Obstacle Mapping with a LEGO Robot in an Unknown Environment

Jack Hennessey, Salil Vishnu Kapur and Parmeet Singh

***Abstract*— In this paper, we report the different techniques implemented for an EV3 Lego Robot for obstacle mapping in an unknown environment involving Path Planning, localization and sensor modeling.**

I. INTRODUCTION

Unexplored, enclosed environments represent a common challenge in autonomous robotics. Robotic staging areas such as mine shafts, collapsed buildings, or even a common home for an autonomous vacuum cleaner are all prime examples. The objective of the development of this robot was to create an autonomous wheeled mobile vehicle capable of entering and exiting an unknown enclosed environment, and producing a map of the space for an operator to interpret in post-processing. Obstacle mapping in robotics involves an amalgamation of three key modules, path planning, localization and accurate sensor modeling. Without all three of these elements obtaining reliable readings as a robot travels through an environment is impossible.

There exists a plethora of algorithms that may be applied to each individual element that will enable the successful mapping on an unknown environment such as SLAM, RANSAC, and the bug path planning methods. However, as the chosen implementation platform was the LEGO EV3 robot, which only has a 300MHz single-core processor and a limited 64MB of RAM, algorithms must be lightweight and optimized to run or maintain an acceptable execution time [1]. This creates a level of difficulty in that advanced, reliable algorithms such as SLAM and RANSAC may not be employed unless stripped down to their rawest elements.

II. METHODOLOGY

A. ALGORITHM

The design for achieving the final target was designed by keeping different functionalities under the different sub modules. For this Executor, Path Planning, Mapper, Mover, Utility, Danger Checker and sensor test modules were built. The Architecture is explained in detail in Fig 1. All these Modules were built by keeping their proper interaction among them and by confining to the object oriented principles :

- Executor module :- It makes call to the 7 phases traverse around a global coordinate axes.
- Path Planning :- This does the planning of the next move of robot to finally guide it to reach it's goal.
- Mapper :- This does the 360˚ view of the environment when being called by Path Planning to do so.
- Mover :- Used to move the robot in a straight line from one position to the other and maneuvers.
- Utility :- Used as a helper module with quite a few utilities which will be used by other modules.
- Danger Checker :- Notifies in advance in case the robot gets stuck in a dead end that it cannot maneuver out of.

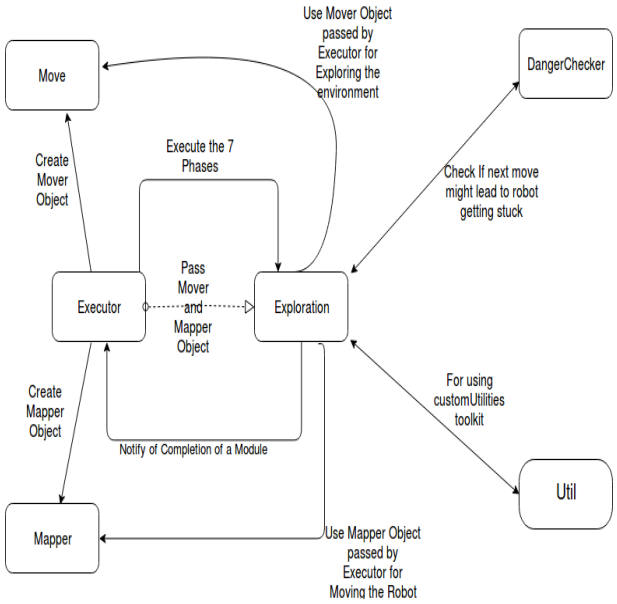

Figure 1: Code Architecture in the Robot Brain

The exploration is designed by keeping a global coordinate axes and by doing an approximately square exploration around the center of the global coordinate axes. This approach is based on a very general Idea where Unmanned Underwater vehicles are made to explore an environment by keeping a global coordinate axes in the center and making the UUV rotate around it. The robot Path planning is designed based on human vision, as a human has a vision angle of around 180˚ so though the robot takes a 360˚ view but uses the 180˚ view for further moving to it's target, which is termed the correct orientation view of the robot.

*Research supported by Dalhousie University.

