Path Planning and Patrolling for a Car-Like robot in an Urban Campus Environment

Contents

1 Introduction **3**

2 Literature Review **4**
2.1 Motion Planning . . . 4
2.2 Patrolling Algorithms . . . 8
2.2.1 Random (RAND) . . . 8
2.2.2 Conscientious Reactive (CR) . . . 8
2.2.3 Heuristic Conscientious Reactive (HCR) . . . 8
2.2.4 Heuristic Pathfinder Conscientious Cognitive (HPCC) . . . 8
2.2.5 Cyclic Algorithm for Generic Graphs (CGG) . . . 8
2.2.6 Generalised Multilevel Subgraph Patrolling Algorithm (MSP) . . . 8
2.2.7 Greedy Bayesian Strategy (GBS) . . . 8
2.2.8 State Exchange Bayesian Strategy (SEBS) . . . 8
2.2.9 Concurrent Bayesian Learning Strategy (CBLS) . . . 8
2.2.10 Dynamic Task Assignment Greedy (DTAG) . . . 9
2.2.11 Dynamic Task Assignement based on sequential single item auctions (DTAP) . . . 9

3 Problem Formulation **10**

4 Pre processing **12**
4.1 Route Network Definition File . . . 12
4.2 Information Extraction . . . 13
4.2.1 Adjacency List . . . 14
4.2.2 Way-points Dictionary . . . 14
4.2.3 Control Points Dictionary . . . 15
4.3 Cubic Bezier Curves . . . 15
4.3.1 N degree Bezier Curve . . . 15
4.3.2 Cubic Bezier Curve Generation . . . 15

5 Autonomous Driving **18**
5.1 Trajectory Generation and Tracking . . . 18
5.2 Navigation Strategy . . . 21
5.2.1 Obstacle Free Environment . . . 21
5.2.2 Handling Obstacles and Other Cases . . . 22

6 Multi Robot Patrolling Problem **25**
6.1 Introduction . . . 25
6.2 Evaluation Criteria . . . 25

7 Simulations and Test Results **27**
7.1 Test Layout . . . 27
7.2 Additional Layouts . . . 28
7.2.1 Road network . . . 28
7.2.2 Star . . . 29
7.2.3 Spiral . . . 30
7.2.4 Trefoil . . . 31
7.3 Occupancy Grid for Obstacle Detection . . . 32
7.4 Software Architecture . . . 33
7.5 Autonomous Driving Case Studies . . . 35
7.5.1 Stop Point . . . 35

7.5.2 Follow or Overtake . . . 37
7.5.3 Intersection . . . 41
7.6 Multi Robot Patrolling Algorithms Comparison . . . 43
7.6.1 Experimental Conditions . . . 43
7.6.2 Simulation Results . . . 44
7.6.3 Greedy Bayesian Strategy (GBS) . . . 56
7.6.4 Comparisons . . . 66
7.6.5 Conclusion . . . 66
7.7 Integration of Multi Robot Patrolling Algorithm with the Autonomous Driving . . . 67
7.7.1 Graph Structure of the Layout . . . 67
7.7.2 Patrolling Algorithm Integration . . . 68

8 Conclusion **69**

Chapter 1

Introduction

In today's world of large cities and congested roads, almost all face the wrath of heavy traffic [15]. One, then, wishes if she had some sort of super power which allowed her to port from one place to another. This seemingly unrealistic dream might not be fulfilled any time soon but there has been significant increase in efforts towards finding solutions which betters the commute experience. These have been in the form of newer methods of transport like Hyperloop, etc, large scale cab-sharing services, car-pooling, and many more.
Even though most of the methods are effective in reducing the amount of vehicles on road, they still inherit the inefficiencies of human driven endeavours. *Autonomous vehicles* addresses this in part. It removes the necessity for her to occupy a driver's seat, thus freeing her from being a chauffeur and better employ her time and resources. It also acts as an avenue to have inter-vehicle communication to better manage traffic congestion using multi-agent techniques.
The larger problem of achieving achieving autonomous navigation in a less than certain environment and with complete dependency on its on-board sensors is one of the biggest challenges of today's world. Right from auto-makers, chip-manufacturers, software-oligarchs, to multi-million dollar start-ups, are investing in finding solutions to the transportation problem. The SAE J3016 standard introduced 0-5 levels of autonomy that can be achieved (Level 0 - Complete manual control to Level 5 - Fully Autonomous Navigation). There are a few commercially manufactured vehicles which already do Level 2. But achieving Level 5 has not been a possibility until very recently due to the need for heavy computing power as well as hi-tech range of cameras and other sensors. Though various companies, like Tesla Motors, Waymo, etc [3] have claimed that a fully autonomous vehicle will be a reality in the near future, one cannot downplay for it not being a purely marketing gimmick. Though, one thing is for certain, given the heavy investment in the field of autonomous navigation [3], it is only a matter of time before we have vehicles on road without people in it.
There are a few research labs around the world which have consistently worked on this problem since the inception of DARPA funded Grand Challenges ('04, '05, '07) due to various incentives provided by various government as well as private agencies. Some notable milestones in the pursuit for self driving since then include Intelligent Vehicle Future Challenges from 2009 to 2013, Hyundai Autonomous Challenge in 2010, the VisLab Intercontinental Autonomous Challenge in 2010, the Public Road Urban Driver-less Car test in 2013 and the autonomous drive of the Bertha-Benz historic route[4].
This report encompasses the attempt made at solving the problem of *Path Planning and Patrolling for a Car-Like Robot in a Urban Campus Environment* proposed by *Center for Artificial Intelligence and Robotics, DRDO*. The task is to come up with a patrolling strategy of an urban campus environment using multiple autonomously driven car-like vehicles while maintaining unpredictability of the path traversed.
The different chapters in the report are as follows - A brief Literature review of existing academic works and other publications, is presented in (2), followed by formulation of the problem in (3). The next chapter (4) is on pre-processing the input data. The chapter on autonomous driving (5) addresses the trajectory generation and tracking aspects. A simulation is carried out on a test layout using Robot Operating System (ROS) Indigo framework[5] and Stage simulation environment[16]. The case study is presented in (**??**). We conclude by listing out a few pros as well as cons of the work carried out and propose a future direction of work in (**??**).

Chapter 2

Literature Review

2.1 Motion Planning

The research on self driving cars has made significant progress in the recent past. Several academic institutions and research industries are actively working in making autonomous driving a reality. The major impetus was generated in the DARPA Urban Challenge of 2007 [8]. Out of 11 cars selected in the final, 6 cars successfully completed the race.
Since then, several completions and demonstration of autonomous driving have taken place across the world [7]. Intelligent Vehicle Future Challenges were held from 2009 to 2013 in Chain with the initial support of the National Natural Science Foundation of China. Hyundai Autonomous Challenge took place in Seoul in 2010, 2012 and 2014 with increasing difficulties in the mission statement. VisLab with partial support from European Research Council conducted the Intercontinental Autonomous Challenge in 2010 on an almost 16000Km path from Parma, Italy to Shanghai, China. Industries like Google and Tesla are also progressing fast in the realm of self driving cars. The state-of-art research on self driving cars are presented in the recent papers by Gonzalez et al. [9], Paden et al [4] and Katrakazas et al. [10]. We present the main observations in these papers and the progress made thereafter. Table 2.1 compares the different methods implemented in the literature for decision making in self-driving cars. The methods are broadly classified into (1) graph search techniques (2) sampling based technique (3) curve fitting techniques and (4) optimisation based techniques.
In graph search techniques, the environment is discretised and a shortest path is found between the current location and the next waypoint avoiding the on-road obstacle. Vehicle constraints are not taken into consideration and so the output path is not smooth. Sampling based technique incrementally builds the path between the current location and the next point of interest assuming a discretised sampling space. Rapidly-exploring Random Tree, RRT (2.1) randomly samples the environment to quickly explore the free space. However, it generates jerky paths. On the contrary, lattice planner (2.1) satisfies the differential constraints of the vehicle resulting in a smooth path.

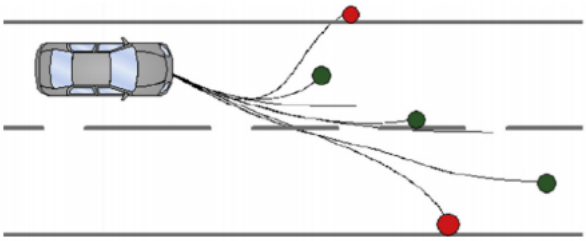


Figure 2.1: An RRT graph [10]

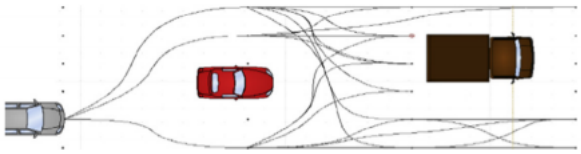


Figure 2.2: A lattice planner graph [10]

The path can be generated using curve fitting techniques. There are various methods to generate smooth curves respecting the vehicle constraints, road constraints and obstacle avoidance. The output is a parametric curve representing the path for the vehicle. Given a set of points, several paths can be generated. An optimal path can be selected based on some criteria. Optimisation based techniques produces optimal paths respecting the vehicle dynamics and road constraints. However, mostly the optimisation problems are NP hard and are computationally costly. Numerical techniques produce faster solutions though suboptimal. Moving horizon methods are also employed. Ofter more than one techniques are used to achieve local path planning problem. The winner of DARPA Urban Challenge, CMU's vehicle Boss used numerical optimisation methods for local path planning and lattice graph with graph search techniques for parking. Vislab Intercontinental Autonomous Challenge used clothoid curve generation and RRT for motion planning. The current research emphasis for autonomous driving are on risk assessment and management. In [13], [12], avoidance strategies are developed for small unexpected obstacles on the road. Handling adverse weather conditions are addressed in [11]. A paradigm shift from defensive driving nature of the self driving car is presented in [14].

Technique	Advantages	Disadavantages
Discrete representation of the environment		
Voronoi Diagrams	Completeness and maintains maximum distance from the obstacles	Limited to static environment and have discontinuous edges
Occupancy Grids Cost Maps	Fast discretisation and requires small computational power	Vehicle dynamics not taken into consideration and Errors in the presence of obstacles
State lattices	Efficient. Without increasing computational time. Precomputation of edges is possible	Problems with continuity in curvatures. Difficulties in dealing with evasive manoeuvres
Driving corridors	Continuous collision free space for the vehicle to move	Computational cost and constraints on motion
Graph Search Technique		
Dijkstra's algorithm	Finds the shortest path in a series of nodes or grid. Suitable for global planning in structured and unstructured environments.	The algorithm is slow in vast areas due to the important amount of nodes. The search is not heuristic. The resulting path is not continuous and also not optimal. Not suitable for real time applications.
A* family	Based on the Dijkstra algorithm. The search is heuristic reducing computation time.	The resulting path is not continuous. The heuristic rule is not straightforward to find most of the times.
Incremental Search Technique or Sampling Based Technique		

RRT family	Kinematic and real-time feasibility. Able to provide a fast solution in multi-dimensional systems. The algorithm is probabilitically complete. Suitable for local and global planning. Paths are asymptotically optimal in RRT*	The resulting trajectory is not continuous and therefore jerky. The optimality of the path strongly depends of the time frame for the RRT* case. Heavily dependent on the nearest neighbour heuristic to expand. Each node of the tree needs to be checked for collisions while the tree is expanding. Advanced techniques for collision checking pre-suppose perfect knowledge of the environment
Lattices Planner	Able to handle several dimensions (position, velocity, acceleration, time). Suitable for local planning and dynamic environments. Precomputation of edges is possible. Generally appropriate for dynamic environments. Paths comply with the dynamic and kinematic abilities of the vehicle	Computationally costly due to the evaluation of every possible solution in the database. The planner is only resolution complete. Problems with curvature and restrict motion. Time inefficiency with the calculation of a path for evasive manoeuver. May lead to exhaustive sampling or oscillations. Not optimal
Geometric Curve Optimization Techniques		
Interpolating curve planner	Optimisation of the curvature and smoothness of the path is achieved through the implementation of CAGD techniques (compared here below). Suitable for local planning oriented to comfort and safety in structured environments. Depends on a global planning or global waypoints.	Time consuming when managing obstacles in real time because the optimisation of the path and consideration of road and ego-vehicle constraints.
Line and circle	Low computational cost. Simple to implement. Assures the shortest path for a car-like vehicle.	The path is not continuous and therefore jerky, making non-comfortable transitions between segments of the path. The planner depends on global waypoints.
Clothoids	Transitions to and from curves are done with a linear change in curvature. Highways and road designs implement these curves. Suitable for local planning.	Time consuming because of the integrals that define the curve. The curvature is continuous but not smooth (linear behaviour). The planner depends on global waypoints.
Polynomials	Low computational cost. Continuous concatenations of curves are possible (Suitable for comfort)	Curves implemented are usually of 4th degree or higher, difficulting the computation of the coefficients to achieve a determined motion state. Usually vehicles presumed with constant velocity
Beziers	Low computational cost. Intuitive manipulation of the curve thanks to the control points that define it. Continuous concatenations of curves are possible (Suitable for comfort).	Loss of malleability when increasing the curve degree, as well as the computation time increases (Thus, more control points have to be evaluated and correctly placed). The planner depends on global waypoints.

Splines	Low computational cost. The result is a general and continuous curvature path controlled by different knots.	The solution might not be optimal (from the road fitness and curvature minimisation point of view) because its result focuses more on achieving continuity between the parts than malleability to fit road constraints.
Optimisation based Technique		
Function Optimization	Road and ego-vehicle constraints as well as other road users can be easily taken into account. Outcome is Lipschitz continuous. Locally optimal	Time consuming since the optimisation of the function takes place at a given time horizon. The planner depends on global waypoints.
Model Predictive Control	Kinematic and road constraints are considered. Smooth path. Locally optimal	Optimisation sensitive to number of variable and constraints. High complexity.
Obstacle Prediction and Intention Estimation		
Markov Decision Process and its variants	Predicts pedestrian and vehicle intentions	Intentions are assumed unchangeable. Large computational effort.
Game Theory	Handle both on-road and intersections.	Perfect information is required.
Driving corridor based methods	Applicable to both lane and intersection. Vehicle dynamics considered.	Computational effort rises with number of obstacles. Ignorance of social interaction between traffic participants. Other vehicles assumed to have constant velocity.

Table 2.1: Comparison of various techniques based on [9], [4], [10]

2.2 Patrolling Algorithms

A preliminary analysis of 11 patrolling algorithms was carried out in this report. Each of these algorithms are briefly described in this section.

2.2.1 Random (RAND)

In this strategy, the robots decide the next node to be visited by selecting it randomly amongst the neighbours of the current node they are at.

2.2.2 Conscientious Reactive (CR)

This strategy, proposed by Machado et. al. [17], determines the next node to be visited as the one with maximum idleness value amongst those in the neighbourhood of the current node. The idleness values are shared between various patrolling robots.

2.2.3 Heuristic Conscientious Reactive (HCR)

This strategy, proposed by Almeida et. al. [18], is an extension of *Conscientious Reactive* stated above in that it uses a weighted combination of idleness values and distance to the neighbouring nodes while determining the next node to be visited.

2.2.4 Heuristic Pathfinder Conscientious Cognitive (HPCC)

This strategy, proposed again by Almeida et. al. [18], considers all the nodes in the graph unlike the previous strategies. As in the *Heuristic Conscientious Reactive* the node with the ideal combination of idleness value and distance from the current node is selected to be visited next. The path to the node selected is determined by standard pathfinding algorithms like Dijkstra's, A star, etc. In the current implementation Dijkstra's Algorithm is used.

2.2.5 Cyclic Algorithm for Generic Graphs (CGG)

This is an offline strategy, proposed by Portugal et. al [19], wherein a path is determined independently for each patrolling robot either as a Hamilton Cycle if possible or one which passes through all the nodes in the graph. Once this path has been determined, the robots move along it iteratively.

2.2.6 Generalised Multilevel Subgraph Patrolling Algorithm (MSP)

This is again an offline strategy, proposed in the same work as *Cyclic Algorithm for Generic Graphs* [19], which creates n subgraphs, where n is the number of patrolling robots, in the given graph such that the distribution of nodes are equal. Each robot, then, is assigned a subgraph to patrol. The patrolling path is determined by *CGG* algorithm.

2.2.7 Greedy Bayesian Strategy (GBS)

This strategy, proposed by Portugal et. al. [20], uses Bayesian principles to determine the next node to be visited. Each robot assigns a probability based on the idleness value to each of the neighbouring nodes of the current node it is at. The node with maximal value is selected as the one to be visited next.

2.2.8 State Exchange Bayesian Strategy (SEBS)

This is an extension of *Greedy Bayesian Strategy* proposed in the same work as above [20]. In addition to idleness values, here the probability of other patrolling robots visiting the nodes is also considered while determining the posterior. Again, the node with maximal value is selected as the one to be visited next.

2.2.9 Concurrent Bayesian Learning Strategy (CBLS)

This strategy, proposed again by Portugal et. al. [21], uses reinforcement learning framework to update the probability of visiting the neighbouring nodes at every iteration. The paper claims that the prediction of each of the robot's planned path is *far from straightforward*.

2.2.10 Dynamic Task Assignment Greedy (DTAG)

This is one of the more recent works, proposed by Farinelli et. al. [22], which takes into account weighted combination of the idleness values of all the nodes in the graph, distances to them from the current node and the distance to those nodes from the starting point of the robot, to determine the next node to be visited. The information sharing is done between the robots to update the idleness values of the nodes visited by the respective robots.

2.2.11 Dynamic Task Assignement based on sequential single item auctions (DTAP)

This strategy, proposed in the same work as above [22], develops on the idea of *DTA-Greedy* to add roboustness as well as higher level of information sharing between the nodes. In this case, the patrolling robots also share the node they intend to visit next. The robot with which lowers the utility function most is then assigned with the task of visiting the concerned node whereas the other robots take up the task of visiting the next best node in their respective order.

Chapter 3

Problem Formulation

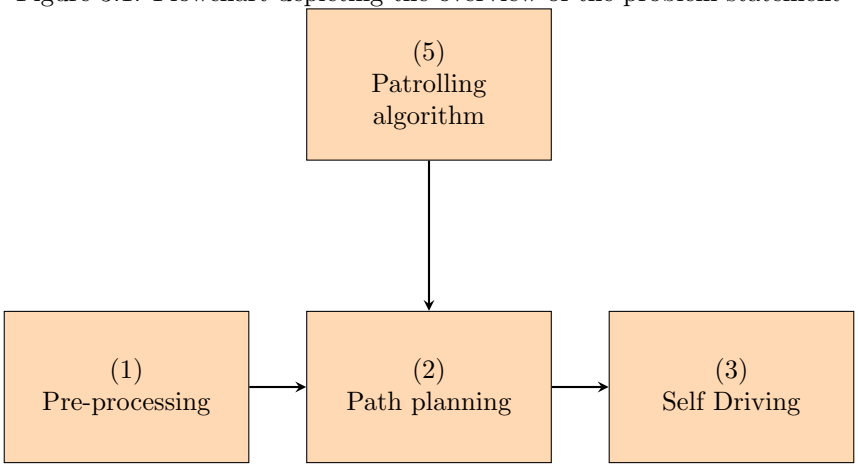
As stated in the introductory chapter, this report address a specific problem of *Path planning and Patrolling*. The project deliverables include an overall patrolling strategy which is unpredictable even on observing the strategy in action for a considerable amount of time. We are also tasked with developing a self-driving algorithm which given complete state information of the environment drives the vehicle from starting point to goal point in a desired manner (*specified path and velocity*) safely.

The following things are assumed to be given:

- A complete map of the environment with traversable roads and other areas
- Complete state information of all the moving vehicles (dynamic obstacles) in the environment
- The shape of the obstacles can be represented as a rectangle by fitting a bounding box.
- Robot Vehicles fitted with range sensors and communication equipments
- Sparse traffic
- Indian driving standards (left handed)
- Roads consist of two-lanes

The campus map is given in the form of a *Route Network Definition File* (RNDF) format (4.1). The provided RNDF is considered to be ground truth and the way-points are assumed to be lying on the middle of the lane.

Figure 3.1: Flowchart depicting the overview of the problem statement



In the above flowchart, the pre-processing block refers to creating database of useful information from the given RNDF of the map. This process is carried out only once for a given environment. Taking advantage of the provided map, the traversable paths are generated a priori (*offline*) such that between any two points on the map, a continuous path can be obtained. To define this continuous path, we have used Cubic Bezier Curve representation (4.3).
The path-planning step gives the path to be traversed by the robot-vehicle at any given time instance. The path is represented as an ordered list of way-points. The patrolling algorithm decides the path for every robot-vehicle in action. The task of multi robot patrolling can be approached in different ways depending on the required end result. For example, just in the case of security related implementations, the patrolling problem has been approached either as a coverage problem or an interaction based attacker-defender game etc. This report takes up the multi robot patrolling problem as a graph theory based wherein the key locations in the real-life environment are represented as nodes of the graph and the roads connecting them as edges.
The self-driving block refers to the actual navigation of the robot-vehicle along the given path in the desired manner. This task is divided in to two sub-tasks - *Trajectory Generation and Trajectory Tracking.* The trajectory generation step creates a velocity and profile along the given path accounting for obstacles (both static and dynamic) and also desired behaviour at special locations (intersections, etc.). The trajectory tracking step refers to the robot vehicle following the trajectory generated in the previous step. More on this is described in the chapter 'Autonomous driving'(5).

