Localization and Goal Directed Navigation in an Unknown Environment

Swee Balachandran
Aerospace Engineering

Christopher Lesch
Electrical Engineering and Computer Science

Michael Quann
Mechanical Engineering

***Abstract*—In this paper, an integrated system is developed that enables a robot to map its environment and autonomously navigate from a start location to a goal location with no a priori information about its environment. This problem consists of three fundamental steps. First, the robot learns its local environment and localizes itself within this environment. Second, with knowledge of its environment, it plans a path to its goal location. Finally, the robot navigates the planned path autonomously. These three steps are repeated until the robot reaches its final destination. The integrated system was successfully implemented on the Maebot platform and results of the implementation are discussed in this paper.**

***Keywords*—Scan Matching, Occupancy Grid, D* Lite, PID control.**

I. Introduction

A common problem for mobile robots is to navigate from point A to point B. With a complete map of the environment, one can plan an optimal route between the two points and the robot can follow this path to get from from point A to point B. However, on many occasions, a complete map of the environment may not be available to plan a detailed path. Furthermore, obstacles may be dynamic and consequently, the map can be constantly changing. In such cases, the robot needs to plan a path based on whatever local information it perceives about its environment. In addition, it may also need to re-plan the path as it receives new information about the environment.

The focus in this paper is on developing an integrated system that would enable a robot to autonomously navigate from a start location to a goal location with no prior knowledge of the map. This problem is decomposed into three sub-components. First, the robot simultaneously localizes itself and maps the environment. Second, with knowledge of the local environment, the best possible path is planned to the goal destination. Finally, the robot navigates the planned path. As it explores the unknown parts of the environment, the map is updated and if necessary, the path is re-planned. This process is repeated until the robot reaches the destination. ~~Fig~~ 1 illustrates the architecture used in this paper that accomplishes the above specified task. The integrated system developed in this paper was implemented and tested on The Miniature APRIL Education roBot (MAEBot) platform. The MAEBot is a small mobile platform with a differential drive train equipped with a suite ~~a~~ sensors consisting of a LIDAR scanner, a USB machine vision camera, ~~9 Axis~~ IMU, IR Range finder and wheel encoders.

This paper is organized as follows. In Section II, the related literature is reviewed. In Section III, the approach to this problem is outlined and each component of the approach is discussed in detail. Section IV provides experimental results obtained on the Maebot platform. Section V discusses implementation issues. Section VI provides conclusions and a discussion of potential future work.

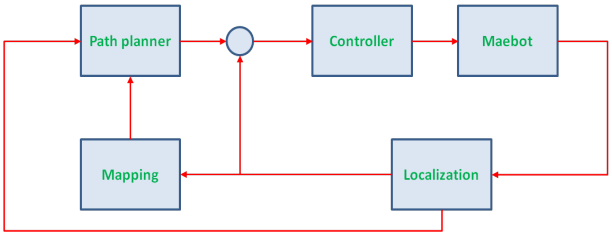


Fig. 1: System architecture

II. Related Work

A plethora of literature is dedicated to path planning for mobile robots. There are several variants to this problem. In the simplest case, it is assumed that a map of the environment, location of obstacles and free space is known a priori. In such ~~a case~~, path planning is typically done offline using optimal control techniques, potential field methods or search based techniques (see [1], [2], [3], [4], [5] and references there in). In several cases, due to uncertainties such as unknown dynamics, unknown obstacles and more, it is often required to re-plan the path and this has lead to research on real time path planning [6], [7], [8]. However, complete knowledge of the environment with the exact location of the obstacles may be hard to come by and this has spawned research on real time path planning with dynamically changing environments [9], [10], [11].

Several researchers have addressed the problem of navigating a robot between two given points in an unknown terrain. Beeson et al. [12] used occupancy grids in combination with a Voronoi graph search to navigate the robot in an unknown environment. Arana-Daniel et al. [13] used an EKF-SLAM to estimate a ~~feature based~~ environment and used a reinforcement learning algorithm to learn the optimal path between two given points. Huang et ~~al~~ [14] used a FastSLAM approach to localize and map the environment and used an RRT search algorithm to compute the path between two locations.