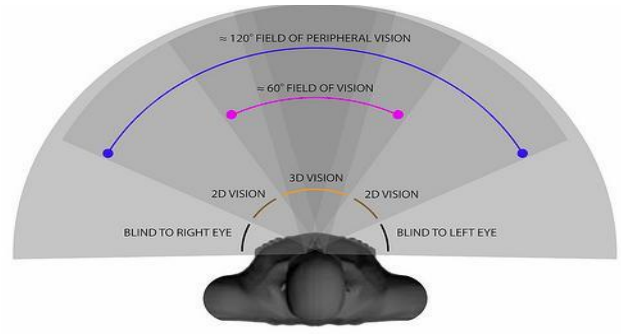

Figure 2: Depiction of human 180˚ view

B. ALGORITHM IMLEMENTATION

The execution of this project begins by running the Executor class, this creates Mapper and Mover objects and triggers the 7 phase exploration of the environment. The 7 phases include the starting point, point in the middle of the door, the four corners of the the playground and the terminal point which is same as the starting point. The first phase crosses the entrance of the gate and reach inside the playground. The second phase involves taking a right and reach the first corner of the playground. Then the robot takes a left and reaches the second corner which is the ending of the third phase. The fourth phase takes a left and reaches the third corner. The fifth phase involves taking a left and towards the fourth corner. The sixth phase involves taking a left and going towards the entrance of the gate. The seventh phase involves reaching the point from where it started. Every phase is completed if the robot reaches somewhere between 30 cm of a waypoint. The executor module does the exploration by calling the Exploration class along with passing Mover and Mapper object. Inside the Exploration controller is triggered which refreshes the mapping using the Mapper object and if there is no obstacle ahead then it heads for the goal directly otherwise it uses the tangent Bug algorithm to head further. And for making every small move, the mover object is used. This approach is followed for finishing up all the 7 phases.

Occupancy grid was used to understand the playground. The playground was mapped onto a 56 by 44 unit grid with 5cm resolution. Each point in the grid was associated with a positive number which indicated the confidence level of whether there is a an obstacle present at that particular point. The ultrasonic scans and activation of the bump sensor populates the values in the occupancy grid. Initially, the entire occupancy grid is initialized to zero except the walls of the playground. Since the dimension and position of the playground is already known, we initialize the coordinates corresponding to the playground walls as one. The prior knowledge of the walls can potentially compensate for the noise in the ultrasonic sensor scans.

We update the point of a received return from each ultrasonic scan in the corresponding point in the occupancy grid. Since the ultrasonic scan values are noisy, we update a patch of values with the point of return as the center. The point of return is assigned the highest value and the surrounding values are lesser which vary according to a Gaussian function. A Gaussian kernel is generated and added to the occupancy grid with the point of return as the center.

The robot might run into a short object that cannot be detected by the ultrasonic scans. In those cases, the bump sensor will be activated. When the bump sensor is initiated, a virtual obstacle is mapped onto the occupancy grid in the shape of a line whose length is equal to the shape of the robot bumper and orientation is perpendicular to the robot pose. Each point in the virtual obstacle is updated with Gaussian kernel i.e. the surrounding points are updated with the values that are less than the bump point.

The robot path planning uses the values in the occupancy grid for obstacle avoidance. The robot checks a rectangular portion of the grid directly in front of the robot's view. The robot perceives a clear path if a certain threshold of field-of-area is clear in front of it.

C. LOCALIZATION

The position of the robot was maintained by using wheel encoders for translational motion of the robot. The change in wheel encoder count gives the amount in degrees the wheel turns. The distance travelled by the robot can be calculated by multiplying the change in wheel encoder count with the wheel radius.

The turn motion is performed by turning both the wheels equally in opposite directions. The amount of encoder counts to turn the wheel is equal to the number of degrees to turn(one wheel rotation takes 360 tacho turns). We calculate the number of encoder counts to turn by equating the distance travelled by wheel which is theta_wheel*wheel_radius with theta_to_turn*(half_of_distance_between_wheels). Therefore number of encoder counts to turn is theta_to_turn*(half_of_distance_between_wheels)/wheel_radius.