Chapter 4

Pre processing

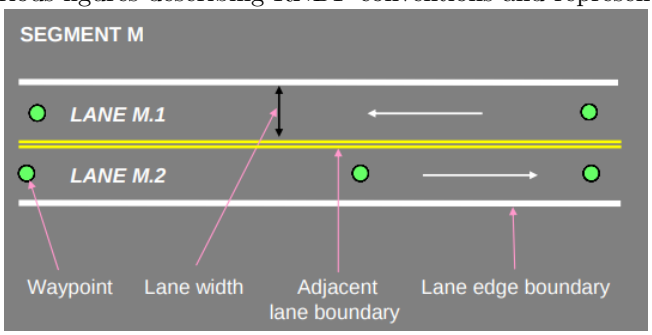
4.1 Route Network Definition File

The Route Network Definition File (or simply, RNDF) is a format of representing accessible roads and areas to a vehicle. This format was first used in DARPA (Defense Advanced Research Projects Agency of the U.S. Department of Defense) Urban Challenge 2007 [1]. It maps the roads and free accessible spaces in an area as numbered segments and zones, with each segment comprised of a number of bordered lanes containing ordered list of waypoints and every zone being defined by its perimeter. All the points are given as Latitudinal and Longitudinal co-ordinates. There are a few special points signifying stop-sign locations, exit points (connecting different segments) and parking spot locations, etc. The RNDF has no specific start point or end point and is most suitable for approximation of roads as two-dimensional networks.

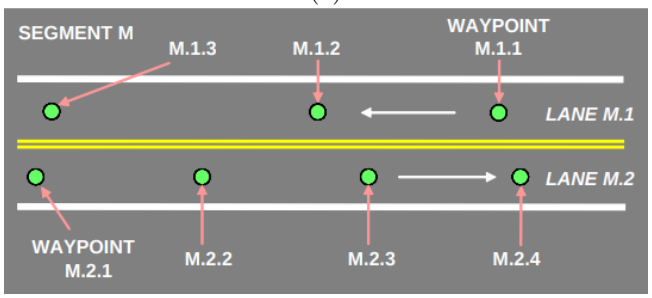
Figure 4.1: A short snippet of Sample RNDF released by DARPA [2]

```
RNDF_name         Sample_RNDF_Rev_1.5
num_segments         13
num_zones         1
format_version          1.0
creation_date         29-Mar-07
segment         1
num_lanes         2
segment_name         Michigan_Ave
lane         1.1         /*no exits, passing lane*/
num_waypoints         4
lane_width         12
left_boundary         double_yellow
right_boundary          broken_white
1.1.1         38.875413         -77.205045
1.1.2         38.875471         -77.204189
1.1.3         38.875585         -77.202593
1.1.4         38.875673         -77.201373
end_lane
lane         1.2
num_waypoints         6
lane_width         12
left_boundary         broken_white
exit         1.2.4         3.1.1
exit         1.2.6         4.1.1
1.2.1         38.875343         -77.205619
1.2.2         38.875438         -77.204198
1.2.3         38.875528         -77.202959
1.2.4         38.875602         -77.201871
1.2.5         38.875616         -77.201664
1.2.6         38.875676         -77.200830
end_lane
end_segment
```

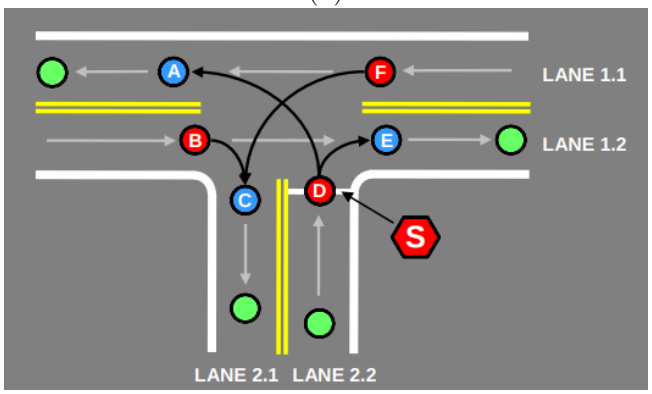
Table 4.1: Various figures describing RNDF conventions and representation scheme



(a)



(b)



(c)

4.2 Information Extraction

The input to the algorithm describing the environment is the provided RNDF. The RNDF provided is considered as be ground truth and hence it is imperative that the file represents accurately the area being mapped. This section describes the process of extracting information from it and storing it to our convenience.
There are in total of 4 files generated by the end of this process - an *adjacency list* of the directed graph representation of the road network, a *way-points dictionary* mapping every way-point ID to its description like location, etc., a *control points dictionary* mapping every pair of IDs of consecutive way-point to a set of control points defining a curve between the two way-points and lastly an image file of the layout depicting road boundaries.

4.2.1 Adjacency List

The adjacency list stores information about connectivity of individual way-points. Here, each individual way-point a node and there is a directed edge going out from it to the nodes depicting way-points it is connected. If there is a directed edge between nodes A and B, it signifies that there is a traversable path from A to B.

Figure 4.2: A part of 'adjacency_list.txt' file created for the test layout.

```
3.1.3:['3.1.4']
3.1.2:['3.1.3']
3.1.1:['3.1.2']
6.1.2:['6.1.3']
3.1.6:['3.1.7']
3.1.5:['3.1.6']
3.1.4:['3.1.5']
3.1.9:['3.1.10']
3.1.8:['3.1.9']
3.1.7:['3.1.8']
6.1.3:['1.1.1', '4.2.1', '7.1.1']
```

4.2.2 Way-points Dictionary

In the given RNDF, the way-points are represented by way-point IDs. Each ID has a latitude and longitude pair associated with it which gives the location of the way-point given by that particular ID. Way-points dictionary stores this information along with all other information about the point. There are a few special points marked as *stop points*. At these points, the vehicles are expected to stop, check if the path is free, then proceed. This information is also captured in the dictionary. Also, in-addition to this, secondary information including lane width, number of lanes to the left of current way-point and number of lanes to the right of current way-point is also stored.

Figure 4.3: A snippet of 'waypoints_dictionary.txt' created for the test layout

```
1.1.7|-141.3|86.13|0|4|0|0|1
1.1.8|-139.97|92.21|0|4|0|0|1
1.1.9|-136.53|97.36|0|4|0|0|1
1.1.10|-131.38|100.8|0|4|0|0|1
1.1.11|-125.3|102.0|0|4|0|0|1
1.1.12|-114.0|102.0|1|4|0|0|1
1.2.1|-114.0|98.0|0|4|0|0|1
1.2.2|-125.3|98.01|0|4|0|0|1
1.2.3|-129.85|97.11|0|4|0|0|1
1.2.4|-133.7|94.53|0|4|0|0|1
```

In the above figure 4.2.2 every row corresponds to a way-point in the map. The columns signify (from left to right) - *way-point ID, latitude, longitude, stop point (0 or 1), lane width, number of lanes to the left, number of lanes to the right in the same direction, number of lanes to the right in opposite direction.* The key thing to be observed is, we have taken left hand drive convention, in line with the Indian standards. Also, latitudinal and longitudinal coordinates have been substituted by Cartesian coordinates by taking centre of the map as origin of the inertial reference frame.

4.2.3 Control Points Dictionary

The control points dictionary is created after generating the curves defining the path connecting every two consecutive way-points on the map (explained in the next section 4.3). Each pair of way-point IDs is mapped to four control point locations on the map using which the curve can be defined commpletely.

Figure 4.4: A section of 'controlpoints_dict.txt' file created for the test layout

```
3.1.3|3.1.4|[212.05, 100.87]|[213.845, 100.261]|[216.705, 98.724]|[218.21, 97.57]|
3.1.2|3.1.3|[205.15, 102.0]|[207.106, 101.977]|[210.221, 101.49]|[212.05, 100.87]|
3.1.1|3.1.2|[115.0, 102.0]|[123.167, 102.0]|[196.594, 102.0]|[205.15, 102.0]|
6.1.2|6.1.3|[82.96, -101.98]|[74.512, -101.981]|[-77.552, -101.999]|[-86.0, -102.0]|
3.1.6|3.1.7|[225.88, 86.11]|[228.259, 77.497]|[270.161, -77.879]|[272.43, -86.52]|
3.1.5|3.1.6|[223.0, 92.47]|[224.073, 90.867]|[225.36, 87.993]|[225.88, 86.11]|
3.1.4|3.1.5|[218.21, 97.57]|[219.717, 96.414]|[221.944, 94.047]|[223.0, 92.47]|
3.1.9|3.1.10|[270.36, -97.19]|[269.267, -98.614]|[267.65, -99.936]|[266.04, -100.73]|
3.1.8|3.1.9|[272.67, -92.09]|[272.323, -93.881]|[271.472, -95.741]|[270.36, -97.19]|
3.1.7|3.1.8|[272.43, -86.52]|[272.904, -88.325]|[273.016, -90.307]|[272.67, -92.09]|
6.1.3|1.1.1|[-86.0, -102.0]|[-87.4, -102.0]|[-112.6, -102.0]|[-114.0, -102.0]|
```

4.3 Cubic Bezier Curves

This section describes the process of generating curves to create paths connecting the different way-points on the map. Since we have the complete map information *a priori*, we can use that to create a network of desired paths such that the path generation aspect of the problem can be handled by a one-time computation. To represent the paths, we have opted for C^1 continuous cubic bezier curves.

4.3.1 N degree Bezier Curve

A Bezier Curve of degree n can be represented as

$$P(s) = \sum_{i=0}^{n} B_i^n(s) P_i,\ 0 \leq s \leq 1 \tag{4.1}$$

where P_i are control points and $B_i^n(s)$ are Bernstein polynomials given by

$$B_i^n(s) = \binom{n}{i} (1-s)^i s^{n-i},\ i \in \{0, 1, ..., n\} \tag{4.2}$$

4.3.2 Cubic Bezier Curve Generation

The control points are generated using the way-points. The generated Bezier curve passes through the way-points. For N way-points, there will be N-1 Bezier curves generated between each pair of consecutive way-points. These Bezier curve segments are joined end to end with tangential continuity which results in C^1 along the path.
Cubic Bezier curve needs two anchor points and two control points. We set two way-points, P_i and P_{i+1}, as anchor points and compute two control points, $C_{i-1,2}$ and $C_{i,1}$, based on waypoints P_{i-1} and P_{i+2}. Now, we interpolate a curve for P_i, $C_{i-1,2}$, $C_{i,1}$, and P_{i+1} using Cubic Bezier Curve equation i.e, $n = 3$ in (4.1). Interpolated Bezier Curve can be computed using the following approach:

- Take four consecutive points $P_{i-1}, P_i, P_{i+1}, P_{i+2}$
- Find two control points $C_{i-1,2}$, $C_{i,1}$ based on the first three points P_{i-1}, P_i, and P_{i+1}
- Find the other two control points $C_{i,2}$, $C_{i+1,1}$ based on the last three points P_i, P_{i+1}, P_{i+2}
- Return the interpolated Cubic Bezier Curve using control points P_i, $C_{i,1}$, $C_{i,2}$, P_{i+1}

At any way-point, the tangents for both of the curves should be equal in order for the resulting curve to be C^1 continuous. This is achieved by selecting the control points such that the first control point of the curve ($C_{i,1}$) and the last control point of preceding curve ($C_{i-1,2}$), and the way-point at which the two curves meet (P_i), are collinear.

A simple method is presented here to determine control points for the given four consecutive points P_{i-1}, P_i, P_{i+1}, P_{i+2}. In the first step, we find the midpoints of three consecutive pairs.

$$M_1 = \frac{1}{2}(P_{i-1} + P_i),\ M_2 = \frac{1}{2}(P_i + P_{i+1}),\ M_3 = \frac{1}{2}(P_{i+1} + P_{i+2}) \tag{4.3}$$

Then, we select points on the line segments CM_1, CM_2 that join M_1 and M_2, and M_2 and M_3. The points are such that the distance of the new point on a line segment is proportional to the length of the adjacent line that joins the way-points. We calculate the proportions to select new points with below equations.

$$\begin{aligned} k_1 = \frac{|P_{i-1}P_i|}{|P_{i-1}P_i| + |P_iP_{i+1}|}, k_2 = \frac{|P_iP_{i+1}|}{|P_iP_{i+1}| + |P_{i+1}P_{i+2}|} \\ CM_1 = M_1 + k_1(M_2 - M_1), CM_2 = M_2 + k_2(M_3 - M_2) \end{aligned} \tag{4.4}$$

Now, we translate the line segments CM_1M_1 and CM_2M_2 such that CM_1 and CM_2 are aligned with P_i and P_{i+1}. Control points $C_{i,1}$ and $C_{i,2}$ are given by

$$\begin{aligned} C_{i,1} &= K(M_2 - CM_1) + P_i \\ C_{i,2} &= K(M_2 - CM_2) + P_{i+1} \end{aligned} \tag{4.5}$$

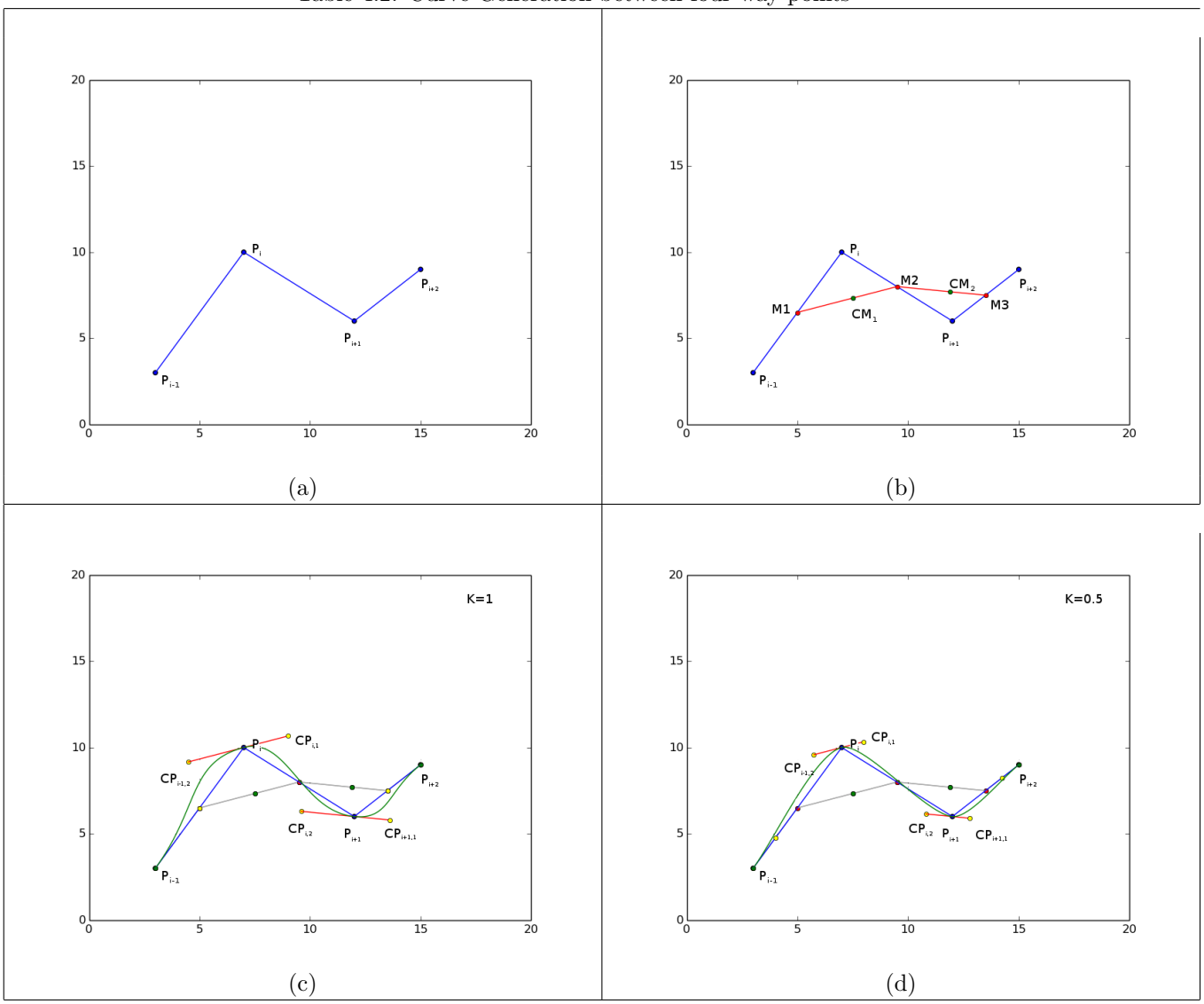
The parameter K in the above equation is calculated using the following algorithm 1. This takes in the input as way-points, desired maximum curvature for the curve being generated and a maximum deviation. It is assumed that the ideal curve is the straight line joining the way-points and hence the maximum deviation lets the user decide by what margin the path is allowed to go awry.

Algorithm 1 algorithm for computing K value

```
get waypoints, desired maximum deviation and maximum curvature
Initialize K = 0.0
repeat
    Compute Control points and generate beziér curve
    Compute max deviation and max curvature for generated curve
    if maxdeviation > desiredmaxdeviation ∩ maxcurvature > desiredmaxcurvature then
        K = K + 0.001                                          ▷ increment K
    else
        break the loop and return K value
until
return K value
```

Note: In the above computations, the assumption that the ideal curve being the straight line joining way-points breaks down in different situations. In such cases, the maximum deviation can be taken as deviation from a circle or ellipse passing through the way-points in the designed manner. Such situations need to be handled manually.

Table 4.2: Curve Generation between four way-points



Chapter 5

Autonomous Driving

5.1 Trajectory Generation and Tracking

This section deals with the defining the self-navigation problem. The car's navigation strategy has been divided into two stages *Trajectory Generation* and *Trajectory Tracking*.
An assumption is made that a virtual body with the kinematics of a *point* (henceforth, referred to as *leader*) is moving along the path generated through the piecewise C^1 continuous cubic bezier curves, deviation restricted to avoiding obstacles, with the desired linear velocity within the acceleration and deceleration bounds specified. This addresses the *Trajectory Generation* part of the problem.
The 'car' (henceforth referred to as *follower*) is then made to follow the *leader*, which is all but a *Trajectory Tracking* problem. Thus effectively, the trajectory is generated for a body with kinematics of a point which is then tracked by a body with kinematics of a car. This division does create a dead zone between the *leader* and the *follower*, but a robust controller to address the tracking problem must handle the issue within reasonable bounds.
At this stage let us assume that the path to be traversed is given as an ordered list of way-points $W(t) = \{p_1, ..., p_{i-1}, p_i, p_{i+1}, ..., p_n | i \in \mathbb{N}, p_i \in \mathbb{R}^2\}$ and is being updated at regular interval of time such that the *leader* is always in the road segment defined by W. Let $W(t) \supset S(t) = \{q_1, ..., q_k | k < n\}$ be the ordered list of stop points along the path W. These points denote entry into an intersection. Let $(p_i, c_{i,i}, c_{i,i+1}, p_{i+1})$ be the control points governing the bezier curve between the way-points (p_i, p_{i+1}). Let $s \in [0, 1]$ be the parameter of the curve such that a point $p(s)$ on the curve is give as,

$$p(s) = p_i(1-s)^3 + 3c_{i,i}(1-s)^2 s + 3c_{i,i+1}(1-s)s^2 + p_{i,i+1}s^3 \tag{5.1}$$

Let the perpendicular deviation of the point from the curve be *dev* such that the effective location of the point on the road be given as

$$\hat{p(s)} = p(s) + dev * p(s)^{\perp} \tag{5.2}$$

The lane width along the road joining way-points p_i and p_{i+1} be given by W_i and it is assumed that the roads are of two lanes
The *follower* parameters are listed in the table below

Table 5.1: Vehicle parameters

Parameter	Value	Parameter	Value
Car Length	l m	Safe distance	Δ m
Car Width	w m	Back up distance	κ m
Wheel base	wb m	Sensor Range	λ m
Axle length	la m	Min Velocity Difference	v_{dif} m/s
Min turning radius	r_{min} m	Sigmoid Limit	γ
Max steering angle	ϕ radians	Shift Per Sec	μ
Max acceleration	a m/s^2	Safe Gap	b m
Max deceleration	d m/s^2	Max linear velocity	v_{max} m/s

Most of these parameters are the constants associated with a given car, while some are to be tuned on case to case basis. Some of the non evident terms will be clarified in further sections.
The parameters of the *follower* are also associated with the *leader*. The state of the *leader* is described by $\xi(t) = (x, y, \theta, v, \omega)$ where (x, y, θ) is the pose of the point and (v, ω) were the linear and angular velocities

respectively. Let the set of obstacles at any instant of time be denoted by $O(t) = \{o_1, ..., o_m\}$ where each $o_j = (x_j, y_j, \theta_j, v_j, l_j, w_j)$ as shown in the figure below.

