

The FFD method has a faster and much more efficient frontier detection strategy compared to the WFD algorithm. A novel approach

to the problem of frontier detection was presented by the FFD strategy [3]. The FFD exhibits high efficiency in detecting frontiers because it only processes raw sensors data obtained in real time. It does not search all the cells lying in the explored area or unexplored area. But in order to prevent re-detection of frontier cells and other inconsistencies, the FFD must perform maintenance of the frontier cells frequently. After every scanning operation, FFD must perform four operations namely; Sorting, Contour Generation, Frontier Detection and Frontier Maintenance.
In the sorting step, all the points obtained after each scan is ordered according to their location from the current position of the robot. The occupancy value Po for each cell is also stored in the same order. The complexity of this step could vary depending on the sensor used. As IR sensors are used as proximity sensors, a full round scan generates occupancy values (Po) for the adjacent cells in a linear order and hence no further processing is required for sorting. During the second step called contour generation, all these sorted points are connected and a contour or a boundary line is generated which represents the frontier regions generated so far.
In the next step, all new frontier cells are detected and identified by comparing every adjacent pair of cells lying in the contour generated in the previous step. During comparison of each pair, if the new current cell and the previous cell are both frontier cells, the current cell is incorporated into the same frontier region else a new frontier region is created and the current cell marks its beginning. The final step performs the most important task of maintaining all the frontier cells detected so far. The goal of the final step is to avoid detection of new frontier cells in the already scanned area and also to eliminate the cells which are no longer considered as frontier cells.
The work done in [3] only describes how the frontier cells are detected and maintained. No information is provided on how the best frontier cell is chosen among all the detected frontier regions. Hence we assume that frontier cells are selected in the same way as in the case of BGS and MGS algorithms. Once all the frontier cells are processed by the FFD; we choose a best frontier cell based on the heuristic cost value. Once such a frontier cell is chosen, the robot moves to the best frontier cell location and repeats these four steps until the target is reached. The pseudo code of the FFD algorithm is given below.

WHILE the goal is not reached
  Scan the cells in the current sensing region
  SORT the sensor readings
  GENERATE CONTOUR from the sensor readings
  EXTRACT new frontier cells from the generated contour
  MAINTAIN the new FC detected and remove the old FC
  Choose the best frontier cell based on the cost values and move to the location
END WHILE

Algorithm 2: Fast Frontier Detector algorithm pseudo code [3].

2.5 Incremental Wave Front Detector

The WFD-INC combines the features of both WFD and FFD algorithms. In this modified version of the WFD algorithm, instead of searching the whole map for all the frontier cells, it searches only those cells inside the current active area or sensing region [3]. If the orientation of the grid map does not change, then the location of all the frontier cells remain unchanged. The WFD-INC algorithms takes advantage of this fact by processing only the region which is under the current scanning area and thereby reduces the search space drastically [3]. Since this modified algorithm does not search the whole map; continuous maintenance of the frontier cells detected is highly essential. WFD-INC is supposed to perform better compared to the state-of-the-art frontier detection algorithms but when compared to FFD, it is still limited.
These are the major algorithms that utilize frontier based strategy for robotic navigation and it has already been studied and proved that the MGS algorithm performs better than the BGS in the paper titled Frontier Based Navigation for Multiple Robots [1]. Similarly it has also been found that the FFD performs better compared to the WFD, WFD-INC and FFD algorithms. But no comparison has been done between these two leading frontier based navigation techniques namely the MGS and the FFD algorithm. In this thesis; the two algorithms namely the MGS and FFD are implemented in real world for comparing their performance with the QGS algorithm.