The approach in this paper is very similar to the work done by Errson et al. [15]. Ersson et al. focused on developing a planning and control strategy for navigation in an unknown terrain using a variant of the D* search algorithm. However, they assume that they have full knowledge of the pose of the

robot prior to planning and navigation. In this work, we present an integrated system where mapping, path planning and control are combined to enable an efficient solution to localization and goal directed navigation in an unknown terrain.

III. Approach

The development of the integrated system that enables the robot to map the environment and navigate between two locations can be broken down into three components. First, the robot needs to localize itself in the environment and construct a map of its local environment. The MAEBot is equipped with wheel encoders that provide odometry information from which the pose of the robot can be estimated. However, due to factors such as wheel slip and surface irregularities, odometry information alone doesn't provide reliable estimates of the pose. In this work, to localize the robot, we made use of the MAEBot's LIDAR scanner and exploited a scan registration algorithm to obtain reliable pose estimates. Next, a local map of the environment is constructed using occupancy grids. The occupancy grid enables convenient representation of free space and obstacles, facilitating the use of a grid based search algorithm to find the best path from the current location to the goal location. For this work, the path is planned using the D* lite search algorithm. Once a feasible path is obtained, the path is discretized into waypoints ~~which~~ are then used for guiding the robot along this path. A low level PID controller takes these waypoints and controls the robot to the next waypoint. As the robot moves and explores the environment, the map is updated. Additionally, as new obstacles are detected, the path is re-planned (if necessary). This process is repeated until the robot reaches its final destination.

A. Scan Registration

Scan registration or scan matching is the process of finding the rigid body transformation that aligns two point clouds obtained from two successive laser scans [16]. An Iterative Closest Point (ICP) algorithm [17], [16] is used to perform the scan matching. The ICP algorithm takes as inputs two sets of point clouds typically obtained from successive laser scans. First, the correspondences between the two sets of points is established on a nearest neighbor basis. Once the correspondences between the two point clouds are established, a rigid body transformation that best aligns the two point clouds is found. The ICP algorithm is outlined in Figure 2.

The values of $\Delta x, \Delta y, \Delta\phi$ that minimize the expression on Line 11 in Fig 2 was obtained in closed form as described by Martinez et ~~al~~ [18]. The algorithm is initialized with estimates of $\Delta x, \Delta y, \Delta\phi$ obtained from the odometry readings. It was found ~~that~~ poor choice of initial conditions for the algorithm often resulted in convergence to a local minimum. However, when initialized with odometry estimates, the algorithm yielded reasonable pose estimates. The average point cloud obtained from the ~~MAEBots~~ LIDAR scanner consisted of 298 points. Thus, in this work, the ICP algorithm operates on the 298 points. The parameter d_{max} ~~to~~ establish the known correspondences on line 5 in Fig 2 was chosen after several trials of experimentation. A value of $d_{max} = 0.1m$ yielded reasonable results. Fig 3 illustrates sample point clouds obtained from the ~~MAEBots~~ LIDAR scanner as the robot moved around.

Input : Point clouds A=$\{a_i\}$, B=$\{b_i\}$,
Initial transformation $T_0 = [\Delta x, \Delta y, \Delta\phi]$
Output: Transformation that aligns A and B

1: $T \leftarrow T_0$
2: **while** not converged **do**
3: **for** i=1 to N **do**
4: $m_i \leftarrow FindClosestPointinA(T.b_i)$
5: **if** $||m_i - T.b_i|| \leq d_{max}$ **then**
6: $w_i \leftarrow 1$
7: **else**
8: $w_i \leftarrow 0$
9: **end if**
10: **end for**
11: $T = \arg\min_T \sum_i w_i ||m_i - T.b_i||^2$
12: **end while**