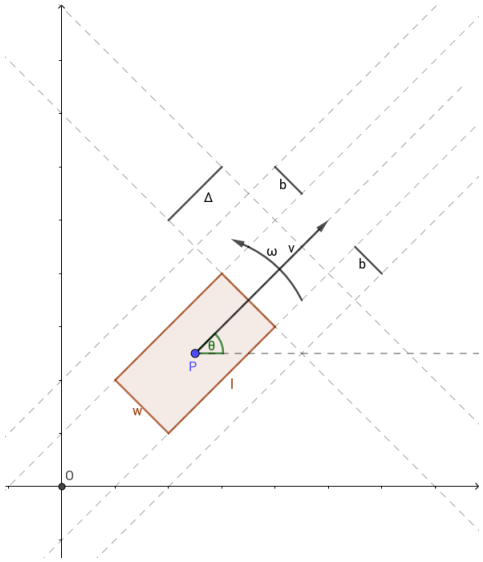


Figure 5.1: A figure describing the *leader* with different variables denoted appropriately

Trajectory Generation

The problem of trajectory generation is stated as follows:

$$\max v(t), \ \forall t \in [0, \inf) \tag{5.3}$$

subject to a set of hard constraints given by

$$\begin{aligned} p(s) &\in Road \\ v(t) &\leq v_{max} \\ \dot{v(t)} &\in [d, a] \\ (x(t), y(t)) &\notin O_{coll} \\ p(s) = q_m &\implies v(t) = 0 \quad \forall q_m \in S \end{aligned} \tag{5.4}$$

here, $O_{coll} = \cup \hat{o_j}$, $\forall o_j \in O$ such that $\hat{o_j}$ is the area defined by the boundaries of the obstacles

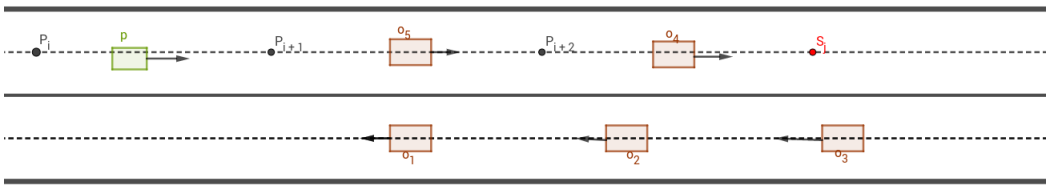


Figure 5.2: A figure describing a typical road scenario

Trajectory Tracking

The car (being referred to as *follower*) is then tasked with following the leader as *closely* as possible. Let the *follower's* state be defined as $\eta = (x_f, y_f, \theta_f, \alpha, l, w)$ where (x_f, y_f, θ_f) describes the pose of the vehicle, α the steering angle and l and w the dimensions.

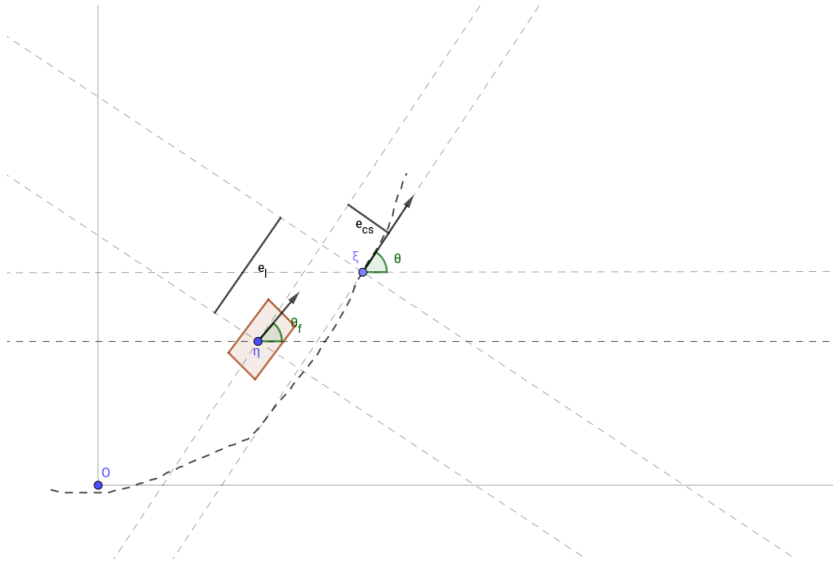


Figure 5.3: Leader-Follower

In the above figure, e_l and $e_|cs|$ denote the errors in longitudinal directions and lateral directions respectively w.r.t to the *leader*. The error in orientation is then defined as $\theta_e := \theta - \theta_f$.
The problem of trajectory tracking can thus be defined as:

$$\min C_1 e_l + C_2 |e_{cs}| + C_3 |\theta_e|, \ \forall t \in [0, \inf) \tag{5.5}$$

where C_1, C_2 and C_3 are constants weighing the respective errors.

5.2 Navigation Strategy

This section deals with the *Trajectory Generation* part of the problem. As stated above, the *leader* concerns itself with moving on a safe trajectory through the layout. Let us first address the problem of generating the trajectory in an obstacle free environment i.e, $O(t) = \emptyset$. Also for the sake of simplicity let us ignore the stop-point criteria.

5.2.1 Obstacle Free Environment

Since there are no obstacles, we can take $dev = 0$ which results in $\hat{p(s)} = p(s)$. Let the *leader* be currently in between the way-points p_i and p_{i+1} respectively. Hence, the location of the *leader* is given by (5.1).
Instantaneous curvature of the curve is given by

$$k(s) = \frac{|p_x p_{yy} - p_y p_{xx}|}{(p_x^2 + p_y^2)^{\frac{3}{2}}} \tag{5.6}$$

In addition to the constraints (5.4) described in the subsection 5.1, let us impose a design constraint to account for the vehicle dynamics such that the linear velocity of the point depends on the curvature.

$$\begin{aligned} r &= \frac{1}{k} \\ v(t) &\leq \frac{r - r_{min}}{r_{min}} \end{aligned} \tag{5.7}$$

Since, $\{1, ..., n\} \times [0, 1] \ni (i, s) \rightarrow t \in [0, \inf)$, $v(t)$ can be uniquely determined by $v(i, s)$. To keep the notations simpler, let us assume that at the current instant, $v(i, s) = v(s)$ By constraints (5.4) and (5.7),

$$v(s) = \min\{\frac{r - r_{min}}{r_{min}}, v_{max}\} \tag{5.8}$$

Here $v(s)$ is the desired linear velocity at the point defined by (i, s). On applying acceleration bounds stated as in constraints (5.4) and considering a small time step δt, $v(s)$ is determined as

$$v(s) = \begin{cases} v(s^-) + a\delta t, & \Leftarrow v(s) - v(s^-) > a\delta t \\ v(s^-) - d\delta t, & \Leftarrow v(s) - v(s^-) < d\delta t \\ v(s), & \text{otherwise} \end{cases} \tag{5.9}$$

Also by the equation for the cubic bezier curve (5.1),

$$\begin{aligned} v(s) &= \dot{s}\sqrt{p_x^2 + p_y^2} \\ \implies \dot{s} &= \frac{v(s)}{\sqrt{p_x^2 + p_y^2}} \end{aligned} \tag{5.10}$$

Given $\dot{s}$ and δt, s is updated as,

$$s \leftarrow s + \dot{s}\delta t \tag{5.11}$$

If $s > 1$,

$$\begin{aligned} i &\leftarrow i + 1 \\ s &\leftarrow s - 1 \end{aligned} \tag{5.12}$$

An updated location $p(s)$ is calculated using the equation (5.1). Also, the orientation and angular velocity is calculation as

$$\begin{aligned} \theta &= \arctan\frac{p_y}{p_x} \\ \omega &= \frac{p_x p_{yy} - p_y p_{xx}}{p_x^2 + p_y^2}\dot{s} \end{aligned} \tag{5.13}$$

Hence, the *leader's* state is updated as

$$\xi(t) = \xi(i, s) = (x, y, \theta, v, \omega)$$

.

5.2.2 Handling Obstacles and Other Cases

An obstacle is any object (stationary or in motion) in the environment other than the once specified in the pre-processed RNDF data. The obstacles are classified into two classes based on the method of handling.

- Cross traffic
- In-lane and adjacent lane traffic

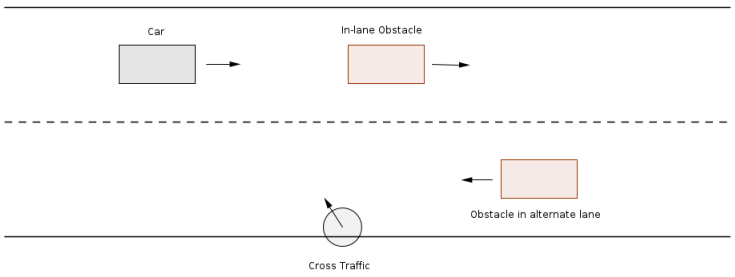


Figure 5.4: A figure describing the different categories of obstacles

The behaviour of the *leader* must be changed at one other type of situations, i.e at stop-points. The *leader* is expected to come to a halt at these points as specified in the constraints (5.4). In addressing each of the issues above, the strategy being followed is to reduce the linear velocity v of the *leader* and/or deviate from the curve by giving a *dev* in the equation (5.2) while still maintaining the C^1 continuity of the newly generated curve. Since any of the three situation might occur (simultaneously) at a given point in time, it is necessary to address these issues in parallel. Also, obstacles' linear velocity and heading direction is considered to be constant locally.

Handling a stop point

Let the current list of stop points S consists of k way-points and the vehicle has already crossed $l-1 < k$ way-points. Hence the upcoming stop-point is q_l. Let the *leader* is currently located at $p(i,s)$ with $\xi(i,s)$. Using the equations of motion, if the body is to come at a halt at a specified point c, it has to start decelerating when it is at a distance of

$$dist = \frac{v^2}{2d} \tag{5.14}$$

where, v is the initial linear velocity of the body, and d is the maximum deceleration. This gives us a check which results in a binary check to see if $p(i,s)$ is at a distance less than $dist$ from the next stop point q_l. The implicit assumption is that the *leader* is moving in a straight line towards the stop point. A suggestive linear velocity component v_{stop} computed as follows

$$v_{stop} = \begin{cases} v_{max}, & if \|p(s) - q_l\|^2 > dist \\ 0, & \text{otherwise} \end{cases} \tag{5.15}$$

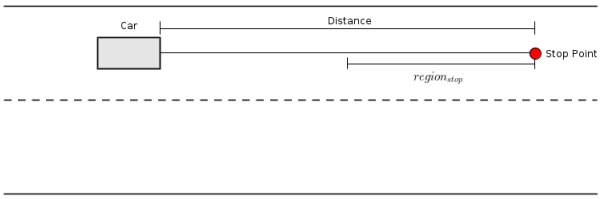


Figure 5.5: A figure demonstrating the situation wherein the object is outside the stipulated deceleration region and hence can continue with maximum allowed linear velocity

Handling cross traffic

An obstacle is classified as cross traffic if it is moving at a significant angle to the heading of the *leader* and is about to intersect *leader's* path. While this check for intersection is carried out, it is assumed that both the *leader* as well as the obstacle will maintain their current heading. The point of intersection between an obstacle o_j and *leader* $\xi(s)$ is given as

$$\begin{aligned} x_{coll} &= \frac{d-c}{a-b} \\ y_{coll} &= ax_{coll} + c \end{aligned} \tag{5.16}$$

where,
$a = \arctan(\theta)$, $b = \arctan(\theta_j)$, $c = y - ax$ and $d = y_j - bx_j$

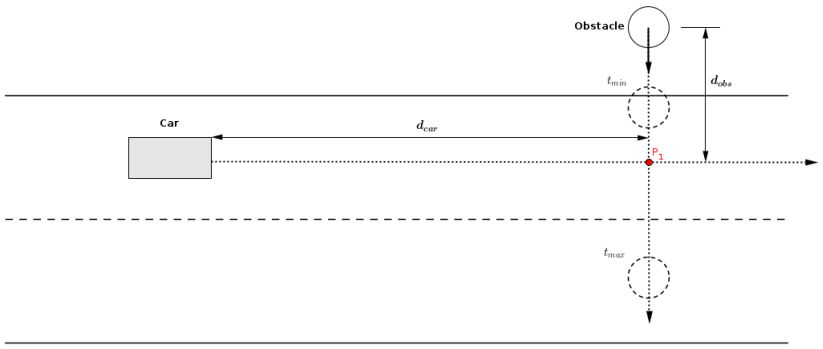


Figure 5.6: A figure depicting a cross-traffic handling scenario

Given (x_{coll}, y_{coll}), the objective then is to maintain a safe distance Δ between the obstacle and the *leader* till the *leader* crosses the point of intersection. Using the equations of motion, the time t_{car} required for the *leader* to reach (x_{coll}, y_{coll}) is given by

$$t_{car} = \frac{d_{car}}{v} \tag{5.17}$$

In order to maintain the safe distance Δ,

$$t_{car} \notin [t_{min}, t_{max}]$$

where, $t_{min} = \frac{d_{obs}-\Delta}{v_j}$ and $t_{max} = \frac{d_{obs}+\Delta}{v_j}$ A linear velocity component v_{cs} is computed as follows

$$v_{cs} = \begin{cases} \frac{d_{car}}{t_{max}}, & if t_{car} \in [t_{min}, t_{max}] \\ v_{max}, otherwise \end{cases} \tag{5.18}$$

Handling In-lane traffic

Any obstacle (stationary as well as those in motion) not addressed under the cross traffic scenario is handled using this strategy. Unlike the previous two cases, there are two variables manipulated to address this issue. The first one is a linear velocity component v_{lane} and the second a deviation dev mentioned in the equation (5.2). The different situations being handled under this class can be listed as follows:

- Inlane occupied, alternate lane free
- Both lanes occupied
- Alternate lane occupied, inlane free

In each of these cases there is a binary choice available either to *follow* i.e, mimic obstacle's behaviour at a safe distance Δ behind, or *overtake*. The *overtaking* is taken as a standard manoeuvre of going around the obstacle on its right. Any obstacle o_j is of interest if it is at a distance of d_{safe} or less from the *leader*, where

$$d_{safe} = \frac{|v - v_j|^2}{2d} + \frac{l + l_j}{2} + \Delta \tag{5.19}$$

If $\exists o_j \, s.t \, \|p_j - p\| \leq d_{safe}$, and the obstacle is in the same lane as the *leader*, then either a decision has to be made either to *follow* or *overtake*.
In case, the decision is to *overtake*, the linear velocity component v_{lane} is set to v_{max} and a desired deviation is calculated as follows $dev_{des} = dev_{obs} + \frac{w+w_j}{2} + b$ where, dev_{obs} gives how far off the obstacle is from the curve defining desired path and b is the safe gap to be maintained along transverse direction of the obstacle where as w and w_j specify width of the vehicle and the obstacle respectively. The objective is thus to be at dev_{des} from the ideal path when the *leader* is besides the obstacle. In order to shift from $dev = 0$ to $dev = dev_{des}$, we propose a sigmoid smoothing function in order to ensure C^1 continuity of the generated trajectory. From the table 5.1, we get γ and μ. Deviation dev is given by

$$dev = \frac{dev_{des}}{1 + \exp(-dev_{param})} \tag{5.20}$$

where,

$$dev_{param} = \begin{cases} 2\gamma(1 - \frac{\|p_j - p\|}{d_{safe}}) - \gamma, & \textit{leader} \text{ is behind the obstacle} \\ \max\{dev_{param} - \mu\delta t, 0\}, & \text{otherwise} \end{cases} \tag{5.21}$$

On the other hand if the decision is to *follow* the obstacle, the linear velocity component v_{lane} is set to v_j and in case of a positive deviation, dev is calculated as in (5.20) without obstacles in front.

The following table describes the action to be taken in a particular scenario

Table 5.2: Follow vs Overtake

Inlane occupied	Alternate Lane occupied	Decision	Remarks
Static	Static	Overtake	Safe gap exists
Static	Static	Follow	No safe gap
Static	Dynamic	Follow	*leader* in any lane
Dynamic	Static	Follow	*leader* in any lane
Dynamic	-	Overtake	$v - v_j > v_{dif}$
Dynamic	-	Follow	$v - v_j < v_{dif}$
-	Occupied	Follow	*leader* in alternate lane

Updating the state of the leader

The *leader's* state $\xi(s)$ is determined the same way as section 5.2.1, just that instead of the equation (5.8), $v(s)$ is given by,

$$v(s) = \min\{v_{max}, \frac{r - r_{min}}{r_{min}}, v_{stop}, v_{cs}, v_{lane}\} \tag{5.22}$$

Chapter 6

Multi Robot Patrolling Problem

6.1 Introduction

The report presents a preliminary analysis of patrolling algorithms available in the literature. The analysis is done with the objective of determining the best possible strategy for *Campus-like Environments*. The goal was thus to compare strategies on two key principles - *frequency of node visits* and *uncertainty in the path taken*.

6.2 Evaluation Criteria

The algorithms are evaluated using the parameters listed as follows:

- Average of Idleness
- Standard deviation of Idleness
- Autocorrelation of Idleness

Idleness of a node at a time instant is given as the time duration since the last visit to the node by any patrolling robot.

Average Idleness at a particular node h over the length of experiment is given by,

$$idl_{avg}(h) = \sum_{i=1}^{n(h)} \frac{idl_i(h)^2}{2T} \tag{6.1}$$

where,
$idl_i(h)$ is the idleness value recorded at every robot visit to the node,
$n(h)$ is the total number of visits to the node,
T is the total duration of the experiment.

Global Average Idleness for the experiment is given by,

$$gidl_{avg} = \sum_{j=1}^{N} idl_{avg}(j) \tag{6.2}$$

where,
N is the total number of nodes in the graph.

Average Idleness as a measure represents the consistency of the governing algorithm. It represents the frequency of visit at every node. For a good patrolling strategy, the average idleness must be as low as possible.

Standard deviation of Idleness at a particular node h over the length of the experiment is given by,

$$\begin{aligned} idl_{var}(h) &= \sum_{i=1}^{n(h)} \frac{idl_i(h)^3}{3T} - idl_{avg}(h) \\ idl_{sd}(h) &= \sqrt{idl_{var}(h)} \end{aligned} \tag{6.3}$$

Global Standard deviation of Idleness for the experiment is given by,

$$gidl_{sd} = \sum_{j=1}^{N} idl_{sd}(j) \tag{6.4}$$

Standard deviation of Idleness represents the variation at the nodes in terms of time intervals between successive visits. It tells us the degree of randomness in the robot visit.

For a patrolling strategy to be deemed successful, the intruder must not be able to penetrate and attack any node in the patrolled area. In this report, we have used graph based representation for the layout and only the nodes are assumed to be the point of interest. The adversary is assumed to have capabilities to watch and log the visits of robots at every node over an extended period of time.

To give a measure of unpredictability, autocorrelation analysis of node visits is done for every node.
Autocorrelation Function is given by,

$$idl_{cor}(h,\tau) = \frac{\sum_{i=\tau+1}^{n}(idl_i(h) - \bar{idl}(h))(idl_{i-\tau}(h) - \bar{idl}(h))}{\sum_{i=1}^{n}(idl_i(h) - \bar{idl}(h))^2} \tag{6.5}$$

where,
τ is the lag,
$\bar{idl}(h)$ is the mean of all the recorded idleness values at node h

This expression gives the degree to which the idleness value at an instant is dependent on the idleness value at τ node visits ago. Even though the idleness is not a discrete measure, this analysis gives a gross estimate of existence of a pattern over time. If there is any *significant* correlation of the current idleness with that of the past, it signifies the presence of an identifiable pattern.

For the patrol strategy to be unpredictable, it is necessary that there exists no identifiable pattern in the data available to the adversary regarding the idleness recorded at any of the robots' visit to each node, i.e, the data must be as *white* as possible. The *whiteness* of the data is claimed statistically. The assumptions about the data include the principles of *stationarity* and *ergodicity* is satisfied.