3. PROPOSED SYSTEM

The proposed system consists of an autonomous robot loaded with the QGS algorithm in an environment as shown in Fig.1. The system senses the environment to detect the presence of obstacles with the help of the IR range sensors. The computer which runs the algorithm is connected and mounted onto the robot. The algorithm issues navigational instructions to the robot wheels based on the path planning strategy. The wall clock time taken by the robot to reach the destination is measured for various combinations of start and target cell locations. For the frontier based algorithms, the only information available to the robot is its current location and location of the its destination. The whole environment is considered as an occupancy grid [4] of size 10x10 where each cell is a square cell with a side of length 0.4m.
In the occupancy grid, initially every cell is considered as an unexplored cell and it is assigned the respective state value (cell[x][y].state). As the scanning function generates information about the cell scanned by the sensor, its state value is updated dynamically. A cell which is occupied by an obstacle is assigned a very high state value or Po. Frontier cells are assigned the least value where as free cells are assigned values in between frontier cells and unknown cells. Hence every cell in the 10x10 grid is represented with a user defined structure variable which contains data about its x-coordinate, y-coordinate and state value. As the robots explores the terrain, the state value is updated from that of an unknown cell to either a free cell, an obstacle filled cell or a frontier cell. Hence all the region which has been covered by the robots sensor is considered as explored region and the rest as unexplored region. Once a cell has been found free of obstacle, it is considered as a free cell and all those free cells which have at least one unexplored cell as its neighbour are considered as frontier cells (FC).
The cells that lie within the sensor sweep range of the last scanned region are known as cells in the current sensing region [2]. The QGS incorporates the same heuristic cost value known as the Goal Seeking Index [2] (GSI) used in the BGS algorithm explained in section 2.1. The GSI value is assigned to all the frontier cells depending on three factors i.e. the distance of the detected FC to the destination cell, distance of the FC from the current robot position and the state value of the FC. The cost of traversal to a cell occupied by an obstacle is very high where as the cost to travel to a free cell or a frontier cell is comparatively low. In the proposed QGS algorithm, similar to all other frontier based navigation algorithms, the strategy used is to maintain a list of all the frontier cells detected so far. Once a FC with maximum GSI is chosen from the sorted FC list, the algorithm issues instruction to move the robot to the chosen FC. The robot now scans the new location in order to identify bet-

ter frontier cells which are more closer to the target. This process repeats until the target is reached or until there are no more frontier cells left to be explored.

3.1 Features

Implementation and analysis of the MGS algorithm and the FFD algorithm exposed its flaws and drawbacks. As both these algorithms were performing a full round scan of the environment, most of the frontier cells detected were not explored at all. This was mainly because the frontier cells lying away from the target were assigned lower GSI value compared to the ones lying closer to the target. These frontier cells with very low GSI value gets pushed towards the bottom of the sorted list and rarely gets explored by the robot. This shows that a large portion of the total exploration time is being wasted on scanning cells which are rarely explored. In order to reduce this unwanted scanning, a new strategy was required which would optimize scanning in such a way that almost all the cells that are scanned, gets explored eventually. The QGS algorithm uses two intelligent approaches; aligning the robot frequently towards the target and developing a new scanning strategy. These two new features are explained in detail below.

3.1.1 Align Function. The main innovative feature of the QGS algorithm is the align function. This function is executed initially and also every time the robot moves or changes its orientation. The whole grid map is divided into four quadrants by assuming the current robot position as the origin. Once the quadrant where the target cell is located is identified, the align function changes the orientation of the robot in such a way that the target is located in the lower right quadrant. Once the robot is aligned in the best possible orientation, it sweeps and scans the adjacent three cells using its IR sensor. Hence the time wasted in order to perform an unwanted full round scan of the environment is extremely reduced. For example, if the robot is facing north and the target was found to be in the south east region of the current robot position, the robot is turned to face east and a clock wise scan is performed towards the south. Hence all the cells lying in the south east region, i.e. the cell to the east or right of the robot, to its south east diagonal and to the bottom, will be scanned and explored. All the cells lying outside the scanned quadrant are excluded, thereby reducing the search space and scan time to a great extent.

3.1.2 Scan Function. The modified scanning function as mentioned above consists of a single 90 degree sweep in a clock-wise direction. Depending on the orientation of the robot, the cell in front of the robot, the cell lying in its diagonal right side and the cell lying to right hand side of the robot will be scanned by the robots sensor. The new scanning function has reduced the search space and the total time required to scan the environment to one-fourth. During each scan, the obstacles detected are updated in the occupancy grid and the once frontier cells are detected, it is pushed to a list called the ”activeArea”. The frontier cells in the activeArea list are sorted in decreasing order of its goal seeking index or GSI. If the best FC detected inside the current sensing region is having a GSI value greater than the previously found best frontier cell, the robot is moved to the selected frontier cell. And if no frontier cells are found inside the current sensing region, the situation is called as a ”TRAP” [2] situation. Once a trap is detected, the robot must select a frontier cell outside the current sensing which is having the largest value of gsi. If no such frontier cells are available even outside the current sensing region, then the robot has failed to reach the destination. The pseudo code of the QGS algorithm is given below.

