Section A-Research paper
ISSN 2063-5346

Path Planning for Mobile Robots using MATLAB and QBOT

[1]Prasanthi Rathnala, [2]Srinivasa Rao Sura, [3]M.S.Pradeep Kumar Patnaik,[4]Manapuram Uma Nitish Babu

[1,2,3,4]Department of EECE,GITAM School of Technology, GITAM, Visakhapatnam, India.

prathnal@gitam.edu, ssura@gitam.edu, kmanipat@gitam.edu, 121910403027@gitam.in

ABSTRACT:

The main aim of this project is to find the shortest path for the autonomous vehicle by using the MATLAB tool which consists of different libraries and which is classified on the concept of Motion planning. Motion planning is like path planning which is to find a sequence of valid configurations that moves the object from source to destination. Path planning algorithms like RRT,A* algorithms are used for developing mobile robots, and autonomous cars in order to find safe, efficient working. Path planning also provides collision-free paths, and least cost travel paths from a source to the destination. Path planning helps in safe and effective navigation, and the optimal algorithm depends on the robot geometry. One such best way for choosing the best path planning approach to avoid path planning problems is using a 'Graph'. There are two popular approaches for creating a graph one is "Search based" algorithm like A* and the other RRT and RRT* are comes under "sampling based" algorithms.

Keywords: MATLAB, Autonomous vehicles, A* algorithm, RRT* algorithm.

INTRODUCTION:

Path planning is a process that involves generating a collision-free path from a starting point to a desired destination while avoiding obstacles along the way. The objective of path planning is to find the optimal path, which could be the shortest path, the safest path, the path with minimal fuel requirements, or one that maximizes area coverage and minimizes energy consumption. Path planning is critical in various fields, including autonomous Vehicles, unmanned aerial vehicles, medical operations, surveillance, and space missions.

To achieve the optimal criteria, path planning algorithms use different techniques such as Dijkstra algorithm and Sampling Based Planning (SBP). SBP has many advantages, including low computational cost and better success rates for complex problems. Rapidly-exploring Random Tree (RRT) and Rapidly-exploring Random Tree Star (RRT*) are some of the popular SBP algorithms used in path planning.

RRT is a simple algorithm that involves connecting the closest available nodes with randomly generated points. It has three steps: selection of a vertex for expansion, expansion, and terminating condition. RRT's main objective is to solve path planning issues, and it generates irregular paths that are solved by RRT*. RRT* is implemented to increase the speed of the algorithm, and it also supports non-holonomic constraints and dynamic environments.

As the RRT algorithm keeps searching for the closest available nodes, the optimal path cost increases depending on the number of vertices generated. Despite this, the algorithm is still fast compared to other path planning algorithms.

RRT *:

RRT* is an enhanced version of RRT that is more effective than its predecessor. As the number of nodes increases, RRT* can find the shortest optimal path to the goal while avoiding obstacles. The primary objective of RRT* is to find the shortest path. Although the principles of RRT and RRT* are similar, the key differences lie in their results. RRT* records the distance travelled from each vertex to its parent vertex, which is known as the cost of the vertex.

When searching for the closest node in the graph, RRT* calculates a neighbourhood of vertices in a fixed radius from the new node. However, RRT* performance decreases when the images are not clear. The algorithm takes longer to complete a single path on average than the default version after observing neighbouring nodes and rewiring the graph. In this process, the major computing effort comes from obstacle avoidance. As long as a node can reach its neighbour node, it is kept on rewiring with the other nodes to find the optimal path between the source and destination.

A* ALGORITHM:

A* is a popular graph traversal and path search algorithm that is widely used in computer science due to its completeness and optimal efficiency. However, one of its main practical drawbacks is its high space complexity, as it stores all generated nodes in memory. While there are techniques to pre-process the graph to improve performance and memory usage, A* is still the best solution in many cases.