The autocorrelation function is evaluated for increasing values of τ. For a completely random signal, i.e, a white signal, $idl_{cor}(h,\tau) = 0$ for all $\tau \neq 0$. A 95% confidence bounds given by $\pm\frac{1.96}{\sqrt{n}}$ around the expected mean of 0, i.e, in order to reject the hypothesis that the signal represented by the data collected over time at the nodes of the graph is unpredictable, $idl_{cor}(h,\tau) \notin (-\frac{1.96}{\sqrt{n}}, \frac{1.96}{\sqrt{n}})$ for any node h. This check is performed both visually as well as numerically to comment on the unpredictabilty of the future node visits by the patrolling robots in at any future time instant.

Chapter 7

Simulations and Test Results

7.1 Test Layout

The simulations were done on a test layout given below (7.1). The layout contains a total of 7 segments with two-lane roads including two intersections, two T-Junctions, one segment with a lane width of 6 metres and the rest with a lane width of 4 metres. While pre-processing the RNDF corresponding to this layout, a few cases had to be handled manually. The maximum curvature was set at 0.1/m and the maximum deviation was 1.0 m.

The test vehicle parameters are given in the table below

Parameter	Value
Length	3.43 m
Width	1.515 m
Wheel base	2.36 m
Axle length	1.295 m
Minumum turning radius	2.3 m
Maximum steering angle	0.6 radians
Maximum accleration	2 m/s^2
Maximum deceleration	2 m/s^2
Maximum linear velocity	8 m/s

Table 7.1: Test Vehicle Parameters

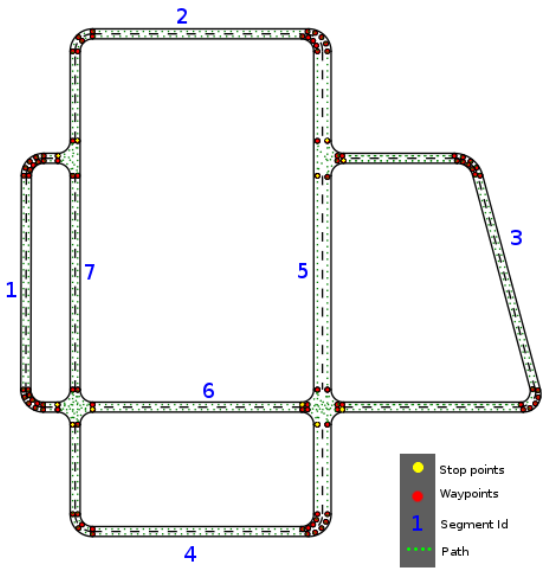


Figure 7.1: Test Layout

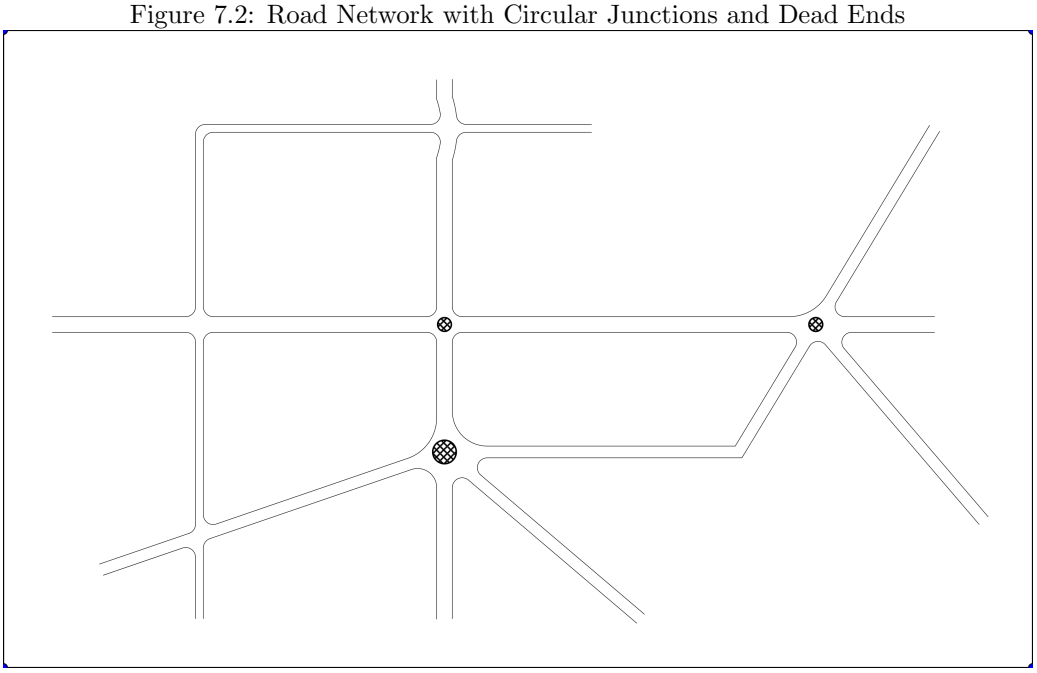
Note: Since the low level controller for trajectory tracking was not worked out, the trajectory tracking feature has not been integrated in the software and hence all the results presented below correspond to the *leader* of the robot vehicle.

7.2 Additional Layouts

The following layouts were created (RNDF files and the additional files generated after pre-processing).

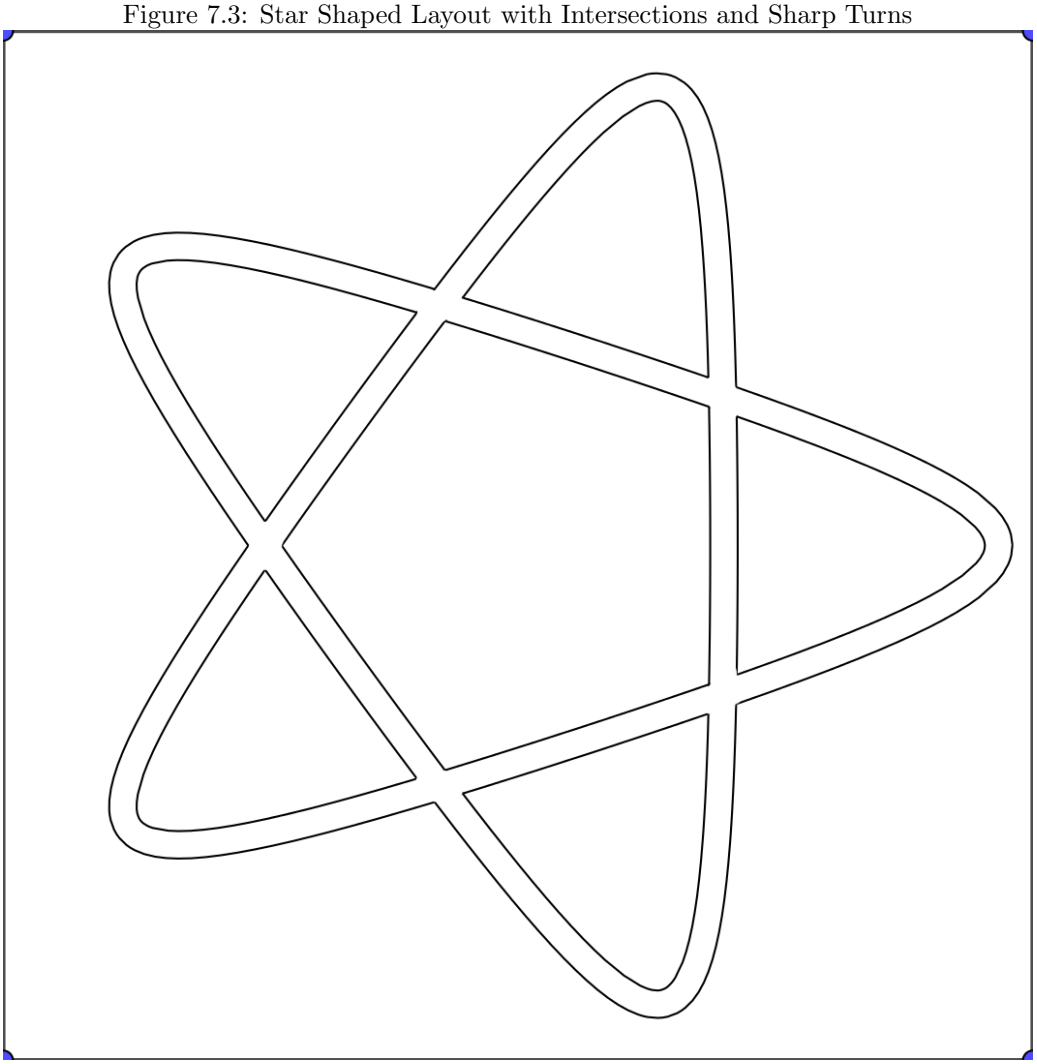
7.2.1 Road network

Figure 7.2: Road Network with Circular Junctions and Dead Ends



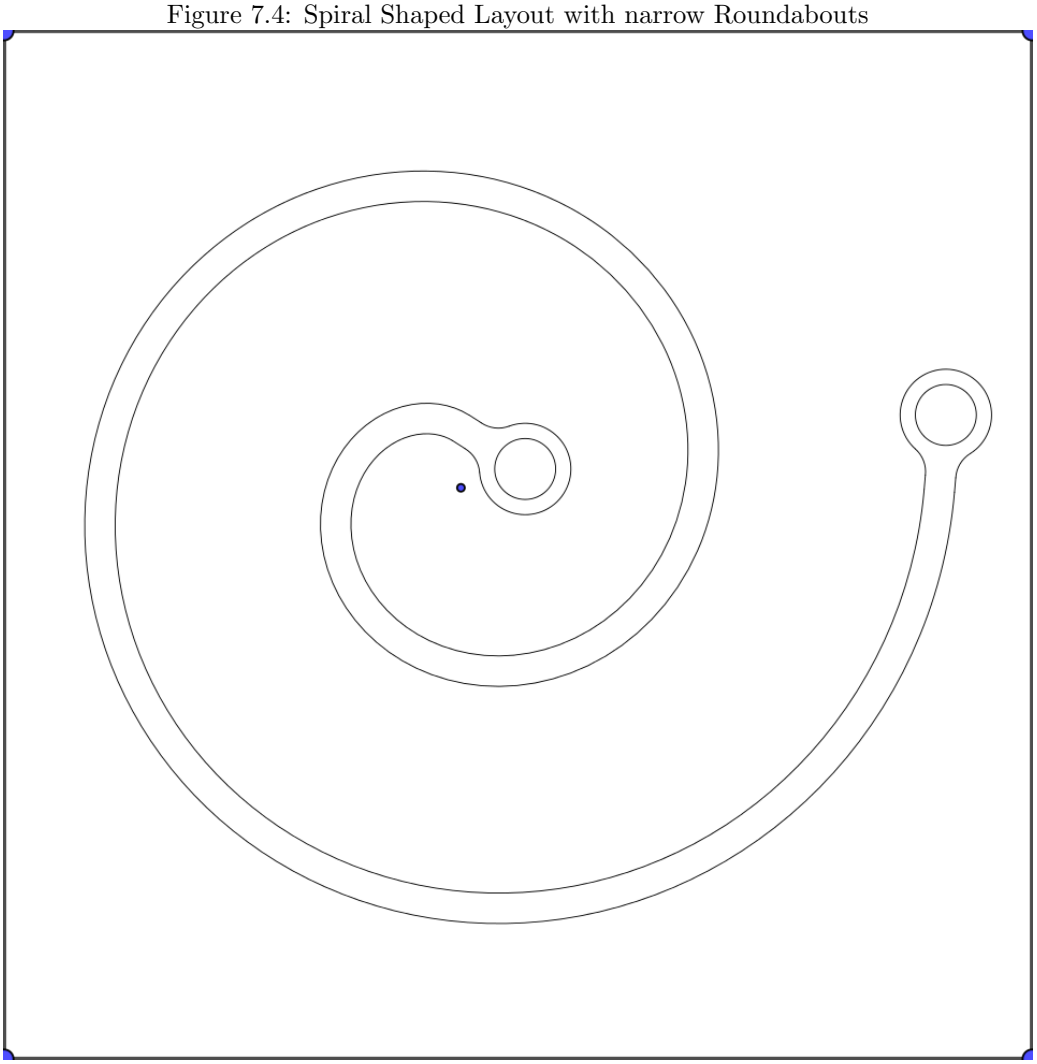
7.2.2 Star

Figure 7.3: Star Shaped Layout with Intersections and Sharp Turns



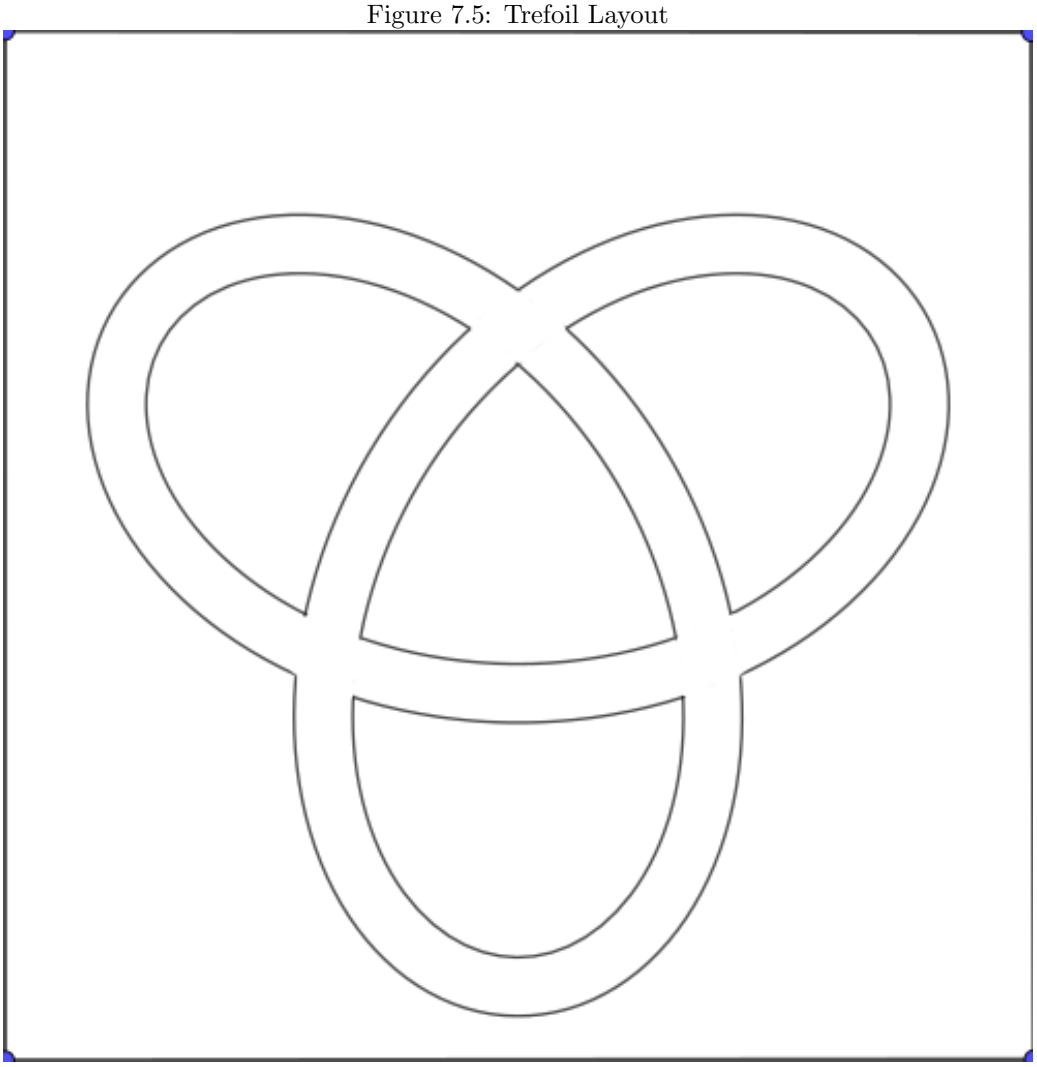
7.2.3 Spiral

Figure 7.4: Spiral Shaped Layout with narrow Roundabouts



7.2.4 Trefoil

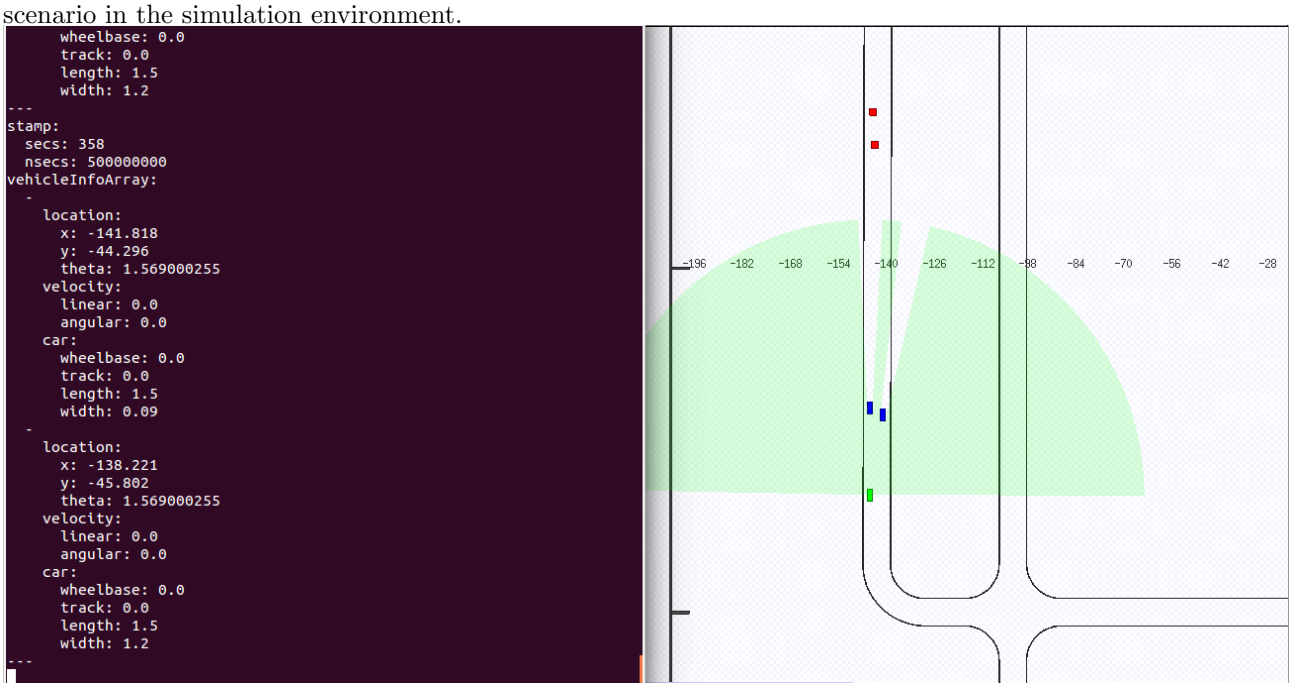
Figure 7.5: Trefoil Layout



7.3 Occupancy Grid for Obstacle Detection

The occupancy grid was created using a range sensor model available in the simulation environment. Range sensor data was converted into occupancy grid which then further was transformed into obstacle information by clustering the occupied cells in the grid using Mean-shift clustering approach and applying a bounding box on the cluster.
Note: This was not integrated during testing due to computational constraints.

Figure 7.6: The details of the obstacles are given by the occupancy grid. On the left, the time stamped processed data from the occupancy grid giving the pose and location of the two obstacles. On the right, the corresponding scenario in the simulation environment.



7.4 Software Architecture

This section explains the algorithm implemented to carry out the task. The implementation was done in ROS framework [5]. To visualize, Stage simulation environment [16] was used.
First, we initialize the test layout 7.1. Then we add one car at a time by specifying it's name, location, type i.e, Passive or Active. Here, *Passive* cars have no behaviours other than lane following with maximum feasible velocity (if set to zero, this acts as static obstacle) and *Active* cars are the ones of interest which move autonomously avoiding obstacles.
Datasets describing adjacency list of the network, locations and key features of the way-points, and control-points between every pair of connected way-points extracted after pre-processing the RNDF is made available.

Every active car has 8 nodes (scripts running in parallel) for its functioning. The interaction between 8 nodes can be observed in figure 7.7. The nodes are executed at a frequency of 20 Hz. Each of the nodes is briefly described below.