WHILE the goal is not reached
  Align the robots orientation to scan the best quadrant
  Scan the three adjascent cells in clock-wise direction
  Update grid map and if new FC detected, push to activeArea list
  IF FC are available in the activeArea
    Choose the FC with the largest GSI as the BEST FC
    Move the newly detected FC in activeArea to old FC list
  ELSE IF frontier cells are present outside the activeArea THEN
    Choose the FC with largest GSI from old FC list
    which is adjacent to an obstacle as the BEST FC
  ELSE
    All accessible area has been explored and robot failed to
    reach target. STOP.
  END IF
  Remove the cells from old FC list which are no longer FC
  Move robot to the BEST FC chosen
END WHILE

Algorithm 3: Quick Goal Seeking algorithm pseudo code

3.2 Problem Scenario

The problem scenario is represented in Fig.1. Here the starting location of the robot is set to the top left corner of the grid and is marked as ’S’. The target location is represented as ’T’ and as the robot moves, its current position is updated and represented as ’C’. Initially all the cells are considered unexplored and represented in light grey shade. As a new cell is scanned, it is considered as an explored cell and its state value is updated in the occupancy grid. Explored cells are shown in white and a cell which is occupied with an obstacle is shown in dark grey. The free cells which are having atleast one adjacent unexplored cell are considered as frontier cells and it is shown in light brown with a cross in the middle. The direction towards the right hand side of the origin is considered as positive X-axis and the direction towards the bottom of the origin is considered as the positive Y-axis. The functions named **addx, addy, subx and suby** gives command to the robot to move one grid cell either to the right, bottom, left or top respectively.
Initially, the only information available to the robot is its current location and the location of the target in the grid map. As the robot detects new frontier cells, it moves towards the best one among them in order to get more information about the terrain. The robot has to reach the target location as fast as possible and if obstacles are detected on route to the target, it must be avoided and updated on the map. The path covered by the robot is represented as dotted lines in the Fig.1. At the end of the program execution, a map is generated in a user understandable form. The map must display the location of the free cells, frontier cells (represented as ’*’) and obstacles detected (represented as ’O’). The goal of the robot is to reach the target location through the shortest path by avoiding the obstacles detected in real-time. The problem defined in this thesis is to implement a new frontier based navigation algorithm in a real-time environment, which is better than the two currently leading frontier based algorithms namely, MGS and FFD, and to reach the target location in the shortest possible time.

3.3 System Architecture

The system architecture is shown in the Fig.2. The input given to the robot is the location of target cell in the grid map. The localization block identifies the location of the robot and quadrant in which the target is located using the align function as explained in section 3.1.1. Once localization is completed, the robot is automatically aligned towards quadrant containing the target. The next block in