Fig. 2: ICP Algorithm

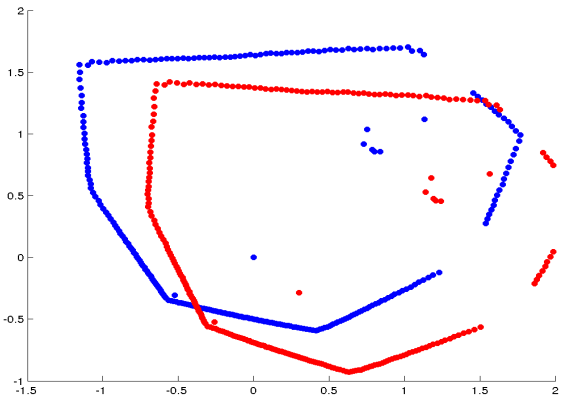


Fig. 3: Point clouds obtained from the MAEBot's LIDAR scanner (units in meters)

B. Occupancy Grid Mapping

With reasonable pose estimates obtained from the scan matching process, it was convenient to construct an occupancy grid map based on the known poses. In this work, we used an occupancy grid algorithm [19]. The algorithm is outlined in ~~Figure~~ 4. We used an inverse range sensor model as described in [20] to update the ~~logodds~~ value of each cell. The details of the occupancy grid map implemented on the MAEBot are outlined in Table I. We note here that a cell width of 5 cm resulted in a map update time that was appropriate for the real time application on the MAEBot.

~~Fig~~ 5 illustrates a simple environment created for the MAEBot. ~~Fig~~ 6 illustrates the map obtained from the scan matching pose estimates and occupancy grid mapping algorithm. The

TABLE I: Map specification

Cell size	5cm
Map dimensions	100 cell × 100 cells
$p(occupied)$	0.98
$p(free)$	0.20

procedure OCCUPANCYGRIDMAPPING($\{l_{t-1,i}\}, x_t, z_t$)
  for all cells m_i **do**
    if m_i *in perceptual field of* z_t **then**
      $l_{t,i} = l_{t-1,i} + inverse\_sensor\_model(m_i, x_t, z_t) - l_0$
    else
      $l_{t,i} = l_{t-1,i}$
    end if
  end for
  return $l_{t,i}$
end procedure

Fig. 4: Occupancy Grid Mapping algorithm

Fig. 5: Environment constructed for the MAEBot

blue robot and blue trajectory is obtained from scan matching while the cyan robot and cyan trajectory indicate the open loop pose estimates based on odometry. It was observed that the pose estimates obtained from the scan matching were consistent with the ground truth.

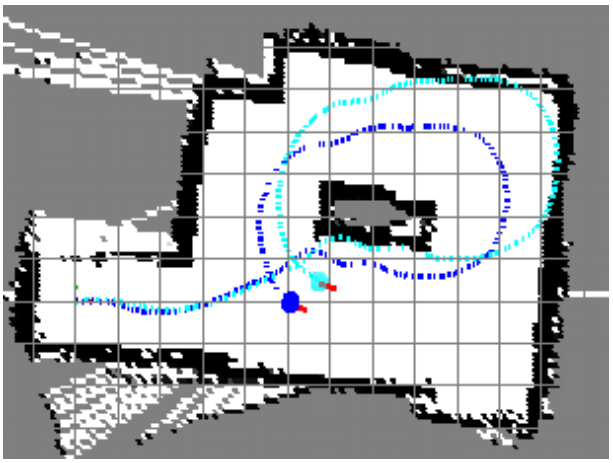


Fig. 6: Occupancy grid map of the maze environment

C. Path Planning

In the first step of the path planning algorithm the occupancy grid undergoes a conversion to a ~~cost based~~ map. Through this process we convert the probabilities of occupancy into a trinary state:

$$cost = \begin{cases} 2, & \text{if} P(occupied) <= 0.5 \\ wall/2, & \text{if} P(occupied) <= 0.8 \\ wall, & \text{otherwise} \end{cases}$$

where $wall$ is a fixed constant larger than the maximum valid path cost. This step to discretize the space is taken so as to help reduce the ~~affects~~ of noise in the occupancy grid results. We then take this cleaned result and apply a dilation of 2 cells in width to all walls. By increasing the cost of grid cells near the walls, but still allowing them to be traversable, the minimum cost path will prefer to stay 2 cells from walls, insofar as possible.