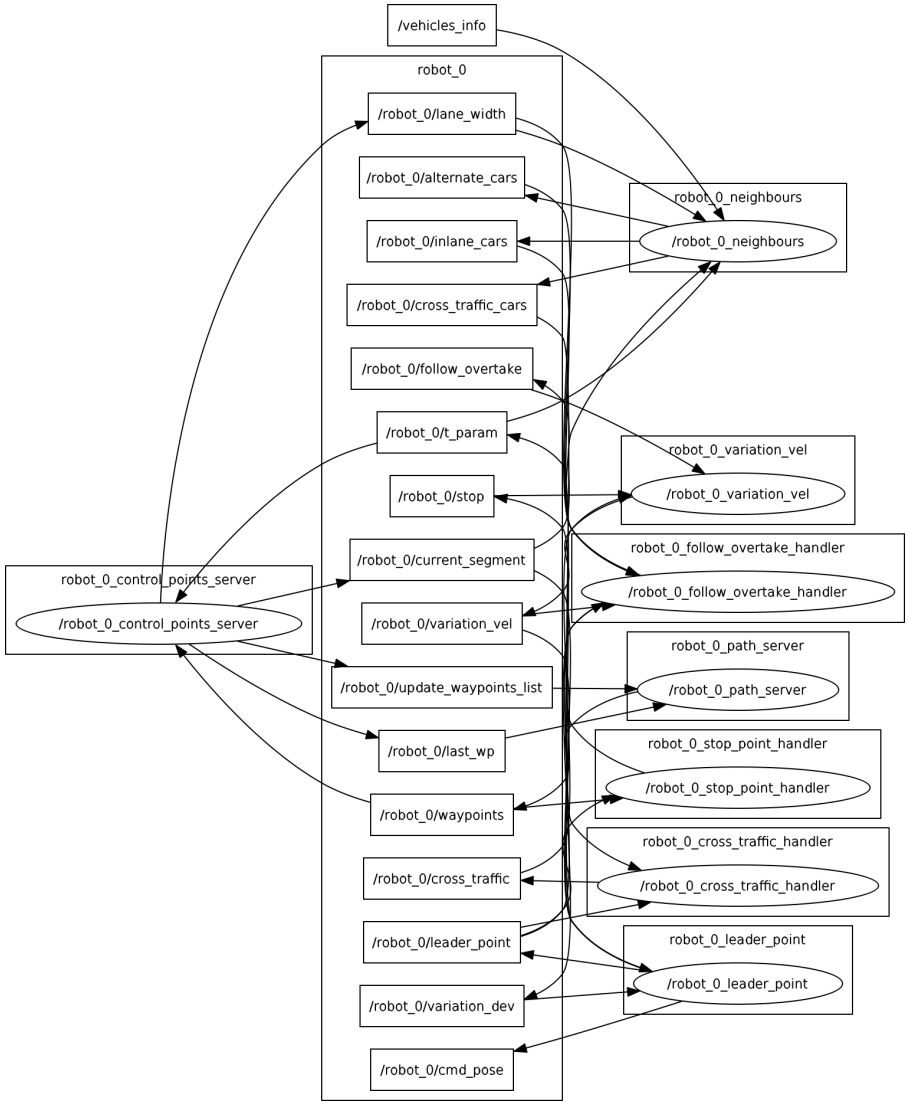


Figure 7.7: ROS rqt graph

- **Path Publisher Node**
  This node, when supplied with a way-point in the layout, generates a sequence of way-points beginning at the given way-point till a stop-point (Point signifying an intersection) is reached. At intersections, the next way-point placed in a different road segment is picked as per the patrolling strategy.

- **Control-points Publisher Node**
  This node keeps track of the pair of way-points between which is the vehicle is currently traversing. It subscribes to a topic publishing t_{param} and returns the appropriate set of control points required to generate the Bezier curve representing the path to be traversed.

- **Leader Point Node**
  This node updates the pose of the *leader*, i.e, it moves the virtual car. The motion is assumed to be taking place on the pre-specified Bezier curve base path. The update of pose is described in the section 5.2.

- **Neighbours Node**
  This node subscribes to topic publishing all vehicles' data (pose, velocity, dimensions) received from the plots. It then separates the vehicles as four kinds of obstacles - *Inlane, Cross-traffic, Alternate lane* and publishes them.

- **Stop-point Handler Node**
  This node publishes linear velocity v_{stop} as zero when the virtual car approaches a stop-point. Else v_{stop} is set to v_{max}.

- **Cross Traffic Handler Node**
  This node publishes linear velocity $v_{crosstraffic}$ depending on the density and proximity to obstacles categorized as *Cross Traffic*.

- **Follow Overtake Handler Node**
  This node publishes linear velocity $v_{followovertake}$ and desired deviation from path $d_{deviation}$ depending on the density and proximity to obstacles categorized as *Inlane and Alternate lane*.

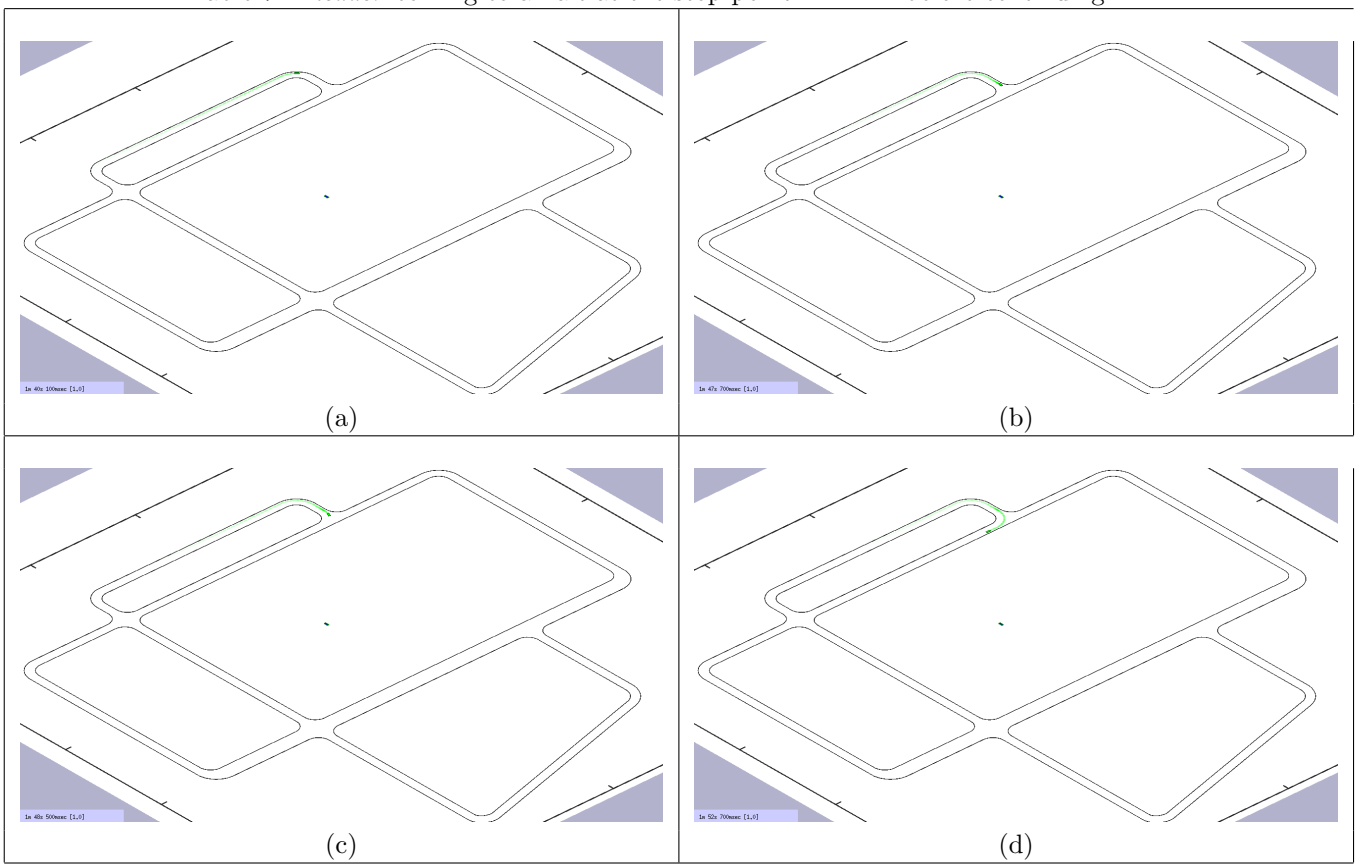
- **Variation Velocity Node**
  This node publishes linear velocity $v_{desired}$, set as minimum of $v_{stop}, v_{crosstraffic}$, $v_{followovertake}$

7.5 Autonomous Driving Case Studies

The results of a few case studies are given below. The green box in the figures below signifies the *leader* corresponding to the robot vehicle and the blue box corresponds to the obstacle.

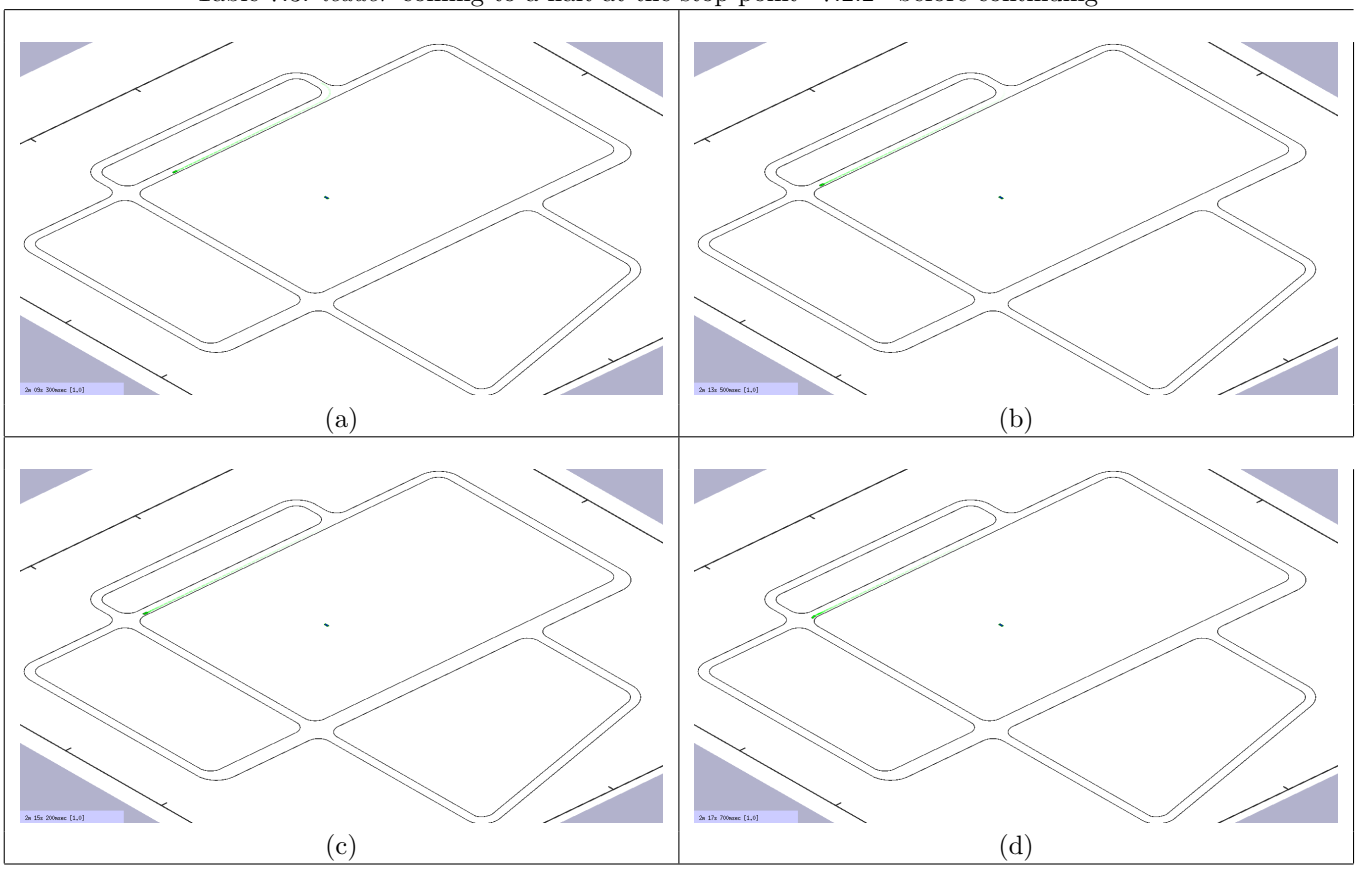
7.5.1 Stop Point

Table 7.2: *leader* coming to a halt at the stop point "1.1.12" before continuing



(a) (b) (c) (d)

Table 7.3: *leader* coming to a halt at the stop point "7.2.2" before continuing



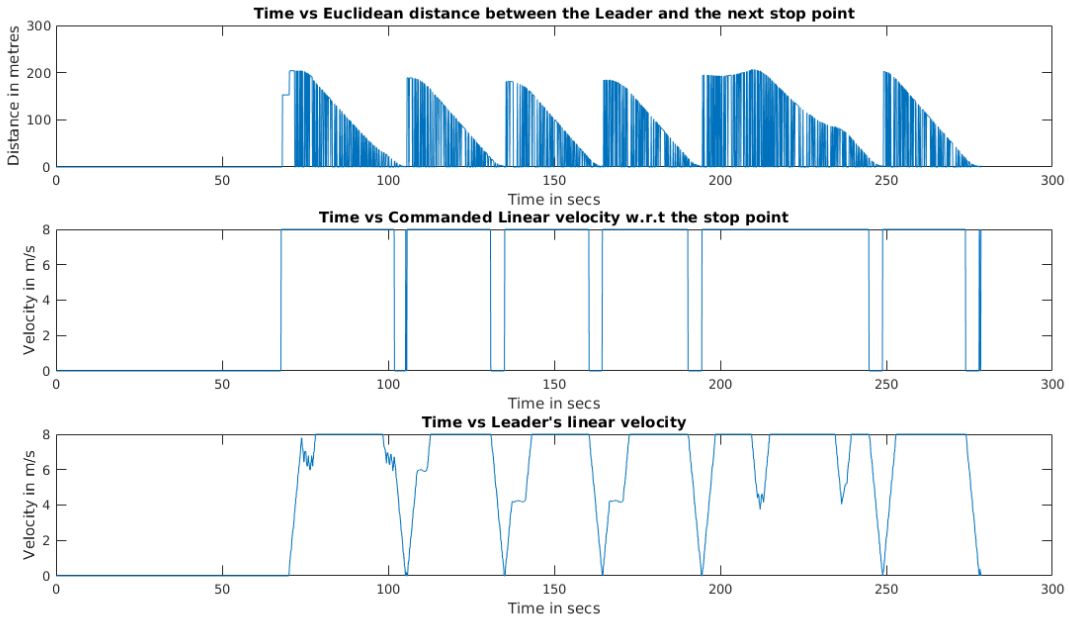
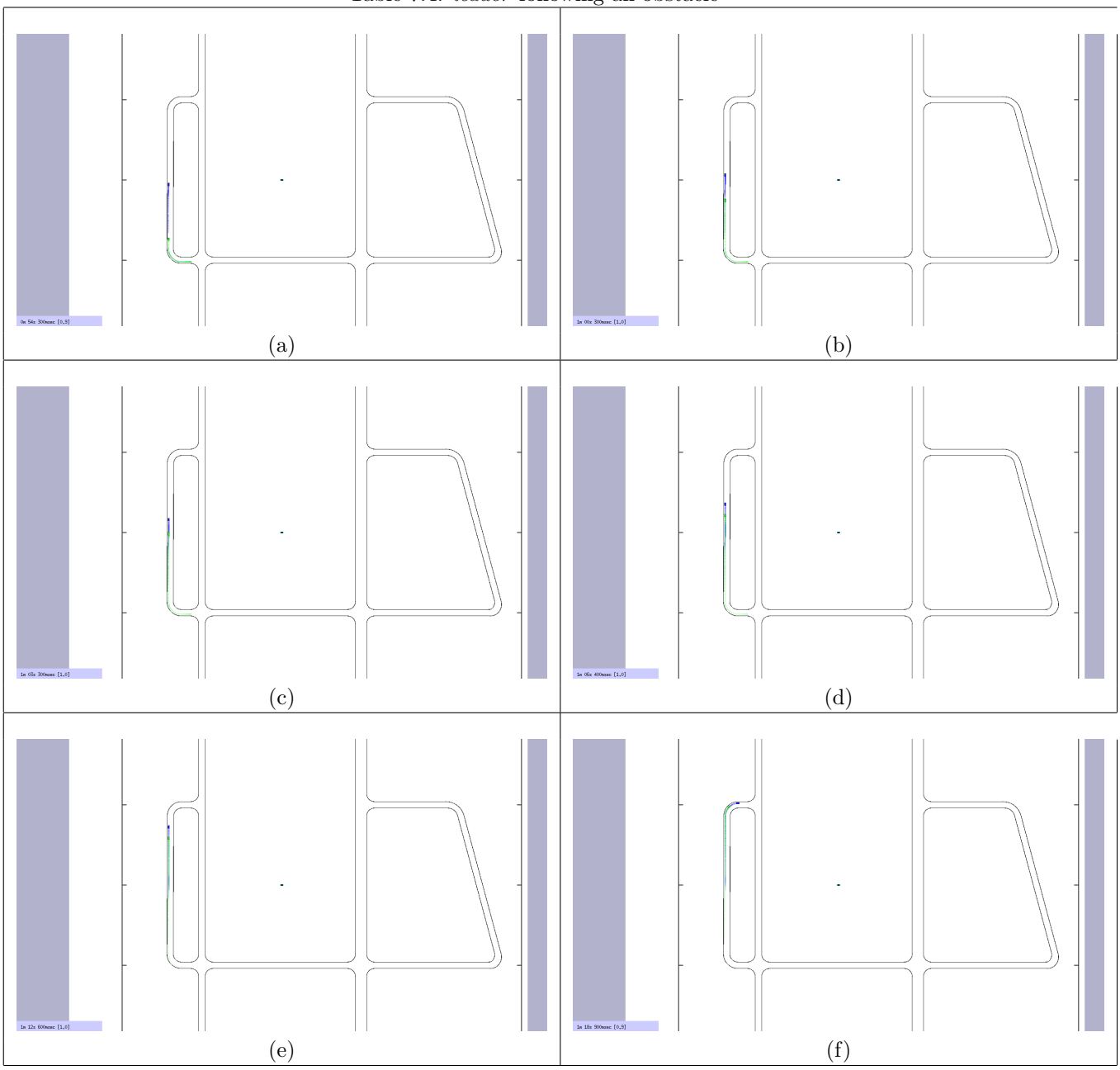


Figure 7.8: Profiles of different parameters

7.5.2 Follow or Overtake

Table 7.4: *leader* following an obstacle



(a) (b) (c) (d) (e) (f)