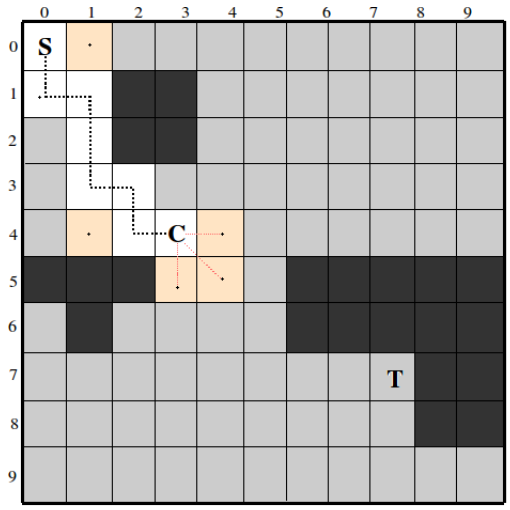

Fig. 1. Problem Scenario

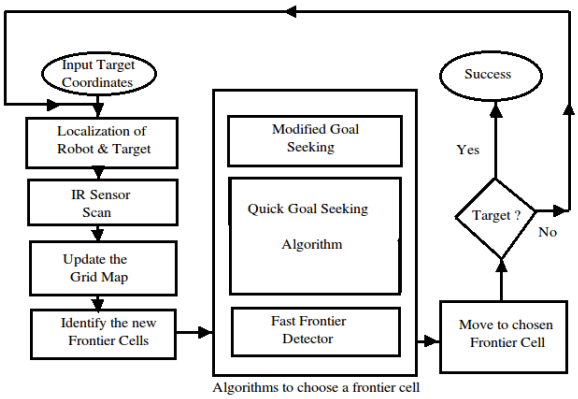

Fig. 2. System Architecture

the system architecture represents the scanning function. The IR sensors are used to scan the environment and identify the FC, free cells and the cells occupied with obstacles around the robot. The state values of the scanned cells are updated in the grid map. After the identification block, all the FC lying within and outside the current sensing region are identified and sorted in the decreasing order of its GSI. The middle block represents the three frontier based algorithms out of which one algorithms is used to choose the BEST FC. The BEST FC is chosen depending on the strategy proposed by the corresponding algorithm. The next block is used to issue commands to the actuators of the robot in order to move to the BEST FC chosen in the previous step. The algorithm keeps executing from the beginning until the robot reaches the target location, else it terminates.

3.4 Implementation

The robot chosen for the chassis is the iRobot Create and an IR range sensor is used to detect obstacles. Sharp IR sensor model GP2Y0A21YK is used as the range sensor and it is connected to an

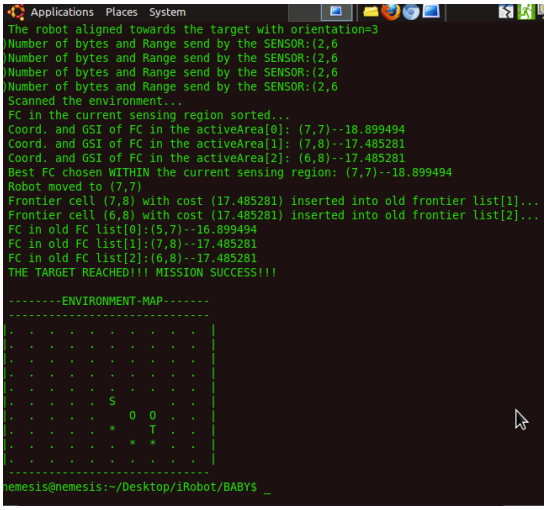

Fig. 3. Map Generated

analogue input pin of the Arduino UNO board. The arduino board contains program to calibrate, smooth-en and average the raw voltage values coming from the IR sensor to a distance value for better accuracy. This distance value can be accessed from the arduino board by the net book using a regular serial communication protocol. The iRobot and the arduino (along with the IR sensor) can be connected to a net book using a USB-Serial converter cable and normal USB cable respectively. Hence the robot used for navigation consists of an iRobot mounted with a net book, an arduino board and an IR range sensor.

Since we are concerned only about scanning the adjacent cells from the robots current position, we limit the IR sensors capability by reducing its detection range from (0-1.5m) to (0-0.4m). Hence during each scan, the arduino board returns a number which represents the presence or absence of an obstacle in the probed cell. Once an obstacle is detected by the robot, its location is saved and proceeds towards the goal by avoiding it.

Both the algorithms are implemented in C language and Linux based GCC compiler is used for compilations. Ubuntu 11.04 is used as the operating system and all the code and hardware peripherals used are open source. A special open source library called the COIL is included which simplifies the control of the iRobot through USB-serial port communication. The arduino 1.0.5 drivers are installed to interface the arduino serial communication. Once the target is reached, a map of the traversed environment is generated and displayed to the user in the same command window as shown in Fig.3.

3.5 Technical Details

—The operating system used is Linux Ubuntu 11.04.