The orientation of the robot was updated using a Kalman filter for the turning of the robot. The gyroscope readings were used as feedback for the finding the change of orientation of the robot with every turn command given to the Robot.

D. PATH PLANNING

The overall idea is to visit the set waypoints or reference points, map the occupancy grid and return to the starting point. The four corners of the playground are included in the set of waypoints that the robot covers. The tangent bug algorithm[3] is used to avoid obstacles in the course. The algorithm uses ultrasonic scans to detect discontinuities in the 180 degrees field of view of the robot. If the robot detects a discontinuity it heads towards the point that is a set distance before the discontinuity. In addition to the ultrasonic scan, the robot performs extra checks to make sure the path ahead of the robot is clear. It creates a rectangle region in front of the robot. This check is described in Fig 3. The occupancy grid is used to sum up values of the coordinates that fall in the region of the rectangle. If the sum from the rectangle is lesser than a threshold, the robot assumes a clear path for itself.

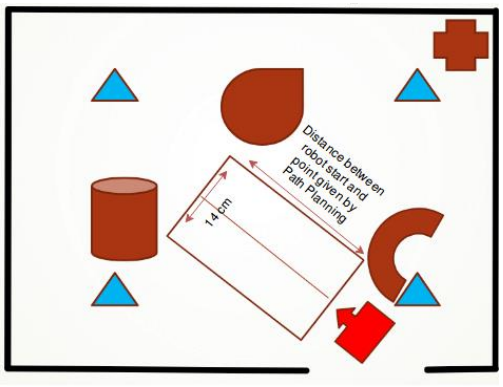

Figure 3: A rectangular field view to check a clear path for the robot

E. DEAD END CONDITIONS

The robot tries to reach to each of the waypoints or reference points before exiting the gate and reaching the terminal state. A condition may arise where an obstacle blocks the robot at the corner when it tries to reach the waypoint. It might not be possible to go around the obstacle to get to the reference points. To avoid these conditions we assess the ultrasonic scans for the robot. If the ultrasonic scans indicate that the opening is narrow the robot avoids going towards that reference point and heads for the next waypoint. In general, the errors in localization accumulate with greater number of moves initiated by the robot. We limited the number of moves for each waypoint to 5. Therefore, if the robot is not able to reach a waypoint in five moves, the robot motion is initiated towards the next waypoint. This is done to maximize the robot chances of reaching its terminal state by exiting the door.

III. EXPERIMENTAL HARDWARE

A. ULTRASONIC SENSOR CHARACTERIZATION

Due to the mapping nature of this objective, an ultrasonic range sensor was employed by mounting it through a gear train to a servomotor so that it could be rotated through a full 360˚ sweep, recording range measurements at discrete angle increments. This setup produces an array of polar coordinates from the center of the robot, that may be converted to global coordinates with knowledge of the pose of the robot. To enable a probabilistic approach to path planning, sensor characterization was employed to apply a confidence interval to ultrasonic measurements. Fig 4 displays a hit-map of ultrasonic readings from the robot in a square room with random-shape objects on the left and top(viewed from above), and an empty room in the remaining areas. It can be clearly seen the sensor does not return a straight line despite being swept along a wall. This is due to the finite cone angle of the ultrasonic beam, only giving returns from the closest object within the cone as outlined in [2]. From this result, the cone angle can be interpolated to be 32.8˚ ± 9.6˚, which was used to apply a Gaussian distribution about point readings from the sensor.