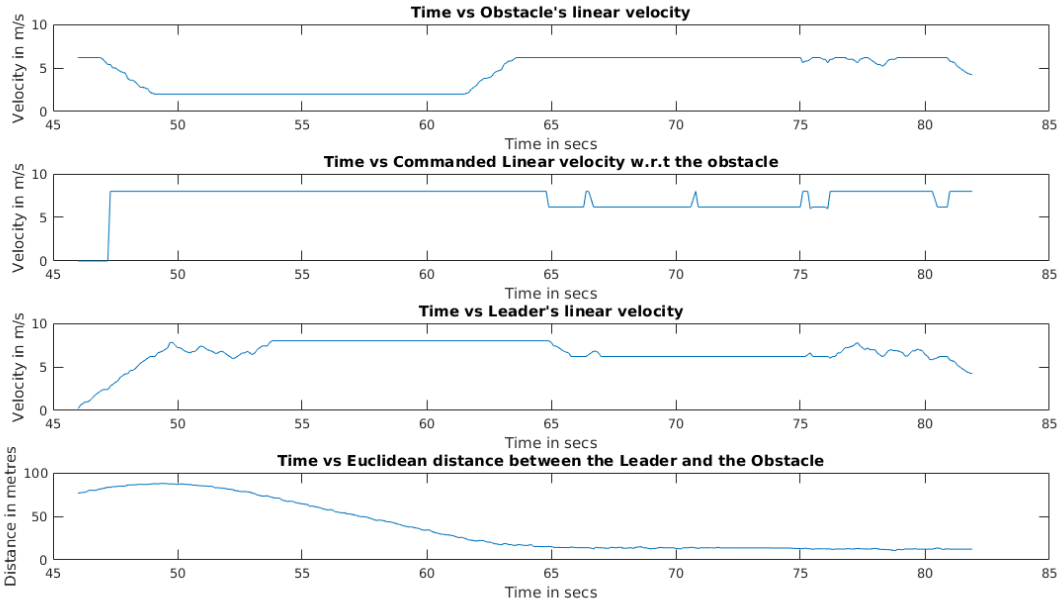
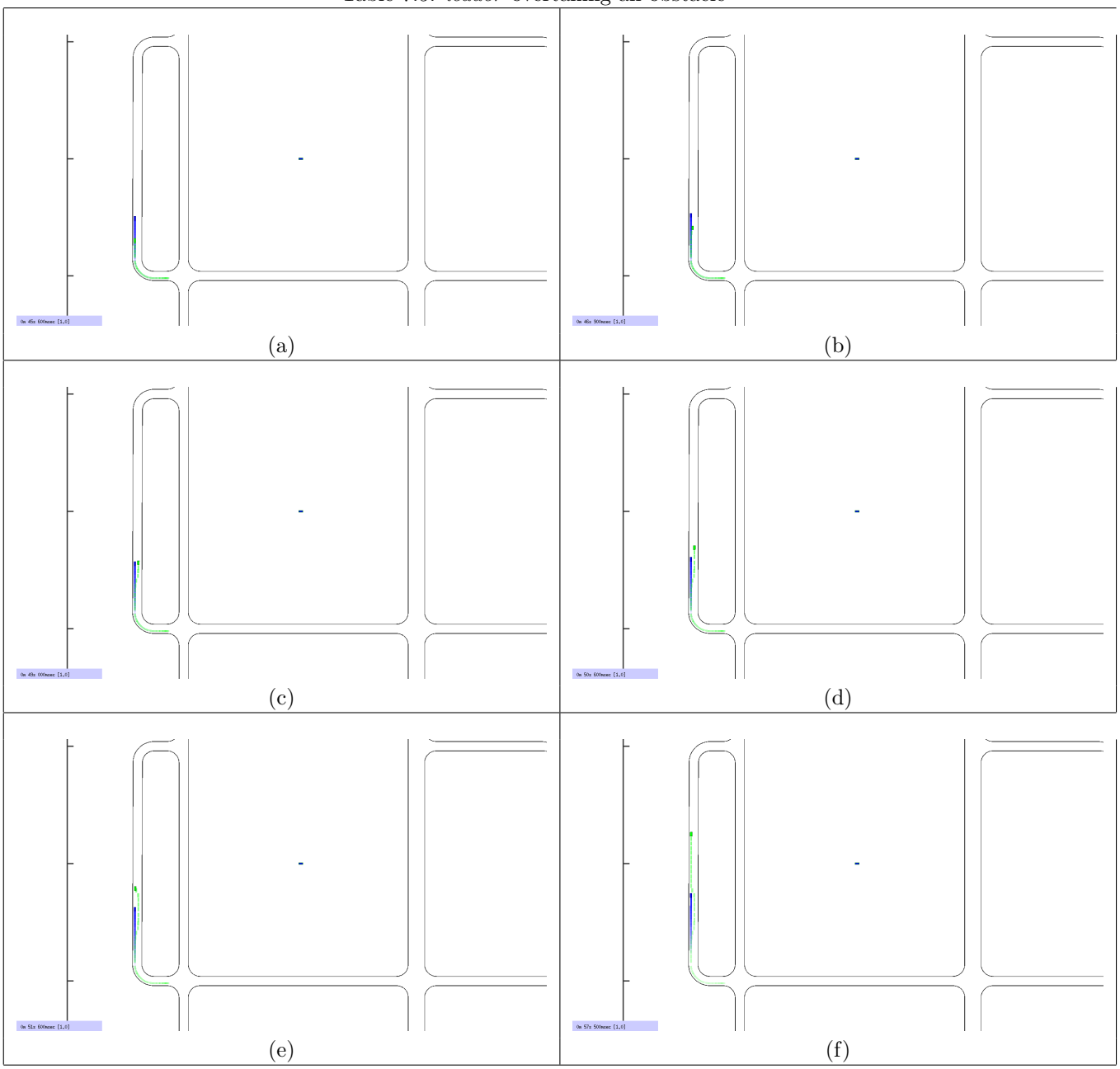


Figure 7.9: Profiles of different parameters

Table 7.5: *leader* overtaking an obstacle



(a) (b) (c) (d) (e) (f)

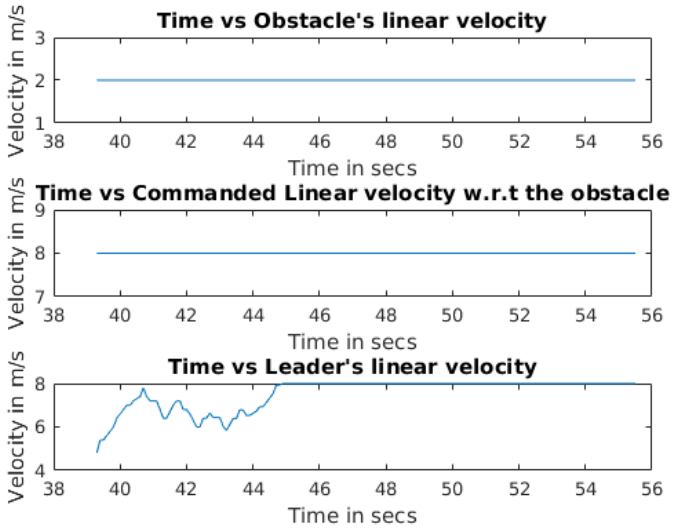


Figure 7.10: Profiles of different parameters

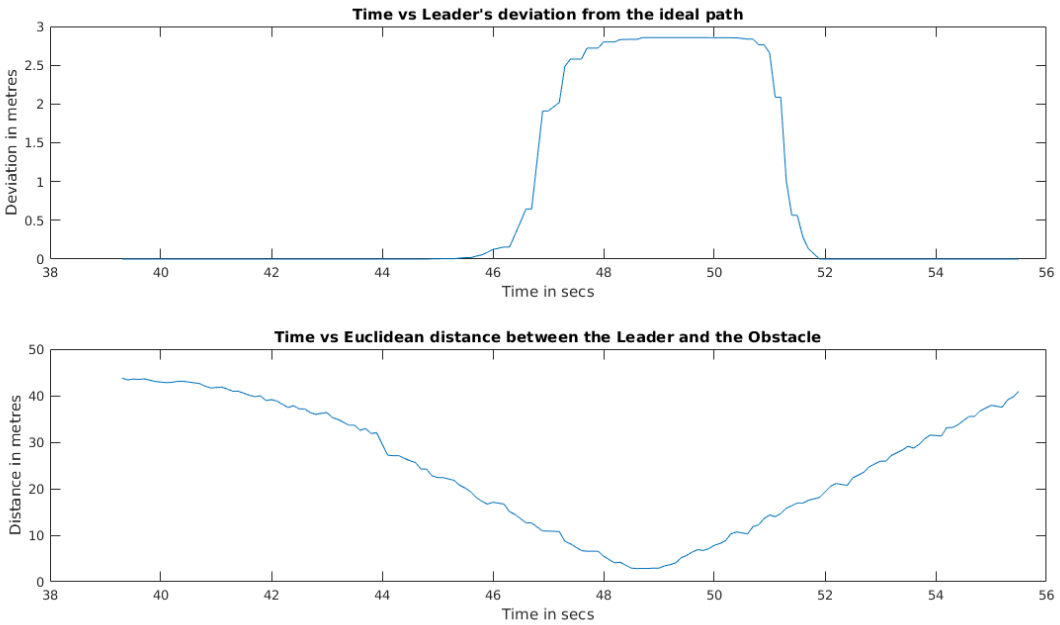
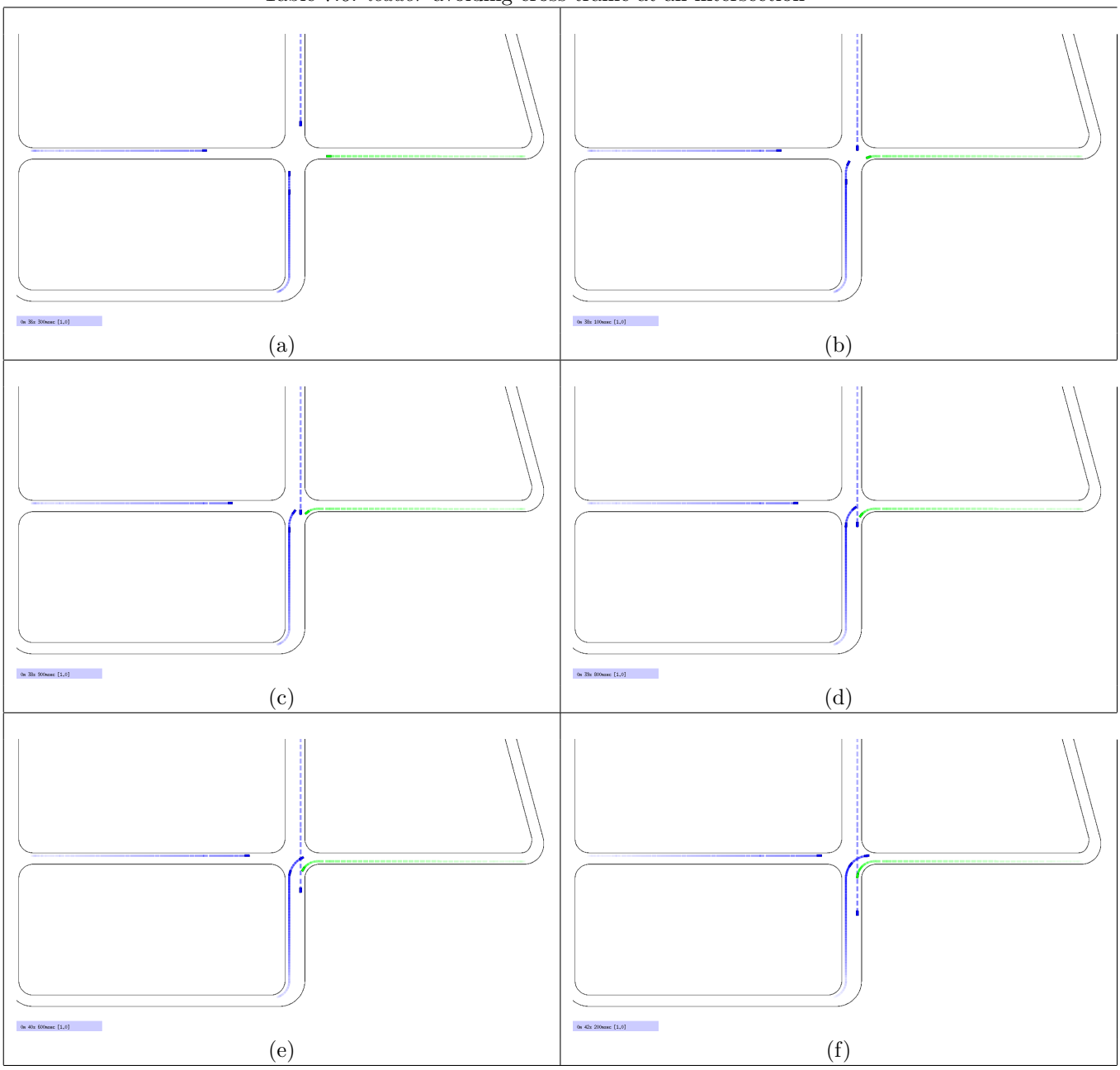


Figure 7.11: Profiles of different parameters

7.5.3 Intersection

Table 7.6: *leader* avoiding cross-traffic at an intersection



(a) (b) (c) (d) (e) (f)

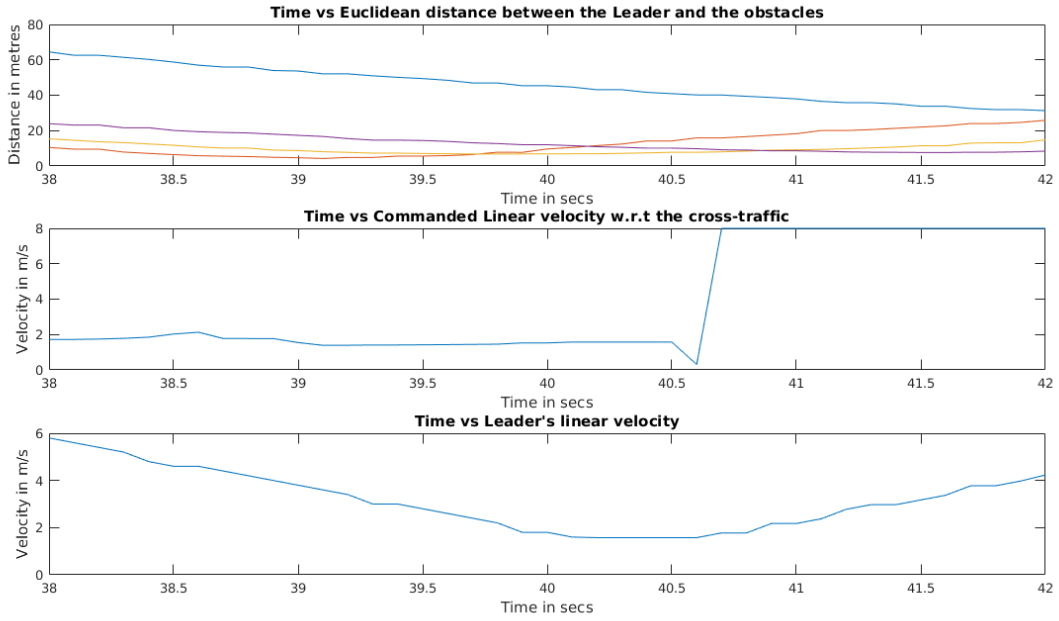


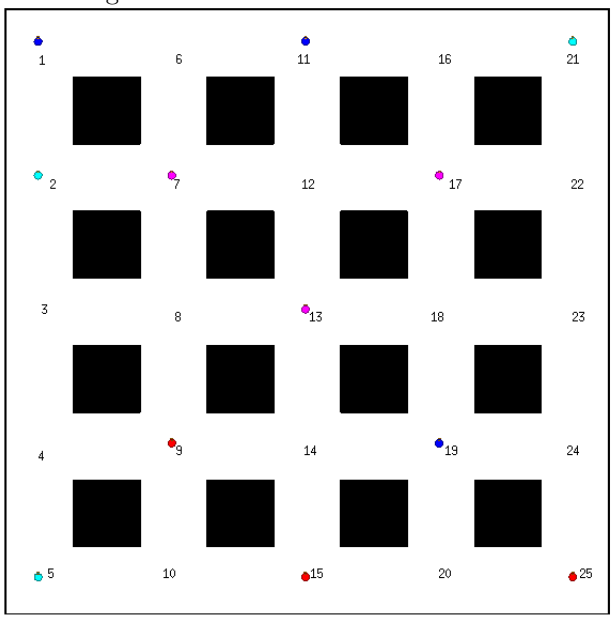
Figure 7.12: Profiles of different parameters

7.6 Multi Robot Patrolling Algorithms Comparison

7.6.1 Experimental Conditions

The algorithms were run on a Stage/ROS based package *patrolling sim* created by D. Portugal and R. Rocha [16]. It had a few of the standard indoor layouts, with a differential drive robot model and the benchmark patrolling strategies already coded in. Hence at this preliminary stage, the task was to carry out simulations with different parameters and analyse the results. The localization as well as motion planning was carried out using the ROS Navigation Stack standard features.

Figure 7.13: The grid used for simulation with various coloured robots



The layout shown above was used to conduct simulations of patrolling using various benchmark strategies using 8 differential drive robots for 20,000 seconds each.

7.6.2 Simulation Results

Random (RAND)

Figure 7.14: Instantaneous Idleness (in secs) vs Time (in secs)

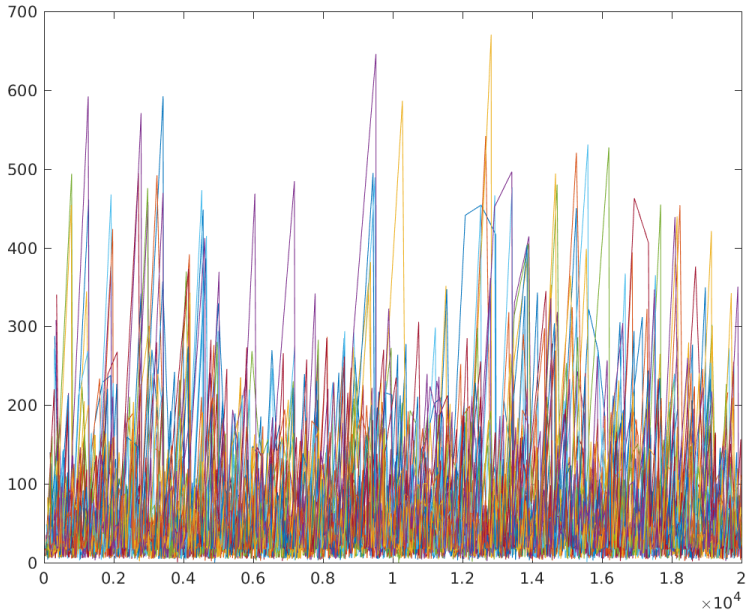
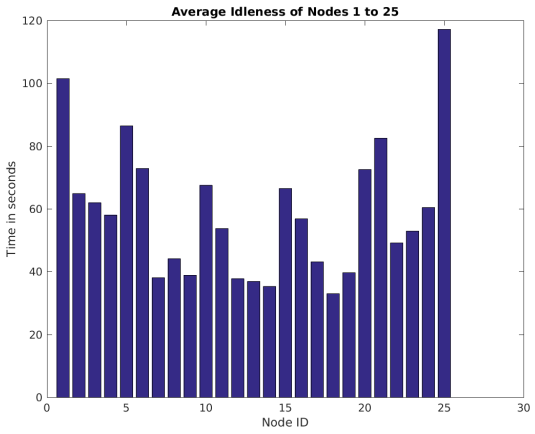


Table 7.7: Mean and Standard Deviation of different nodes



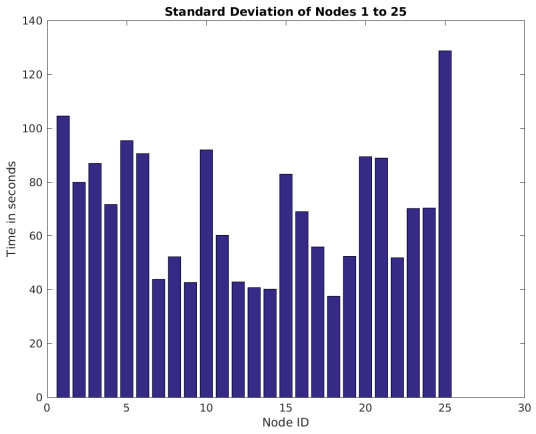


Figure 7.15: Histogram of Idleness values at different nodes

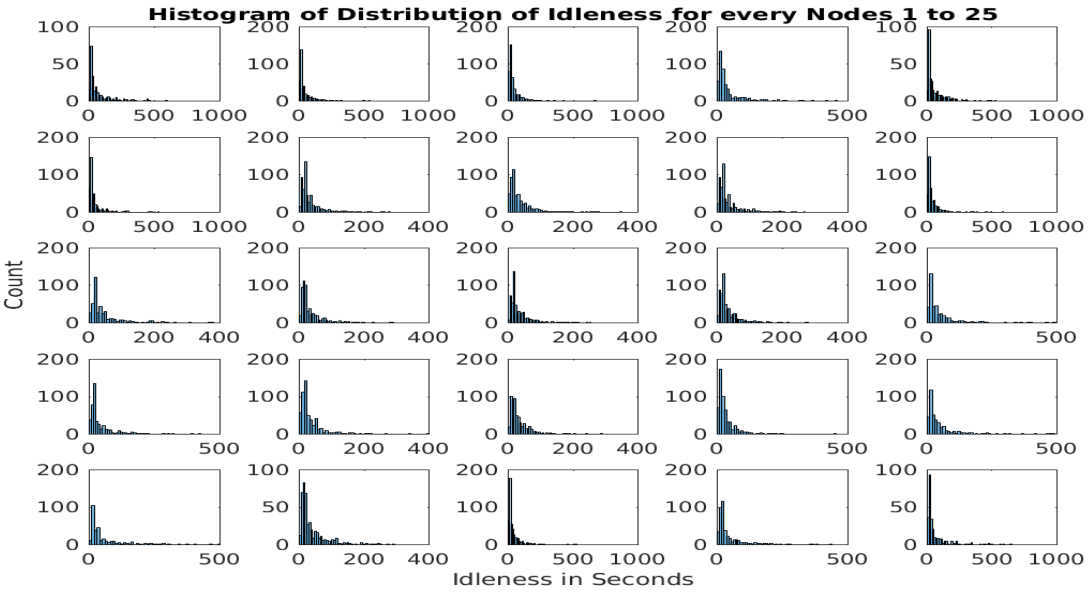
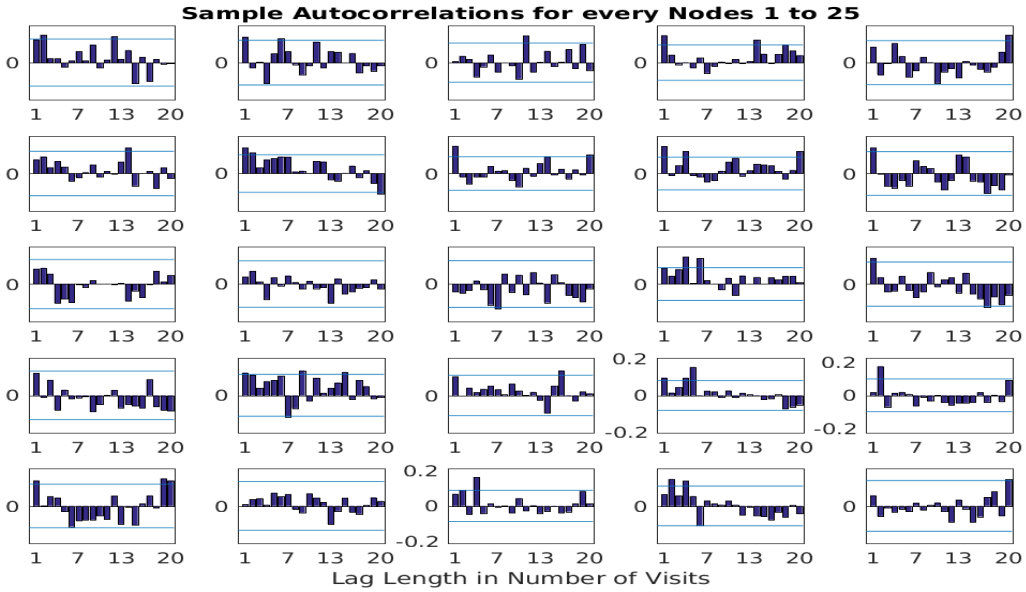