—C Language is used for coding the main program and it is compiled in the normal gcc compiler available in Ubuntu 11.04

—The robot used is iRobot-Create (Fig.4). It has a three-wheel differential drive chassis with two driving wheels that contain incremental encoder and one supporting swing wheel. The netbook controls the iRobot by connecting it to its serial port through a USB-Serial converter cable.

—COIL is used as the external library to control the iRobot. COIL stands for Create Open Interface Library and is an open-source library which makes it much easier to start controlling the iRobot

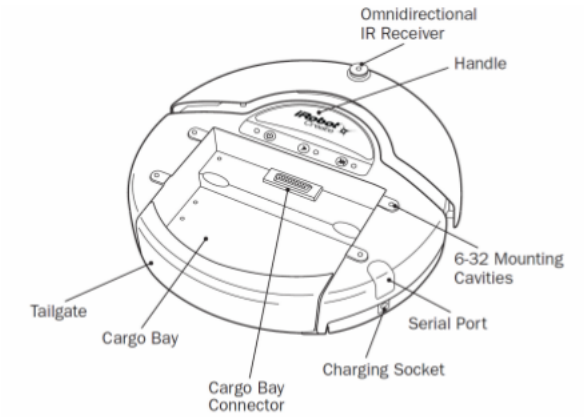

Fig. 4. iRobot

as well as reading data from it by using C functions. Small size of COIL library makes it easier to run on a net-book. But the drawback of using COIL is that, it does not work on Windows OS. Linux and Mac-OS are the only operating systems that support the use of COIL library.

—SHARP IR range finder-GP2D120 is used for obstacle detection.

—Arduino UNO is used to connect the Sharp IR sensor onto the computer. The IR sensor is connected to an analogue pin which sends voltage readings to the arduino board. But these voltage readings are raw values and needs to be processes for accuracy. The code required to calibrate and smooth-en the raw IR sensor readings are burned onto the arduino board.

4. RESULTS AND ANALYSIS

The frontier based navigation algorithms; QGS, MGS and FFD have been successfully implemented and tested in real-time environment. The robot explored ten map cases with obstacles placed at varying locations as shown in Table 1. All the three algorithms were executed for each map case. The wall clock time taken by the algorithms to complete the exploration is the parameter used to compare performance.

In order to compare the time complexities of the algorithms, a graph is plotted between the obstacle density of the environment and the time taken to explore it. Obstacle density gives a measure of how densely the obstacles are placed in the environment. In order to calculate the obstacle density we consider a rectangle with the starting cell located at one corner and the target cell at the diagonally opposite corner. Obstacle density is calculated by finding the ratio of number of cells occupied with obstacles to the total number of cells in the rectangle. The Table 2 contains data about obstacle density of the map case and the time taken in seconds by the algorithms to explore it. It is evident from the Table 2 that as the obstacle density increased, the time taken by the algorithms to explore the environment also increased proportionally. The graph plotted from the data in Table 2 is shown in Fig.5.

5. CONCLUSION AND FUTURE ENHANCEMENTS

A new frontier based navigation algorithm, quick goal seeking algorithms was successfully designed and implemented. Two leading frontier based navigation algorithms; modified goal seeking and fast frontier detector algorithm has also been implemented and tested under similar environments. It is clearly evident from the results that the QGS algorithms outperfomrs the remaining two algorithms with respect to the total time taken and the total number of cells traversed to reach the target node. The new innovative align function and scanning strategy has proved to be highly efficient and time saving.

The future works include improving the newly developed QGS frontier based navigation algorithm by incorporating high performance sensors like laser range finders better robot for indoor as well as outdoor navigation. The improved QGS algorithm would be able to navigate highly complex and challenging environments and hope to revolutionize the frontier based navigation strategy.

6. REFERENCES

[1] Yamauchi, Brian.: Frontier based exploration using multiple robots. In: Proceedings of the Second International Conference on Autonomous Agents, pp. 4753. Minneapolis, Minnesota (1998).

[2] Jisha, V.R., Ghose, D.: Frontier Based Goal Seeking for Robots in Unknown Environments. In Journal of Intelligent Robotic Systems, 0921-0296, Springer Netherlands (2012-09-01).