In the next step the optimal path from the robot's position to the goal position is determined by implementing a grid based path planning algorithm. Our work focuses on a derivative of the A* algorithm called D* Lite, developed by Koenig and Likhachev [9]. The traditional A* algorithm is designed for navigation in known terrain, where the locations of obstacles are known a priori and frequently even assumed to be constant. In our case, however, we are interested in repeatedly planning a path as the robot moves along it and discovers new terrain along the way. While A* is still completely valid for such a purpose, its use would require completely replanning the path anytime a map update is received. Planning the full path is an expensive process and can become intractable for real time applications as the map size grows. D* Lite is an incremental path planning approach that attempts to minimize the number of nodes expanded when the occupancy grid probabilities change.

The D* Lite algorithm, as implemented by [9], is reproduced in the Appendix in Figure 11, and an overview follows. The algorithm works by maintaining two estimates of the requisite path distance $rhs(s)$ and $g(s)$. Where:

$$rhs(s) = \begin{cases} 0 & \text{if} s = s_{goal} \\ \min_{s' \in Pred(s)}(g(s') + c(s', s)) & \text{otherwise} \end{cases}$$

and is therefore more informed about the minimum cost. The values of $g(s)$ and $rhs(s)$ start at infinity, with the exception that $rhs(s_{goal}) = 0$ as above. This is carried out in the procedure $Initialize$. Before proceeding we must define the term consistent to imply $g(s) = rhs(s)$ and inconsistent to imply the opposite. Then, for all inconsistent nodes we calculate a key following the procedure $CalcKeys$, and add those nodes to a priority queue, U, sorted in increasing order of keys. The priority queue determines the order of node expansion in the procedure $ComputeShortestPath$, and is initialized with the only inconsistent node, s_{goal}. $ComputeShortestPath$ then continues expanding nodes until s_{start} becomes consistent and the key of the next node to expand is larger than that of s_{start}. Note here that we had to invert the direction of the traditional path problem, instead of planning from the ~~robots~~ location to the goal we plan from the goal to the ~~robots~~ location. This step allows us to preserve our estimates of many path costs between map updates, but also requires that the goal

location be stationary. When a map update is received, any nodes to which edge costs have changed must be updated by a call to the $UpdateVertex$ procedure. This re-organizes the priority queue in an optimal manner and minimizes the number of nodes that will be expanded when we next call the $ComputeShortestPath$ procedure.

D. Guidance and Control

The estimates of the robot's pose from the scan matching algorithm, $\mathbf{x}_R = [x \quad y \quad \theta]^T$, along with waypoints, (x_{wp}, y_{wp}), given by the path planning algorithm, provide the necessary information to control the robot's motion. The MAEBot is a differential drive robot, which can be modeled simply as a unicycle such that:

$$\dot{x} = v\cos\theta \tag{1}$$
$$\dot{y} = v\sin\theta \tag{2}$$
$$\dot{\theta} = \omega \tag{3}$$

where control inputs are calculated that determine the linear speeds of each wheel, v_r and v_l, affecting the motion by

$$v = \frac{1}{2}(v_r + v_l) \tag{4}$$
$$\omega = \frac{1}{L}(v_r - v_l) \tag{5}$$
$$\tag{6}$$

in which L is the distance between the left and right wheels.

A PID controller is used to calculate motor inputs to allow the robot to follow the given path. The waypoints of the path and the robot's pose are used to find a desired heading, θ_d, where

$$\theta_d = \tan^{-1}\frac{y_{wp} - y}{x_{wp} - x} \tag{7}$$

With this, the error between the robot's current heading and the desired heading is given by:

$$e_\theta = \theta_d - \theta \tag{8}$$

The forward velocity, v, was set to be constant, while the angular velocity, ω, was determined by PID control of the e_θ term.