Developed in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael at the Stanford Research Institute (now SRI International), A* is an extension of Dijkstra's algorithm that only finds the shortest path from a source to the goal. Despite its age, A* remains a valuable tool for path planning in many applications such as autonomous vehicles, medical procedures, surveillance operations, and space missions. While A* may have more lines of code and take longer to execute compared to newer algorithms, it is still a reliable and effective solution for many path planning problems.

PRINCIPLE OF OPERATION:

In Fig.1, the detailed block diagram of path planning for mobile robots using MATLAB &QBOT presented. The methodology of path planning constitutes two phases and they are as follows:

- Start QBOT
  - ➢ Starting the QBOT and making connectivity.
- Generate occupancy map using QBOT
  - ➢ Generating occupancy map by simulating in Simulink tool.

To generate this occupancy map and to operate we required MATLAB Simulink. These two phases can again be divided into six steps namely:

- Connectivity
- QBOT Placing
- Simulink
- Interface
- Mapping keyboard
- Deploy & Start

The Phase 1 involves Connectivity, which is shown in Fig.2.To connect the QBOT Plug in the power adapter supplied with the wireless router. Switch on the wireless router. Connect the PC to any of the ethernet/LAN ports on the router ports with the provided ethernet cable. Using the Windows Network system icon in the taskbar, Open the Network & Internet Setting and Click on Change adapter options. As shown in the Fig.2.Ping the router by typing ping 192.168.223.2 in the Run box

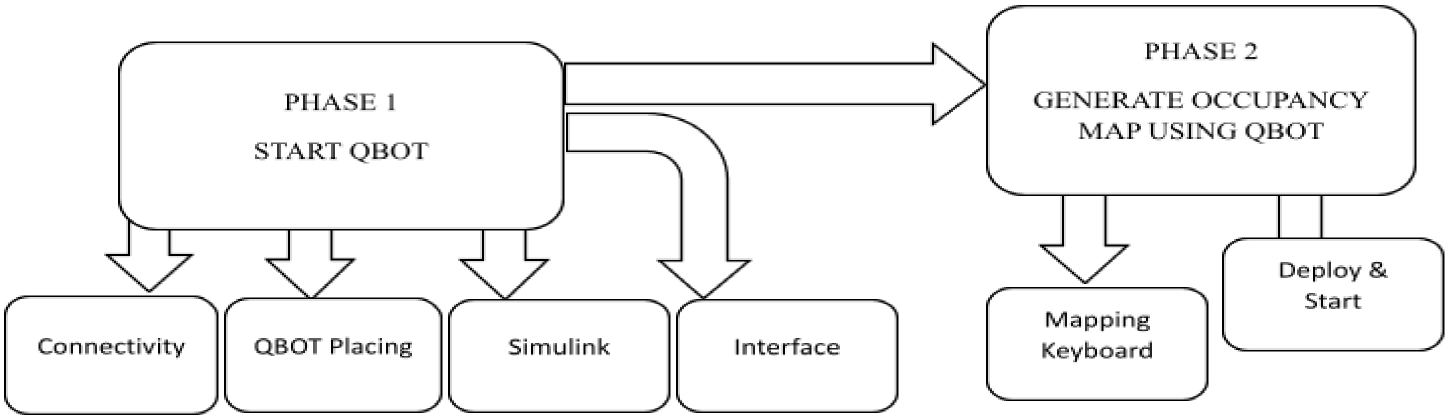
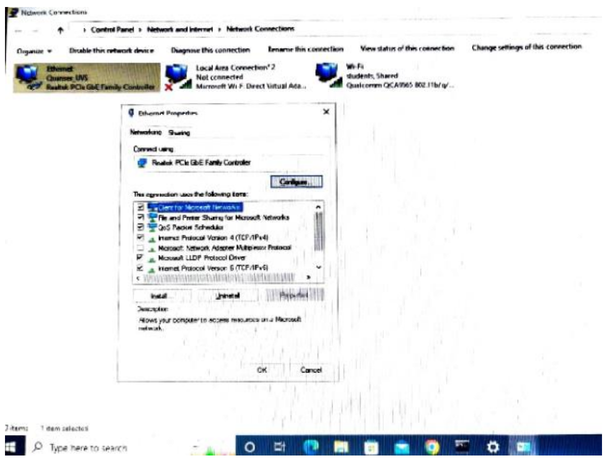