[3] Keidar, Matan, Eran Sadeh-Or, and Gal A. Kaminka.: Fast frontier detection for robot exploration. In Advanced Agent Technology, Springer Berlin Heidelberg (2012) 281-294.

[4] Elfes, A.: Using occupancy grids for mobile robot perception and navigation. IEEE Computation 22(6), 4657 (1989).

[5] Burgard, W., Moors, M., Stachniss, C., Schneider, F.E.: Coordinated multi-robot exploration. IEEE Transaction on Robotics. 21(3), 376386 (2005).

[6] Moravec, H., Elfes, A.: High resolution maps from wide angle sonar. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp.116121. St. Louis, MO, USA (1985).

[7] Murphy, R.R.: Introduction to AI Robotics. Prentice Hall, India (2005).

[8] Keidar, Matan and Gal A. Kaminka.: Efficient Frontier Detection for Robot Exploration. In: The International Journal of Robotics Research, Published on 22nd October (2013).

Table 1. Performance Analysis

Start	Target	Location of Obstacles	QGS Time	MGS Time	FFD Time
(5,5)	(7,7)	(5,6)-(6,6)	0:23	0:36	0:37
(5,5)	(7,7)	(6,6)-(7,6)	0:45	0:54	0:55
(5,5)	(7,7)	(6,6)-(7,6)-(8,6)	0:44	1:11	1:14
(5,5)	(7,8)	(6,6)-(6,7)-(6,8)	0:42	0:45	1:04
(5,8)	(7,9)	(6,8)-(6,9)	F	1:43	1:34
(4,4)	(6,6)	(5,5)-(6,5)-(7,5)-(5,6)-(7,6)	1:02	1:29	1:20
(4,4)	(6,6)	(5,5)-(5,6)-(5,7)-(6,5)-(7,5)	1:10	1:11	1:12
(4,4)	(9,6)	(5,5)-(6,5)-(7,5)-(8,5)-(9,5)	0:42	1:02	4:20
(4,4)	(7,8)	(6,5)-(6,6)-(6,7)	0:44	1:02	1:04
(7,7)	(4,4)	(4,5)-(5,5)-(6,5)-(7,5)	1:15	1:16	1:36

Table 2. Obstacle Density-Exploration Time Analysis

Map Case	Obstacle Density	QGS Time	MGS Time	FFD Time
1	0.22	23	36	37
2	0.22	45	54	55
3	0.33	44	71	74
4	0.25	42	45	64
5	0.22	F	103	94
6	0.41	62	89	80
7	0.31	70	71	72
8	0.27	42	62	260
9	0.15	44	62	64
10	0.25	75	76	96

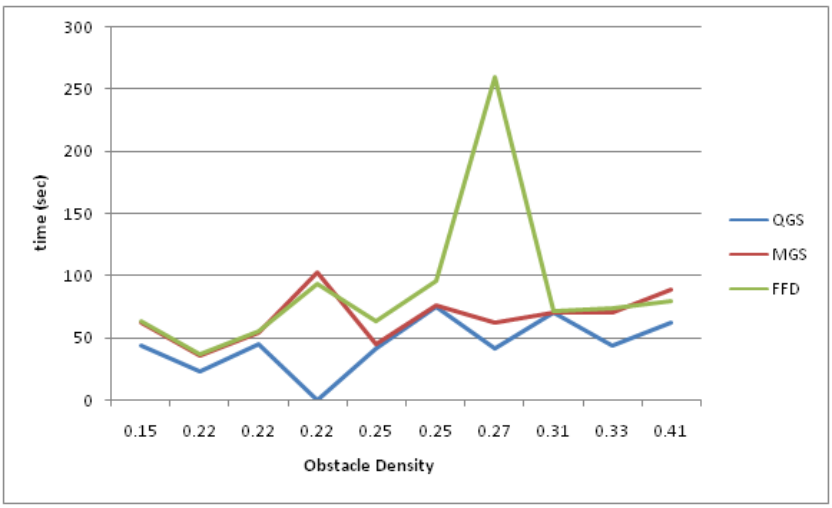

Fig. 5. Graph showing variation of exploration time with obstacle density for QGS, MGS and FFD algorithms