E. Software Architecture

~~Fig~~ 7 illustrates the software architecture that was implemented on the Maebot. In this work we make use of the Lightweight Communication and Marshalling (LCM) library [21] for inter-process communication. ~~Low-level~~ drivers communicate with the various sensors. ~~High-level~~ functionality is achieved by three separate processes. Process 1 (see Fig 7) estimates the current pose using the ICP algorithm. Process 2 estimates the pose of the robot making use of the odometry information obtained from the wheel encoders. Process 3 consists of two threads. Thread 1 runs the occupancy grid mapping and the path planning algorithm. When enabled, thread 1 also renders a VX ~~gui~~ for real-time visualization. Thread 2 runs the PID controller. The list of LCM channels used in our software is listed in Fig 7.

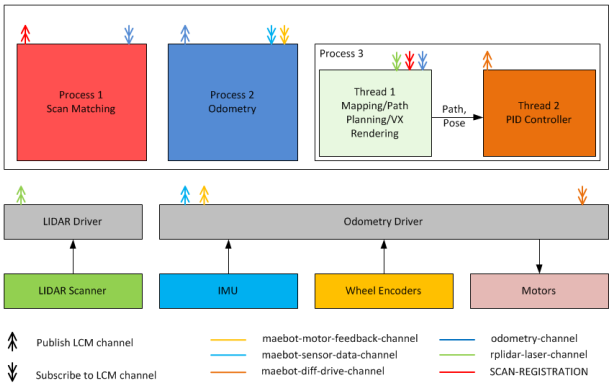


Fig. 7: Software architecture

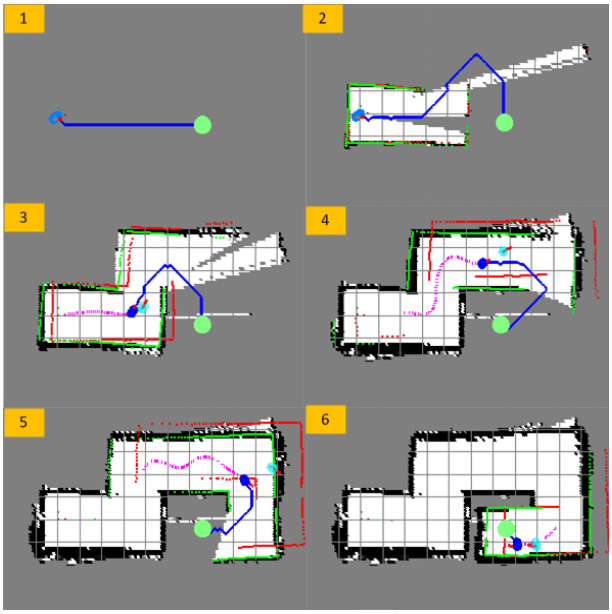


Fig. 8: Snapshots of the robot exploring an environment to get to the goal location

IV. Integrated Results

~~Fig~~ 8 illustrates several snapshots of the integrated system functioning. The blue robot in ~~Fig~~ 8 denotes the pose estimated by scan matching. The cyan robot denotes the pose estimated from odometry information. The green circle denotes the goal location. Initially, when the robot ~~doesn't~~ know anything about the environment, the optimal path to the goal location is a straight line path. This is because we assume that the unexplored region of the environment is traversable. It is evident that as the robot explores and learns more about the environment, the path changes to ensure that the robot is on a collision free trajectory.