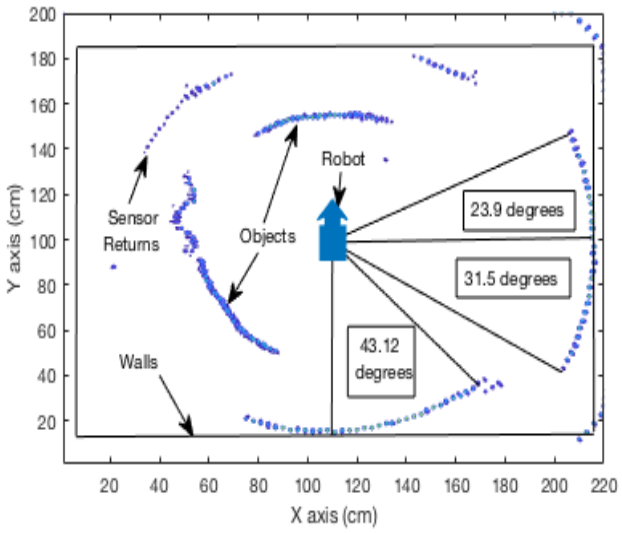

Figure 4: Derivation of cone angle in a square room

B. ROBOT CONSTRUCTION

To maneuver and map in a two dimensional space with unknown obstacles, it is necessary to construct a robot platform that will not hinder the operation of the three elements required for the objective. The following assumptions were made about the algorithms, and the effect these assumptions made are also listed.:

- Path planning algorithms will operate in the bug condition.
    - Robot must have smallest footprint possible.
- Some sensors rely on Cartesian localization to operate.
    - These sensors must be placed over the robot's rotation center.
- Stiffness of the robot components is paramount to achieve accuracy with any measurement.
    - Ultrasonic sensor mated to its servomotor through a worm-gear drive system.

IV. RESULTS

The EV3 lego robot was made to explore the environment, which in our case was 2.07mt and 1.75mt. The results that we got were quite interesting. The robot was able to move around the complete playground and come to the exit gate from where it started. The results are saved as a occumancy_grid.csv file which is visualized to understand the environment which the robot was left to explore. The results of the experiment are shown in the Fig 5 below where the yellow patches show the confidence level of the obstacle.

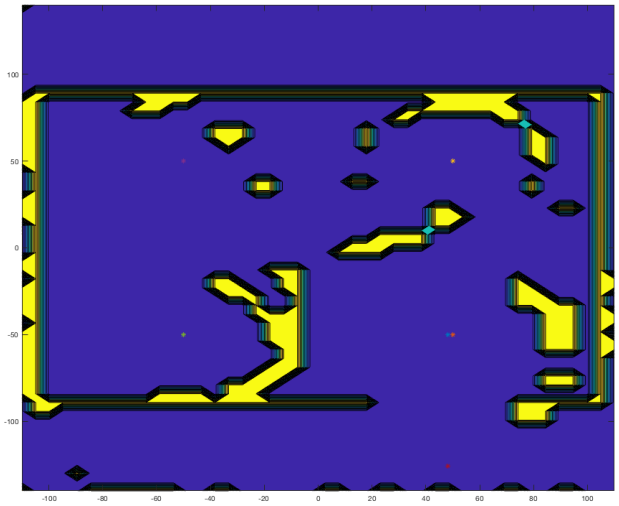

Figure 5: Occupancy Grid after a scan of the entire playground. Blue color indicates empty space and yellow and black indicate regions of obstacles.

IV. CONCLUSION

The problem of obstacle mapping in an unknown environment can be solved. It will, however, take time, money and a combined effort to find the best accuracy of mapping in a tough terrain with complex obstacles. With the experiment that was done there was considerable amount of accuracy in correctly mapping the environment but with considering the limitation of our experimental playground there still needs enormous ways to test our approach using LEGO EV3 robot.

V. FUTURE WORK

For our present experiment we used Kalman filter for rotational angle updating, for our future work we plan to use SLAM with landmarks for transitional motion. Also we plan to improve our path planning by using Bug 2 algorithm by comparing it with Tangent Bug Algorithm. For occupancy grid updation we see enormous scope of using Bayesian updating to improve the occupancy mapping accuracy.

VI. REFERENCES

[1] LEGO EV3 Robot User Guide [Online]. Available: https://www.lego.com/r/www/r/mindstorms/-/media/franchises/mindstorms%202014/downloads/user%20guides/lmsuser%20guide%20lego%20mindstorms%20ev3%2011%20tablet%20enus.pdf?l.r2=-830220782

[2] J. Crowley, “Navigation for an Intelligent Mobile Robot”, *IEEE Journal on Robotics and Automation*, vol: 1, no: 1, pp31-41, 1985.

[3] Kamon, I., Rimon, E., & Rivlin, E. (1998). Tangentbug: A range-sensor-based navigation algorithm. *The International Journal of Robotics Research*, *17*(9), 934-953.