Figure 7.16: Correlation Analysis for different nodes



Conscientious Reactive (CR)

Figure 7.17: Instantaneous Idleness (in secs) vs Time (in secs)

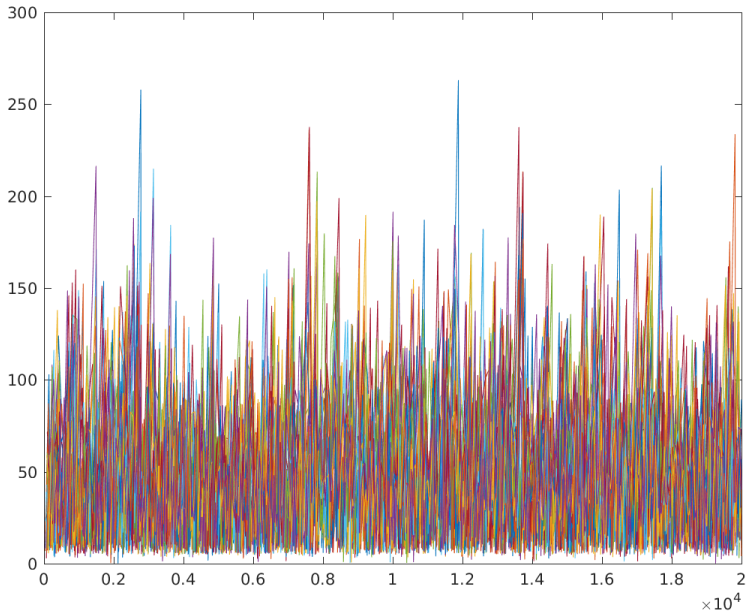
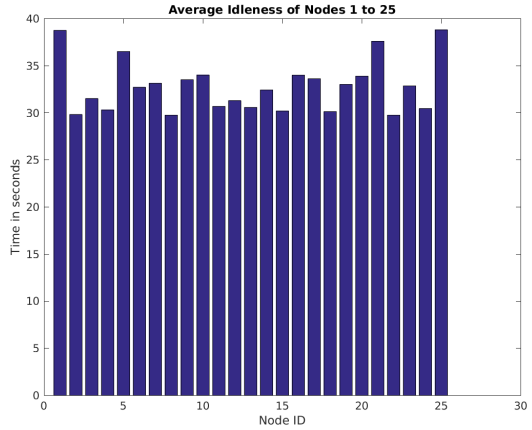


Table 7.8: Mean and Standard Deviation of different nodes



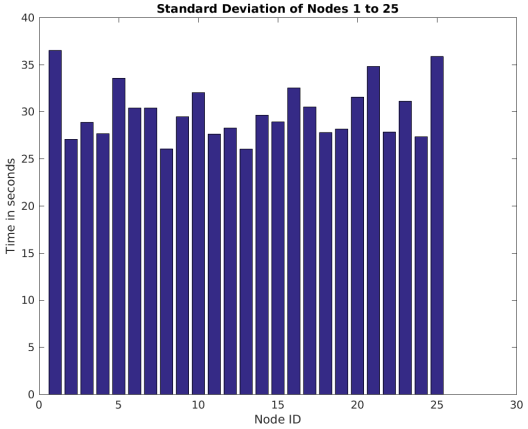


Figure 7.18: Histogram of Idleness values at different nodes

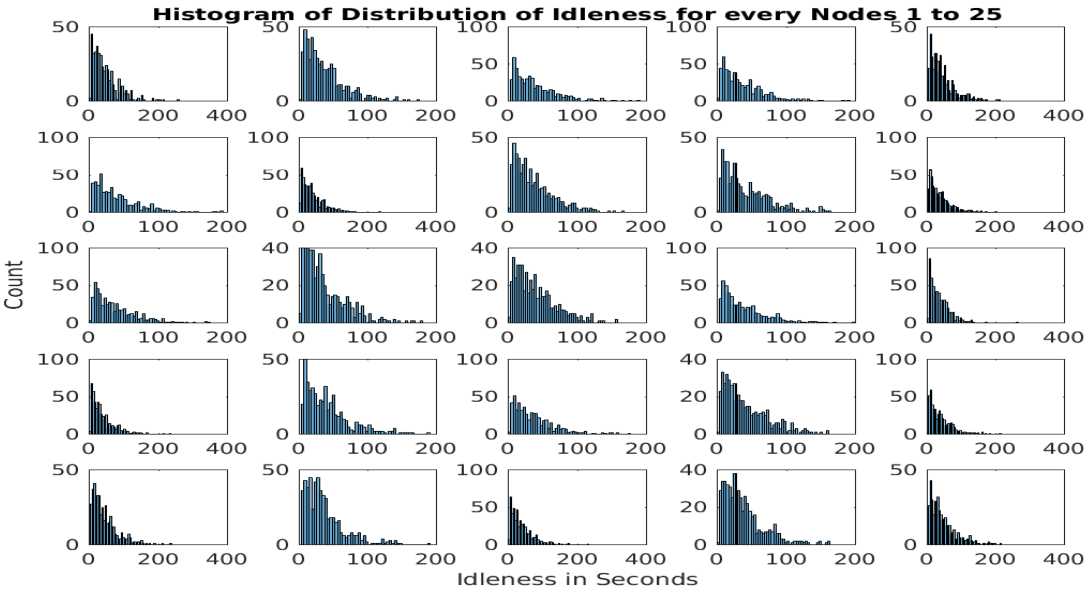
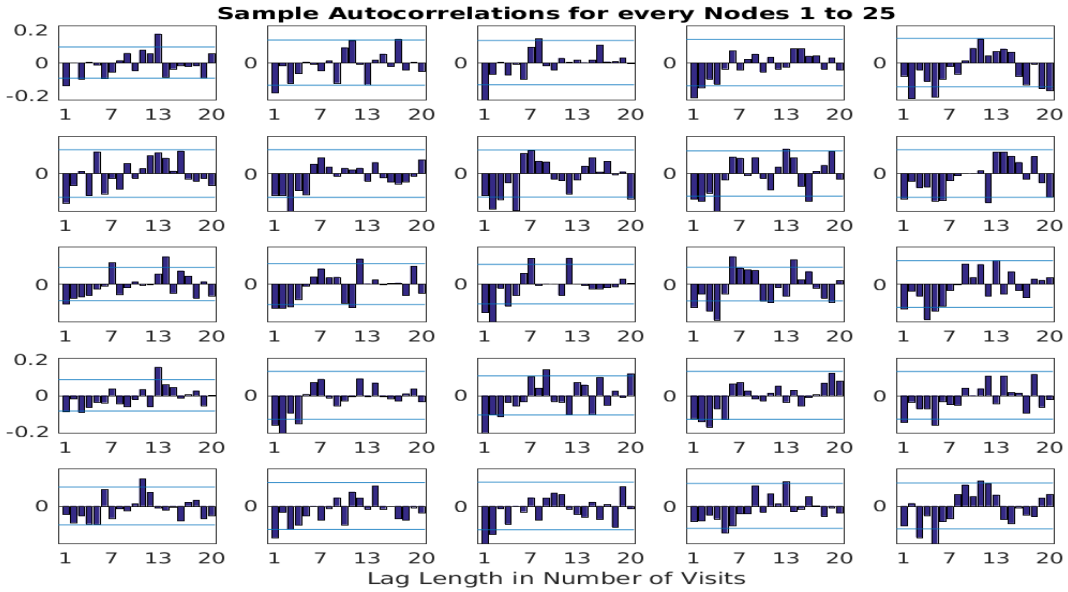


Figure 7.19: Correlation Analysis for different nodes



Heuristic Conscientious Reactive (HCR)

Figure 7.20: Instantaneous Idleness (in secs) vs Time (in secs)

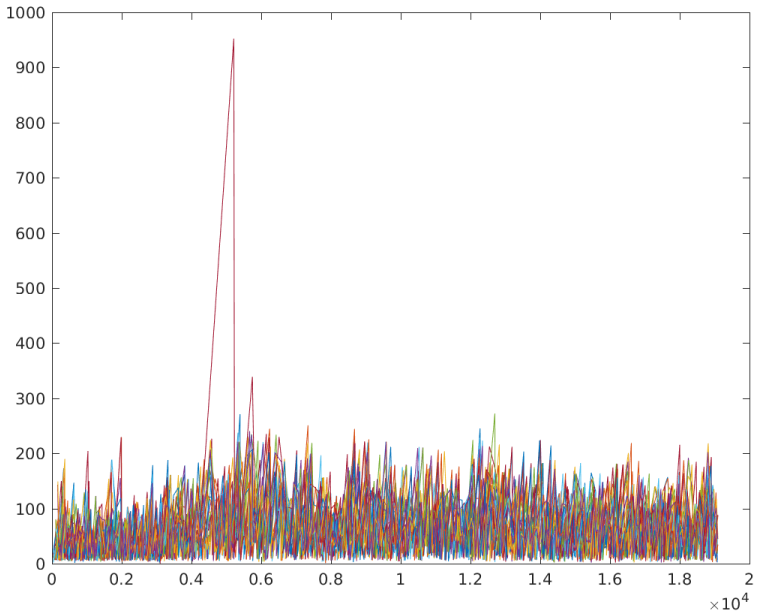
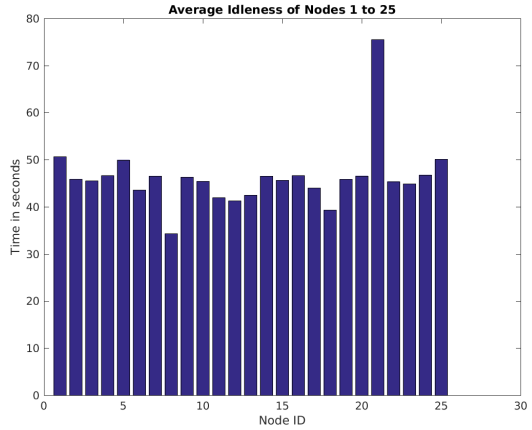


Table 7.9: Mean and Standard Deviation of different nodes



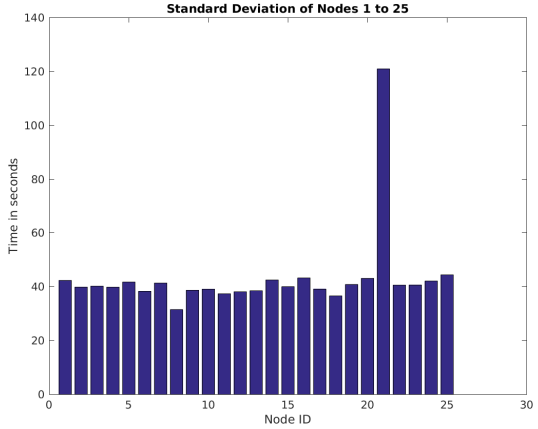


Figure 7.21: Histogram of Idleness values at different nodes

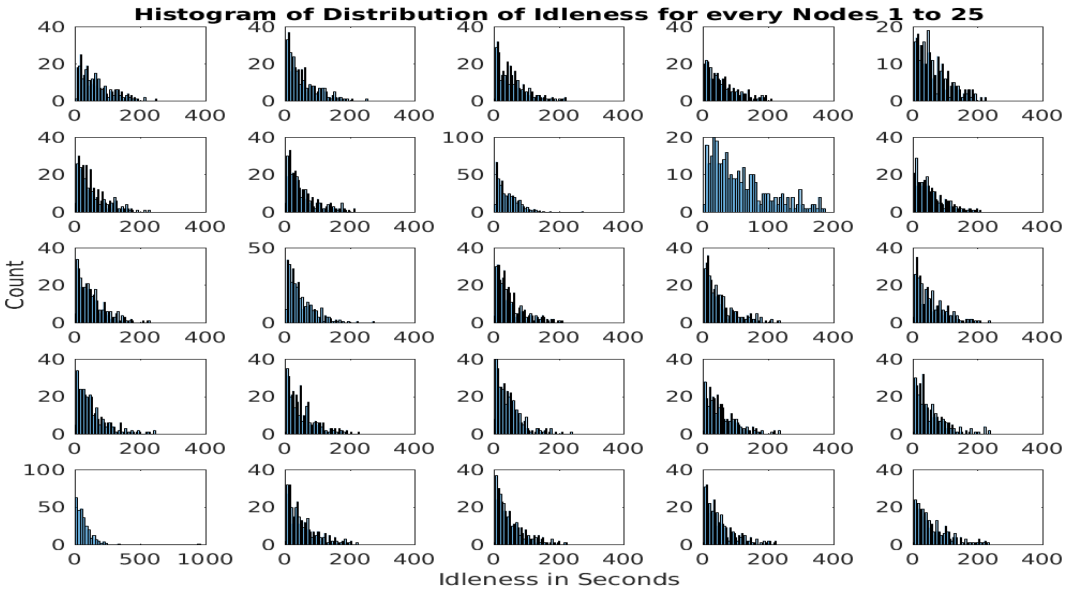
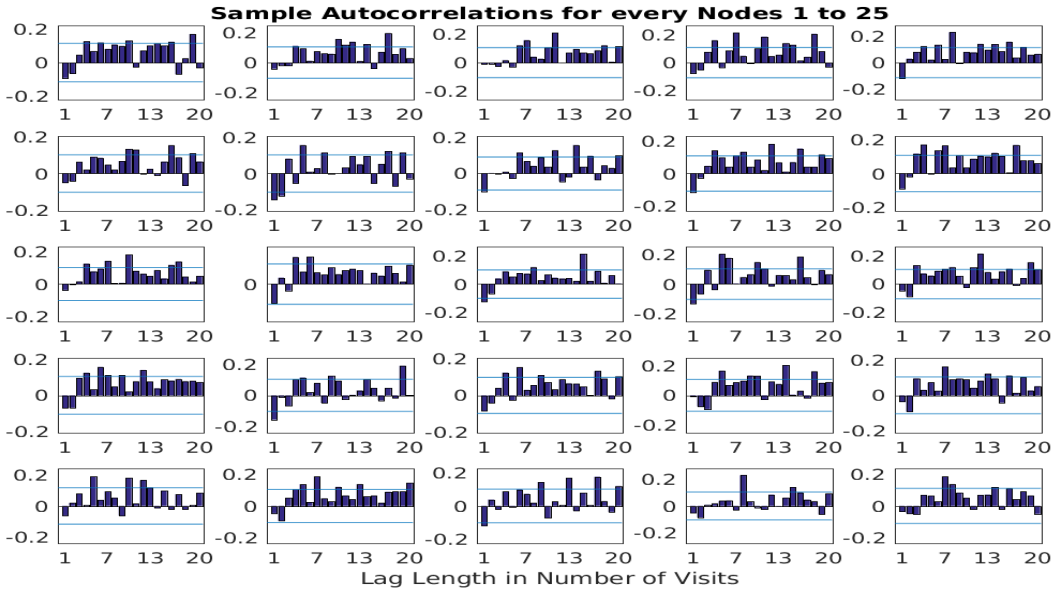


Figure 7.22: Correlation Analysis for different nodes



Heuristic Pathfinder Conscientious Cognitive (HPCC)

Figure 7.23: Instantaneous Idleness (in secs) vs Time (in secs)

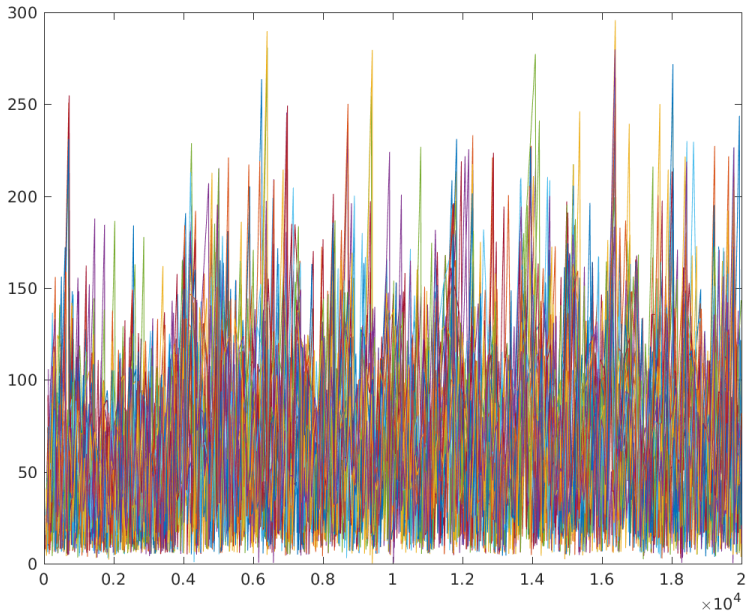
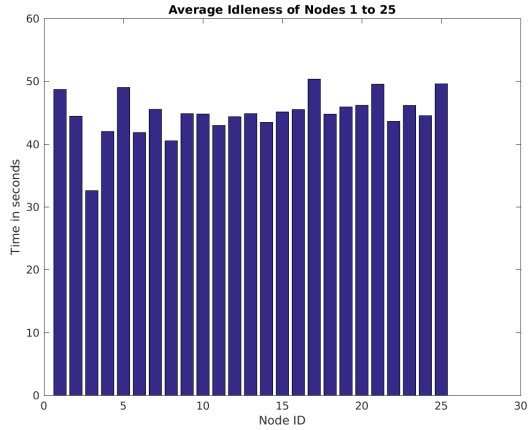