The performance of the D* Lite algorithm was compared to that of the traditional heuristic A* search. In Table II we consider the number of times that the procedure updateVertex is called on average and in total during the trial run shown in ~~Figure~~ 8. The grid size for this trial was set to 100x100 cells. We have chosen such a node based comparison as opposed to a

TABLE II: Path planning performance statistics, showing the mean and total number of nodes expanded for the two different algorithms on the test case shown in ~~Figure~~ 8

Algorithm	Mean	Total
A*	6559	2944853
D* Lite	119	42817

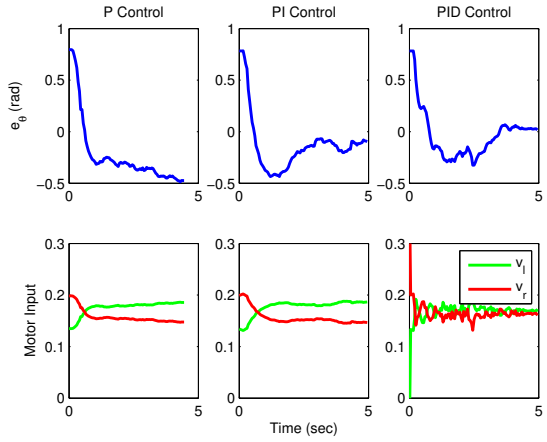


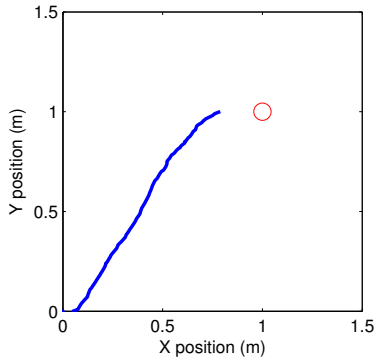
Fig. 9: Heading error and control inputs from $\mathbf{x}_{R,0}$ to goal at $(1, 1)$ meters

run-time based comparison in order to represent performance statistics in a platform agnostic manner. Note that this type of comparison is also agnostic with respect to the particular data structures used and therefore remains valid even as more efficient structures are developed. In accordance with Table II D* expands far fewer nodes than A* by leveraging both the heuristic function and maintaining state information between map updates.

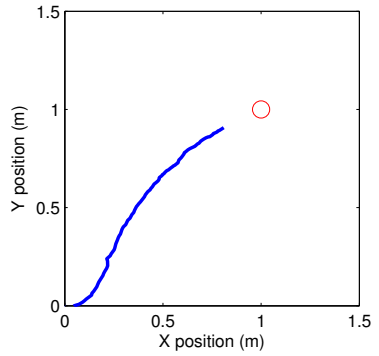
Testing was performed to determine appropriate gains for the PID controller. In Figures 9 and 10, the MAEBot was told to go from its initial pose $\mathbf{x}_{R,0} = [0 \quad 0 \quad 0]^T$ to a waypoint goal set at $(1, 1)$ meters. Strictly proportional control resulted in steady state errors that could not be overcome, in part due to the motor imbalance. This problem was fixed with the addition of an integrator term, which successfully drove the heading back toward the direction of the goal. The derivative term successfully made the response time faster, as exemplified by ~~Figure~~ 10, though at the cost of more erratic control inputs (~~Figure~~ 9). Further work could improve this behavior by incorporating a digital filter to reduce noise.

V. Implementation Issues

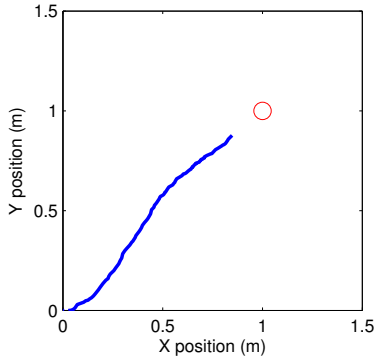
The highly agile nature of the MAEBot posed several challenges when integrating the various components of the system. Rapid changes in heading often resulted in distorting the laser point clouds. This in turn resulted in degrading the accuracy of the scan matching and consequently led to the generation of erroneous maps. To overcome this difficulty we included logic statements that would pause the scan matching and mapping processes temporarily when the robot experienced very large yaw rates.