Fig. 1: Block diagram of Planning for mobile robots using MATLAB & QBOT


Network Connections
› Control Panel › Network and Internet › Network Connections
Organize ▾ Disable this network device Diagnose this connection Rename this connection View status of this connection Change settings of this connection
Ethernet
Quanser_UVS
Realtek PCIe GbE Family Controller
Local Area Connection* 2
Not connected
Microsoft Wi-Fi Direct Virtual Ada...
Wi-Fi
students, Shared
Qualcomm QCA9565 802.11b/g/...

Ethernet Properties
Networking Sharing
Connect using:
Realtek PCIe GbE Family Controller
Configure...
This connection uses the following items:
- [✓] Client for Microsoft Networks
- [✓] File and Printer Sharing for Microsoft Networks
- [✓] QoS Packet Scheduler
- [✓] Internet Protocol Version 4 (TCP/IPv4)
- [ ] Microsoft Network Adapter Multiplexor Protocol
- [✓] Microsoft LLDP Protocol Driver
- [✓] Internet Protocol Version 6 (TCP/IPv6)

Install... Uninstall Properties
Description
Allows your computer to access resources on a Microsoft network.
OK Cancel

3 items 1 item selected
Type here to search

Fig. 2: Examining the Connectivity

The QBOT placing is placing the QBOT 2e in a clear space of at least 2m x 2m with several obstacles strategically placed around the robot. Open Simulink model file. As shown in the Fig.3.

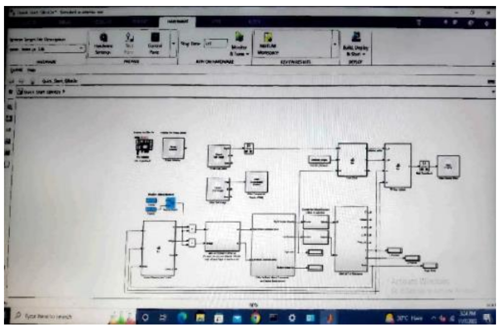


Fig.3: Simulink model file:Quick_Start_QBOT2e.mdl.

The next step is interface. Set the IP. Then Click on the Build model button in the Simulink toolbar. After 5 seconds enable the model using the manual switch highlighted in blue. The robot will begin to rotate slowly. After a complete rotation the robot will move forward 0.5m, make 180 degrees turn, and then return to a starting point. Then Click on the stop button in the Simulink toolbar to stop the model.

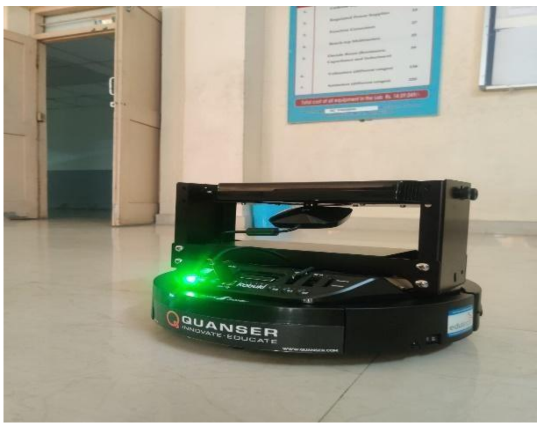


Fig. 4: QBOT 2e

The Phase 2 includes generating Occupancy map using QBOT 2e as shown in Fig.5. Presently Open QBot 2e_2D_Mapping_Keyboard.mdl Simulink file. Provide IP address. Deploy and connect &start the QBOT 2e. A map will be generated Right click on the map and save it to the workshop save it as MAP_OBS.As shown in Fig.6. Now go to the MATLAB workshop, find the item, right click on it and save it as Map_obs.mat in some folder.