(a) P Control



(b) PI Control



(c) PID Control

Fig. 10: MAEBot path from $\mathbf{x}_{R,0}$ to goal at $(1, 1)$ meters

As discused in section III-C the walls in our map are dilated outward to penalize more heavily paths that would take the robot closer to a wall. This gave us the benefit of a greater margin of error in PID control and mimicks the human tendancy to navigate the middle of a pathway. However, in the presence of dilation the change in cost of a single path edge now influences the cost of several neighboring edges. Consequently, this has the ~~affect~~ of increasing the number of nodes that require expansion in the D* Lite algorithm. If we consider the case in the limit, where a wall being observed affects the cost of all other cells in the map, we effectively negate any performance gain that D* Lite could give us. It is therefore desirable to limit the dilation to a minimum safety

distance. For the MAEBot this was found to be 10cm.

Implementation of the controller on the MAEBot proved to be a process of trial and error, and was hindered by motor imbalance and inconsistent robot performance between testing days. The controller itself performed well once these factors were accounted for. Additionally, the path planning algorithm gives numerous way points. Telling the MAEBot to head toward a waypoint too close to it caused unpredictable performance since any small error in pose or way point location could result in a large heading error. On the other hand, if the MAEBot looks at a way point far down the path, it could end up running into a wall. An appropriate balance between these two issues was met during testing.

VI. Conclusion

In this paper, we successfully developed an integrated system that enables a robot to localize itself, build a map of the environment while autonomously navigating from a start location to a goal location with no prior information of the environment. Testing and simulations showed how this approach can be effectively applied to a real physical system.

In this work, we only considered maps of fixed sizes. In other words, we defined an area in which the robot would operate. An interesting future work direction would be to analyze the scalability of our integrated system with respect to planning and navigation in large environments. Our current implementation does path planning over a grid ~~world~~ which often results in non-smooth ~~path~~ and also assumes the robot always moves forward. If the robot encounters a dead end during exploration, our implementation does not enable the robot to back up efficiently. Thus another interesting aspect to pursue ~~in~~ future would be to replace the grid based search with a search that takes into account the different motions of the robot: straight line motion, turning at different angles, reverse motion and more to plan an efficient and smooth path to the goal.

Though implementing individual algorithms and verifying them to work correctly is easy, integrating several algorithms to perform a complex task requires consideration of several issues such as communication between various algorithms, time delays, process synchronization, real time performance on hardware, memory limitations etc. These issues were important lessons from this project.

Appendix

- A video of scan matching and occupancy grid mapping working (Fig 6) can be seen at: https://drive.google.com/file/d/0B_LBVL8rrhC7cWJaMHNxSmVoWDA/view?usp=sharing
- Videos of integrated results (Fig 8) can be seen at: https://drive.google.com/file/d/0B_LBVL8rrhC7aGRIaGdkTnQyY1U/view?usp=sharing https://drive.google.com/file/d/0B_LBVL8rrhC7aTRxVVkyazQyTkk/view?usp=sharing

procedure CalcKey(s)
return
$[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))]$;
end procedure
procedure Initialize
$U = \emptyset$; $k_m = 0$;
  for all $s \in S$ **do** $rhs(s) = g(s) = \inf$
  end for
$rhs(s_{goal}) = 0$;
$U.Insert(s_{goal}, [h(s_{start}, s_{goal}); 0])$;
end procedure
procedure UpdateVertex(u)
  if $g(u) \neq rhs(u)$ AND $u \in U$) **then**
    $U.Update(u, CalcKey(u))$;
  else if $g(u) \neq rhs(u)$ AND $u \notin U$ **then**
    $U.Insert(u, CalcKey(u))$;
  else if $g(u) = rhs(u)$ AND $u \in U$ **then**
    $U.Remove(u)$;
  end if
end procedure
procedure ComputeShortestPath
  while $U.TopKey() < CalcKey(s_{start})$ OR
      $rhs(s_{start} > g(s_{start})$ **do**
    $u = U.Top()$; $k_{old} = U.TopKey()$;
    $k_{new} = CalcKey(u)$;
    if $k_{old} < k_{new}$ **then**
      $U.Update(u, k_{new})$;
    else if $g(u) > rhs(u)$ **then**
      $g(u) = rhs(u)$;
      $U.Remove(u)$;
      for all $s \in Pred(u)$ **do**
        $rhs(s) = \min(rhs(s), c(s, u) + g(u))$;
        $UpdateVertex(s)$;
      end for
    else
      $g_{old} = g(u)$;
      $g(u) = \inf$;
      for all $s \in Pred(u) \cup \{u\}$ **do**
        if $rhs(s) = c(s, u) + g_{old}$ **then**
          if $s \neq s_{goal}$ **then**
            $rhs(s) = \min_{s' \in Succ(s)}(c(s, s') + g(s'))$
          end if
          $UpdateVertex(s)$;
        end if
      end for
    end if
  end while
end procedure

Fig. 11: D* Lite

REFERENCES

[1] C. Martin, S. Sun, and M. Egerstedt, "Optimal control, statistics and path planning," *Mathematical and Computer Modelling*, vol. 33, no. 13, pp. 237 – 253, 2001. Computation and control {VI} proceedings of the sixth Bozeman conference.

[2] Y. Hwang and N. Ahuja, "A potential field approach to path planning," *Robotics and Automation, IEEE Transactions on*, vol. 8, pp. 23–32, Feb 1992.

[3] T. G. McGee and J. K. Hedrick, "Optimal path planning with a kinematic airplane model," *Journal of guidance, control, and dynamics*, vol. 30, no. 2, pp. 629–633, 2007.

[4] H. Chitsaz and S. LaValle, "Time-optimal paths for a dubins airplane," in *Decision and Control, 2007 46th IEEE Conference on*, pp. 2379–2384, Dec 2007.

[5] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," in *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, pp. 9–18, 2005.

[6] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," in *American Control Conference, 2001. Proceedings of the 2001*, vol. 1, pp. 43–49, IEEE, 2001.

[7] H. Wang, Q. Li, and N. Cheng, "Real-time path planning for low altitude flight based on a* algorithm and tf/ta algorithm," in *Automation Science and Engineering (CASE), 2012 IEEE International Conference on*, pp. 837–842, Aug 2012.

[8] H. zhong Zhuang, S. xin Du, and T.-J. Wu, "Real-time path planning for mobile robots," in *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, vol. 1, pp. 526–531, Aug 2005.

[9] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *Robotics, IEEE Transactions on*, vol. 21, pp. 354–363, June 2005.

[10] V. J. Lumelsky and A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment," *Automatic Control, IEEE Transactions on*, vol. 31, pp. 1058–1063, Nov 1986.

[11] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 3310–3317 vol.4, May 1994.

[12] J. N. K. B. Beeson, P., "Towards autonomous topological place detection using the extended voronoi graph," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 1077–1082, Sept 2008.

[13] N. Arana-Daniel, R. Rosales-Ochoa, C. Lopez-Franco, and E. Nuno, "Reinforcement learning-slam for finding minimum cost path and mapping," in *World Automation Congress (WAC), 2012*, pp. 1–6, June 2012.

[14] Y. Huang and K. Gupta, "Rrt-slam for motion planning with motion and map uncertainty for robot exploration," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 1077–1082, Sept 2008.

[15] T. Ersson and X. Hu, "Path planning and navigation of mobile robots in unknown environments," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 2, pp. 858–864 vol.2, 2001.

[16] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp.," in *Robotics: Science and Systems*, vol. 2, 2009.

[17] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 14, pp. 239–256, Feb 1992.

[18] J. L. Martınez, J. González, J. Morales, A. Mandow, and A. J. Garcıa-Cerezo, "Genetic and icp laser point matching for 2d mobile robot motion estimation," *chapter book in Studies in computational intelligence edited by stefano Cagnoni, Springer-Verlag Berlin Heidelberg*, 2009.

[19] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, pp. 46–57, June 1989.

[20] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[21] A. S. Huang, E. Olson, and D. C. Moore, "Lcm: Lightweight communications and marshalling," in *Intelligent robots and systems (IROS), 2010 IEEE/RSJ international conference on*, pp. 4057–4062, IEEE, 2010.