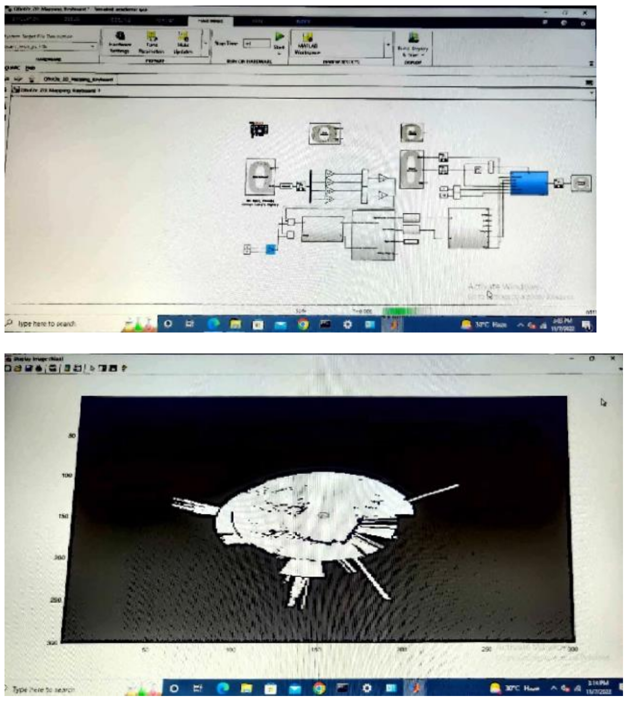
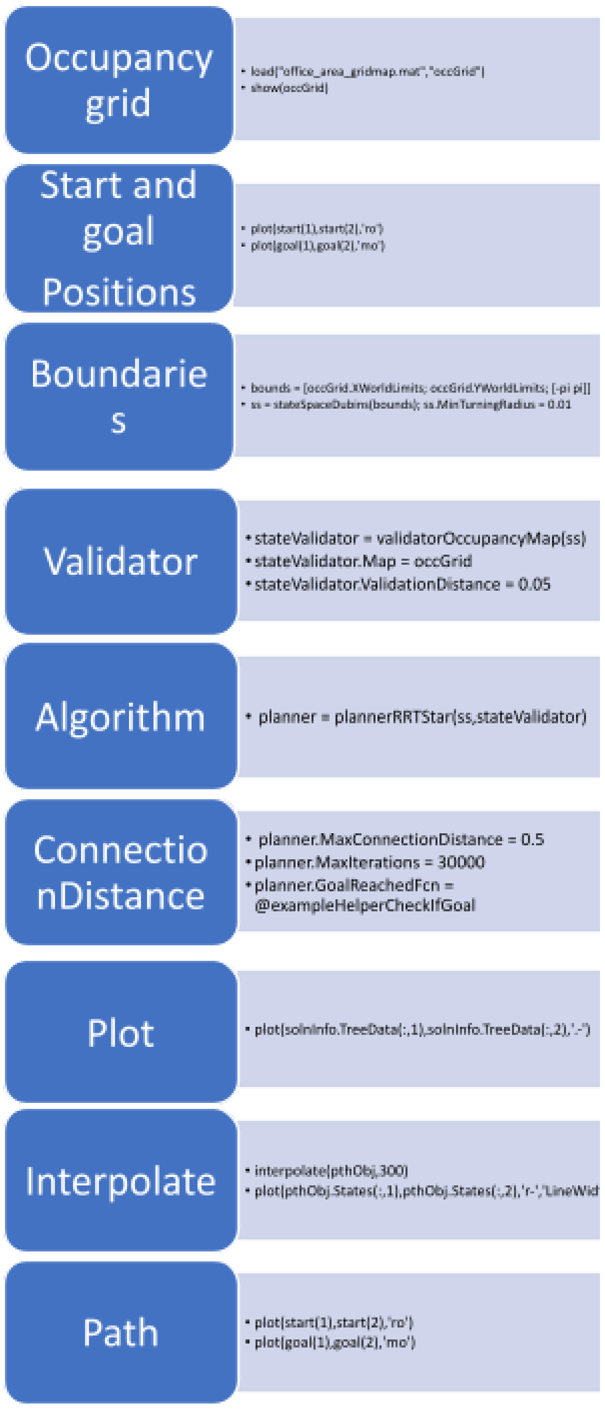


Fig. 5: QBOT E2e mapping Keyboard and Occupancy grid 3D view.



```
• load("office_area_gridmap.mat","occGrid")
• show(occGrid)

• plot(start(1),start(2),'ro')
• plot(goal(1),goal(2),'mo')

• bounds = [occGrid.XWorldLimits; occGrid.YWorldLimits; [-pi pi]]
• ss = stateSpaceDubins(bounds); ss.MinTurningRadius = 0.01

• stateValidator = validatorOccupancyMap(ss)
• stateValidator.Map = occGrid
• stateValidator.ValidationDistance = 0.05

• planner = plannerRRTStar(ss,stateValidator)

• planner.MaxConnectionDistance = 0.5
• planner.MaxIterations = 30000
• planner.GoalReachedFcn =
  @exampleHelperCheckIfGoal

• plot(solnInfo.TreeData(:,1),solnInfo.TreeData(:,2),'.-')

• interpolate(pthObj,300)
• plot(pthObj.States(:,1),pthObj.States(:,2),'r-','LineWid

• plot(start(1),start(2),'ro')
• plot(goal(1),goal(2),'mo')
```

Fig.6: Flow chart for path planning algorithm of different code blocks.

METHODOLOGY:

Fig.6 represents flow charts for path planning algorithms of different code blocks. In this we will be operating with different algorithms like RRT* and Hybrid A* In order to verify whether, we obtain optimal path are not.

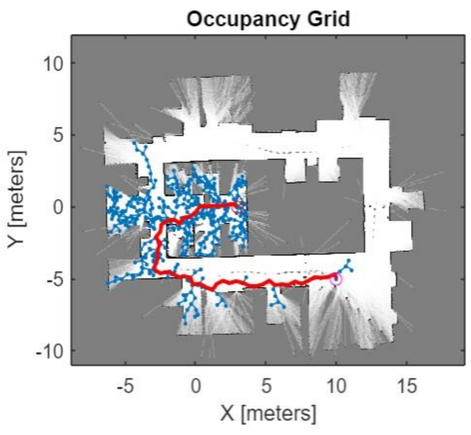
To obtain an optimal path using these algorithms, we use different code blocks. The code blocks used here are namely:

Occupancy grid

- Start and goal Positions
- Boundaries
- Validator
- Algorithm
- Connection distance
- Plot
- Interpolate
- Path

In the first step we need to install the occupancy grid map in the MATLAB tool. Then set start and goal points and insert start and goal positions of the robot. Then identify the grid boundaries using state space dubins. State space dubins are used to know the boundaries inside the occupancy map that need to be sampled. State validator which helps to examine the obstacle in the occupancy grid map for every interval. The next code block is an algorithm which we used to implement the algorithms. Maximum Connection distance will determine how many states should be sampled. If the maximum connection distance is less then it results in more number of states that will be sampled and connect to each other and vice versa. Goal Reached Fcn will determine whether the planner reached the goal or not. This is an inbuilt parameter but we can also customize it as per our requirement. The Plot code block will plot the entire path in the form search tree. Then we need to interpolate and Show start and goal points in the grid map in order to obtain an optimal path. The code used here is simple MATLAB code which specifies the optimal path for any mobile robot.

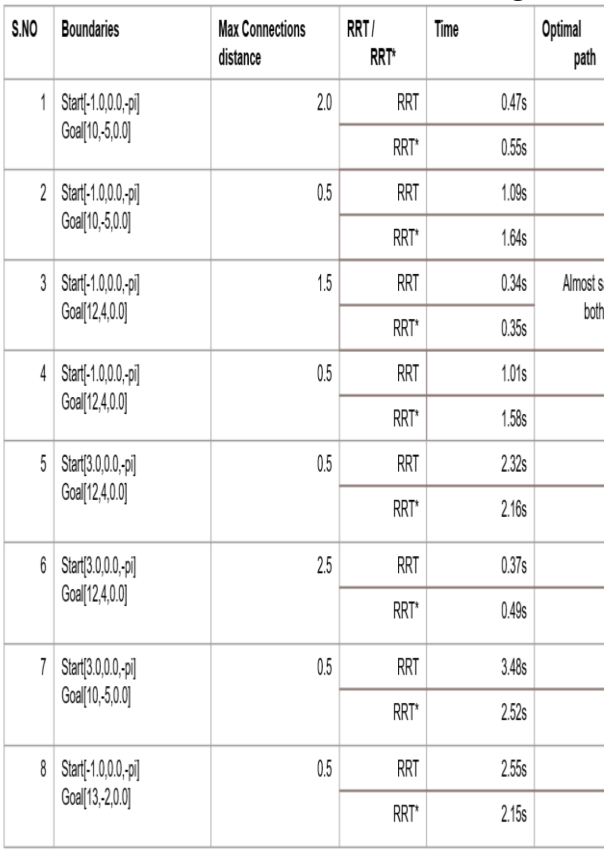
RESULTS:



```
Elapsed time is 4.728544 seconds.
```

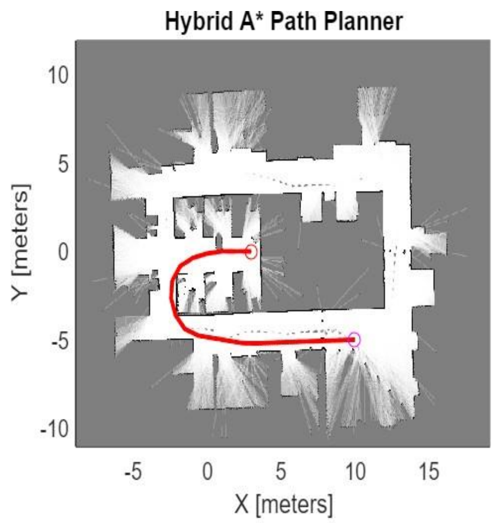
Fig.7: Optimal path generated by RRT* algorithm.

Fig.7 is the optimal path generated by RRT* algorithm from start to goal points.The RRT*algorithm mainly uses maximum connection distance to find the optimal path.In order to differentiate RRT* and RRT algorithms a table has been build.Which is shown in Fig.8.



S.NO	Boundaries	Max Connections distance	RRT / RRT*	Time	Optimal path
1	Start[-1.0,0.0,-pi] Goal[10,-5,0.0]	2.0	RRT	0.47s	
			RRT*	0.55s	
2	Start[-1.0,0.0,-pi] Goal[10,-5,0.0]	0.5	RRT	1.09s	
			RRT*	1.64s	
3	Start[-1.0,0.0,-pi] Goal[12,4,0.0]	1.5	RRT	0.34s	Almost s both
			RRT*	0.35s	
4	Start[-1.0,0.0,-pi] Goal[12,4,0.0]	0.5	RRT	1.01s	
			RRT*	1.58s	
5	Start[3.0,0.0,-pi] Goal[12,4,0.0]	0.5	RRT	2.32s	
			RRT*	2.16s	
6	Start[3.0,0.0,-pi] Goal[12,4,0.0]	2.5	RRT	0.37s	
			RRT*	0.49s	
7	Start[3.0,0.0,-pi] Goal[10,-5,0.0]	0.5	RRT	3.48s	
			RRT*	2.52s	
8	Start[-1.0,0.0,-pi] Goal[13,-2,0.0]	0.5	RRT	2.55s	
			RRT*	2.15s	

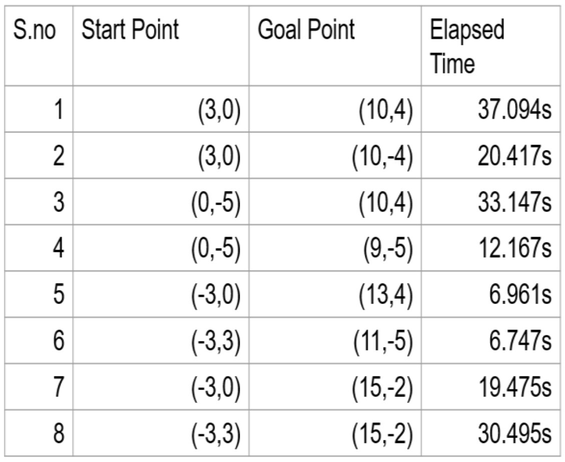
Fig.8: Observations from RRT* & RRT algorithms.



```
Elapsed time is 30.595108 seconds.
```

Fig.9: Optimal path generated by hybrid A* algorithm.

Fig.9 is the optimal path generated by hybrid A* algorithm from start to goal points. The hybrid A*algorithm mainly uses state validator to find the optimal path.Observations table has been bulid using hybrid A* algorithm.Which is shown in Fig.10.



S.no	Start Point	Goal Point	Elapsed Time
1	(3,0)	(10,4)	37.094s
2	(3,0)	(10,-4)	20.417s
3	(0,-5)	(10,4)	33.147s
4	(0,-5)	(9,-5)	12.167s
5	(-3,0)	(13,4)	6.961s
6	(-3,3)	(11,-5)	6.747s
7	(-3,0)	(15,-2)	19.475s
8	(-3,3)	(15,-2)	30.495s

Fig.10: Observation table for hybrid A* algorithm.

CONCLUSIONS:

This project presented the implementation of an efficient and reliable motion planning system, based on RRT*& hybrid A* algorithms. The project can be further developed by testing these algorithms in different autonomous vehicles by taking different sensors. Whenever there is absence of GPS these algorithms will help autonomous vehicles to find the optimal path. This project is also used for finding the optimal path for the mobile robots, which are present in the inner environment.

REFERENCES:

1. Lavalle, S. M. (2006). Planning algorithms. Cambridge University Press.
2. Lan, X., & Di Cairano, S. (2015). Continuous curvature path planning for autonomous vehicle maneuvers using RRT*. In European Control Conference (ECC).
3. Alejo, J. A. Cobano, G. Heredia, J. R. Martínez-De Dios, & A. Ollero. (2015). Efficient trajectory planning for WAN data collection with multiple UAVs. In Cooperative robots and sensor networks (pp. 53-75). Springer International Publishing.
4. Karaman, S., Walter, M., Perez, A., Frazzoli, E., & Teller, S. (2011). Anytime motion planning using the RRT*. In IEEE International Conference on Robotics and Automation (ICRA).
5. Lau, & Liu, H. H. T. (2013). Real-time path planning algorithm for autonomous border patrol: Design, simulation, and experimentation. Journal of Intelligent & Robotic Systems, 75, 517-539.
6. Kong, X., Duan, X., & Wang, Y. (2015). An integrated system for planning, navigation and robotic assistance for mandible reconstruction surgery. International Journal of Intelligent Service Robotics, 9, 113-121.
7. Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. International Journal of Robotics Research, 30, 846-894.
8. Elbanhawy, M., & Simic, M. (2014). Sampling-based robot motion planning: A review survey. IEEE Access, 2, 56-77.