Table 7.10: Mean and Standard Deviation of different nodes



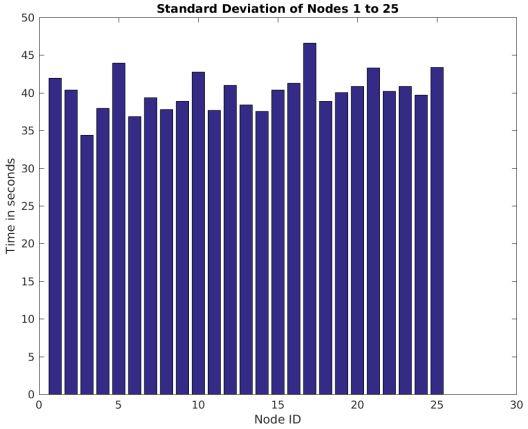


Figure 7.24: Histogram of Idleness values at different nodes

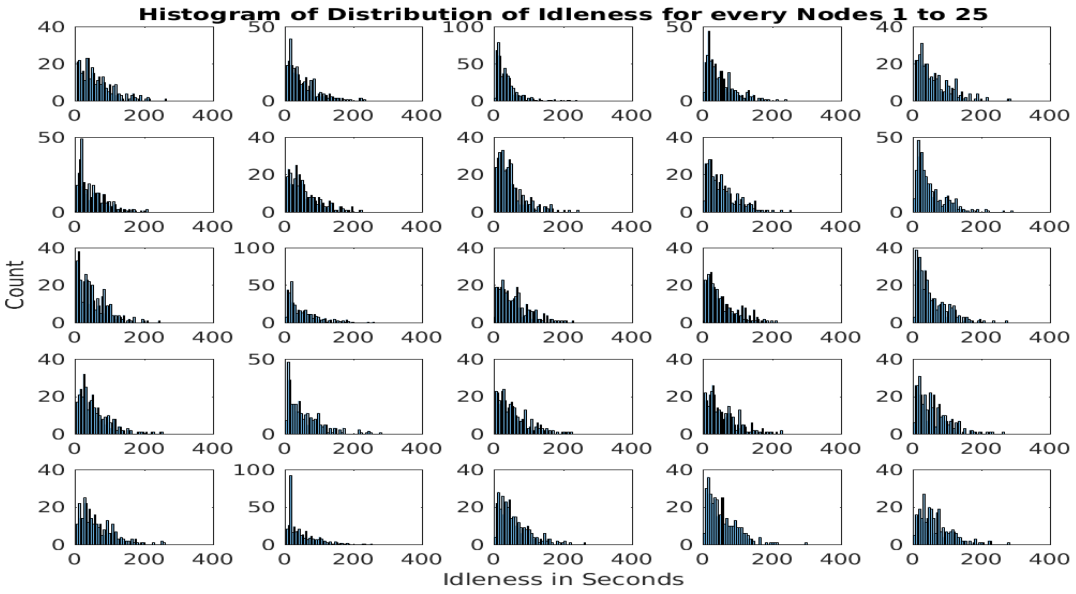


Figure 7.25: Correlation Analysis for different nodes

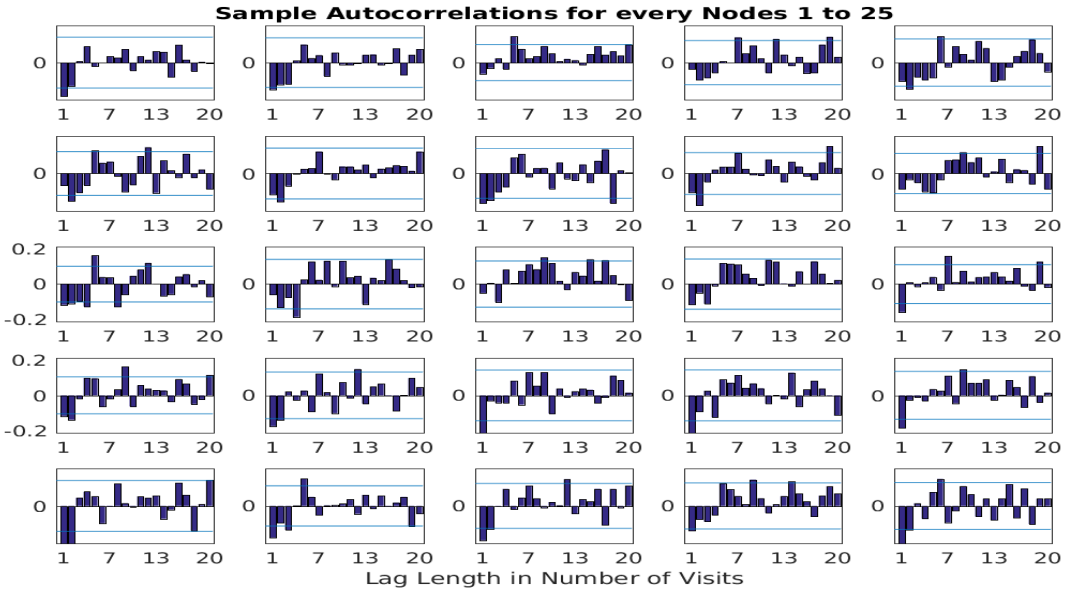


Figure 7.26: Instantaneous Idleness (in secs) vs Time (in secs)

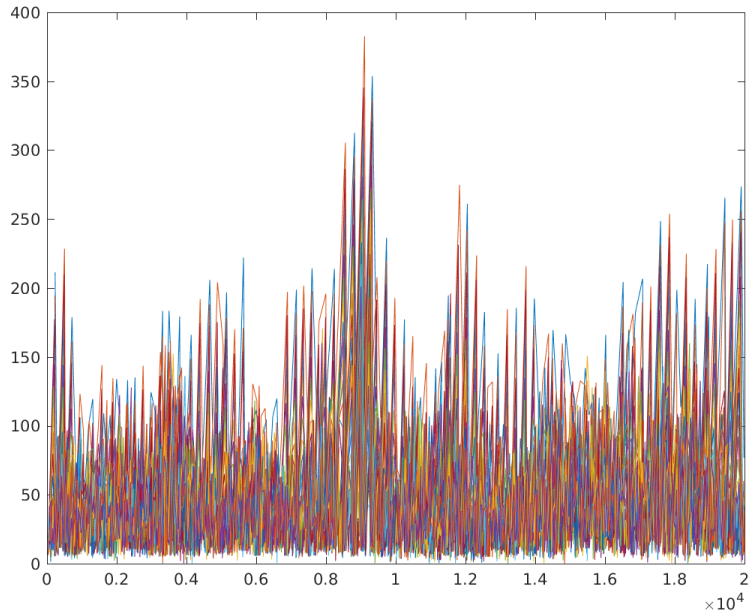
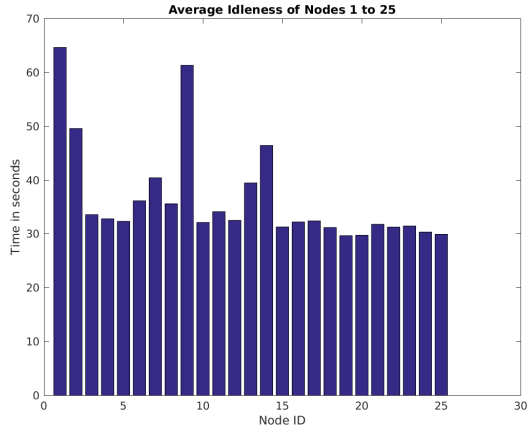


Table 7.11: Mean and Standard Deviation of different nodes



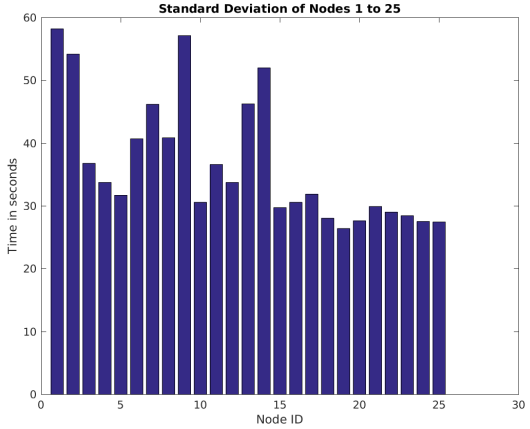


Figure 7.27: Histogram of Idleness values at different nodes

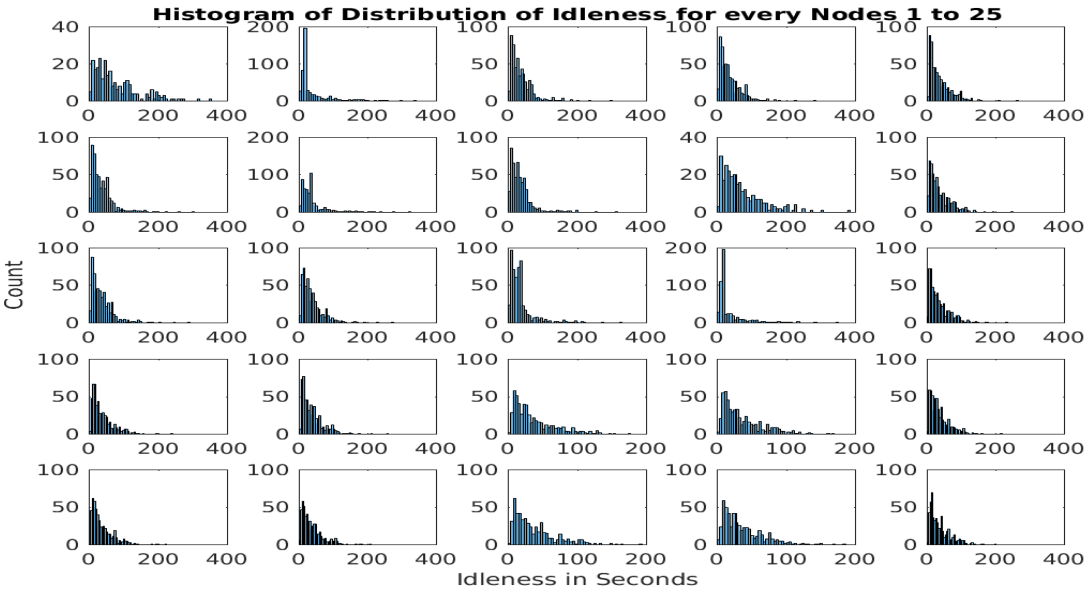


Figure 7.28: Correlation Analysis for different nodes

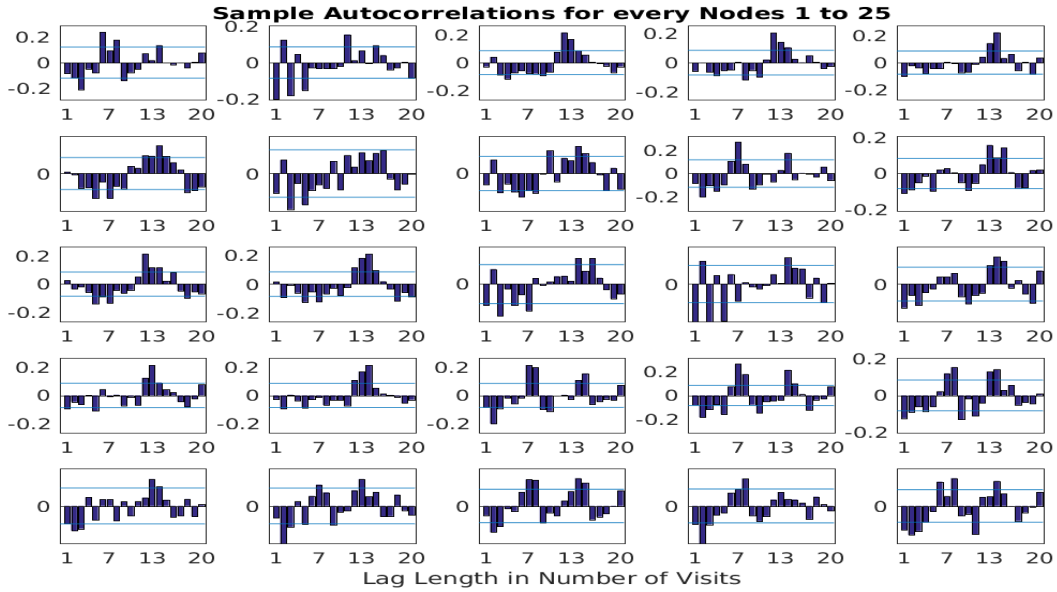


Figure 7.29: Instantaneous Idleness (in secs) vs Time (in secs)

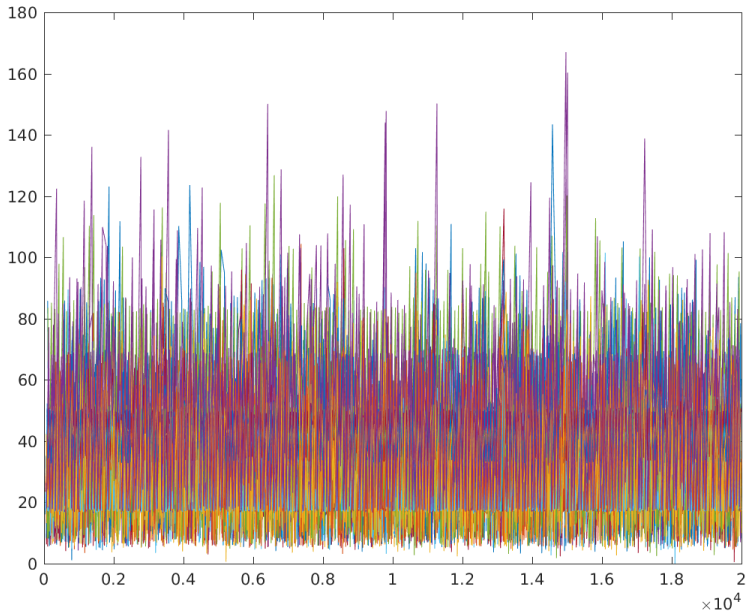
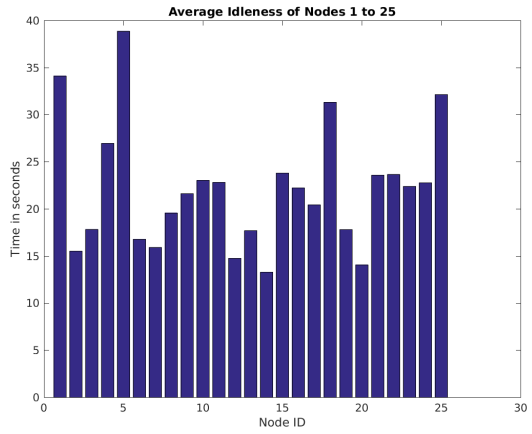


Table 7.12: Mean and Standard Deviation of different nodes



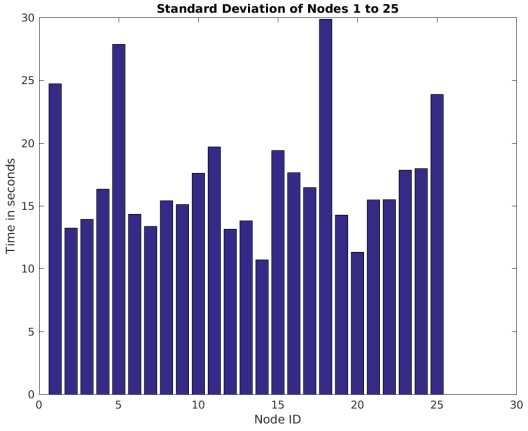


Figure 7.30: Histogram of Idleness values at different nodes

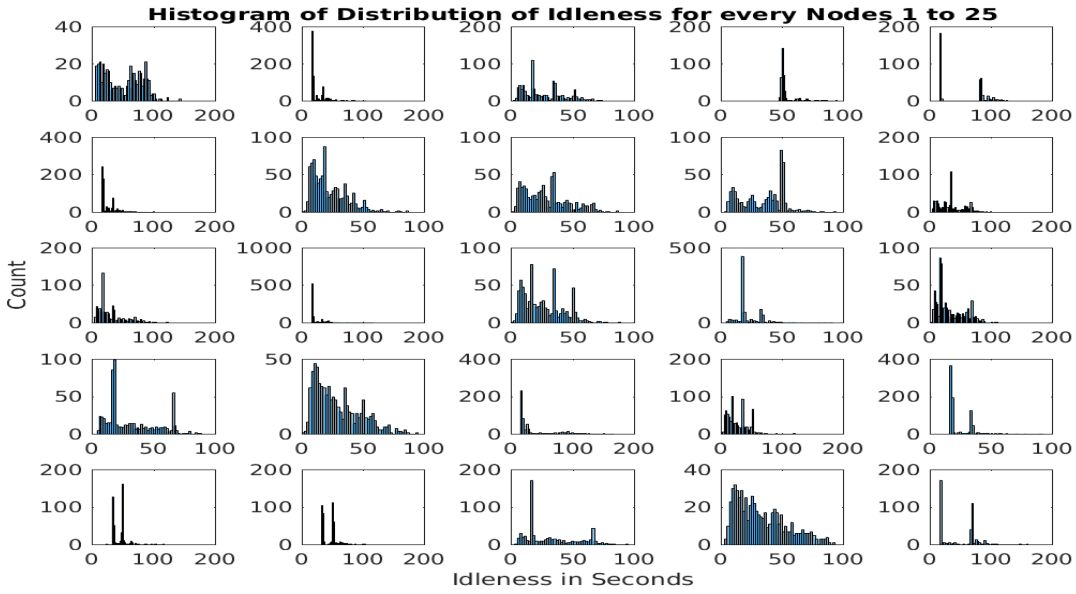
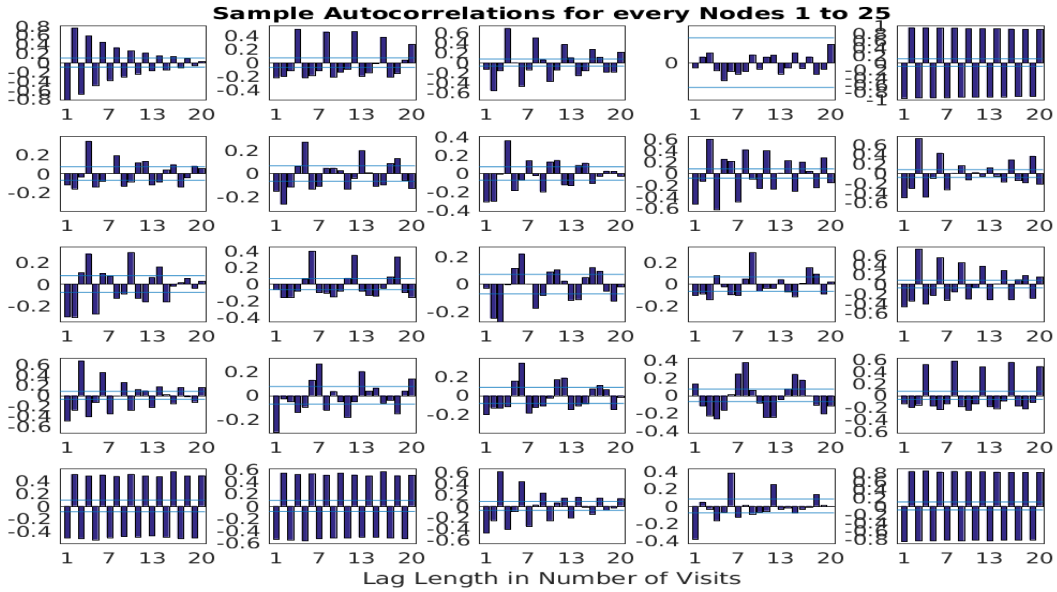


Figure 7.31: Correlation Analysis for different nodes



7.6.3 Greedy Bayesian Strategy (GBS)

Figure 7.32: Instantaneous Idleness (in secs) vs Time (in secs)

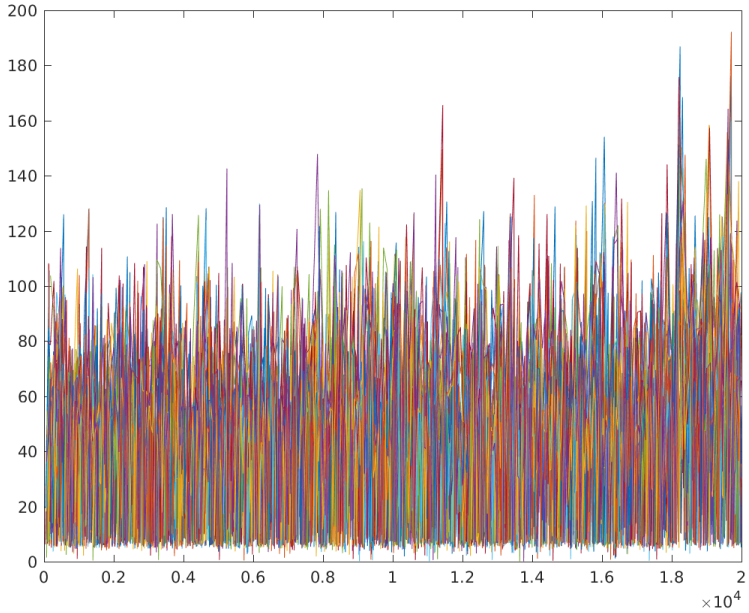
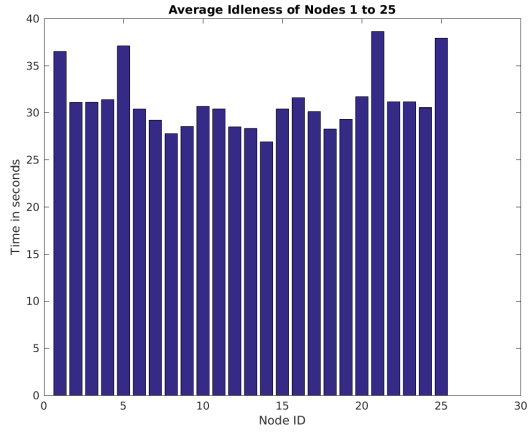


Table 7.13: Mean and Standard Deviation of different nodes



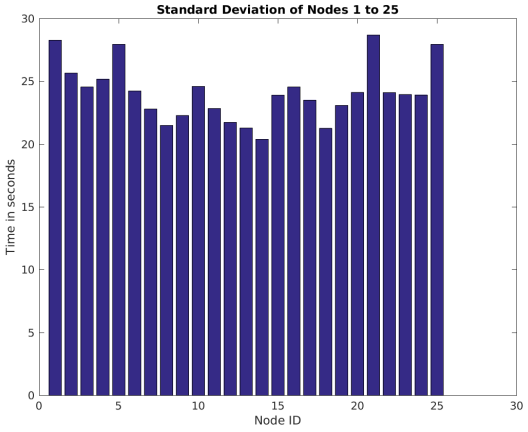


Figure 7.33: Histogram of Idleness values at different nodes

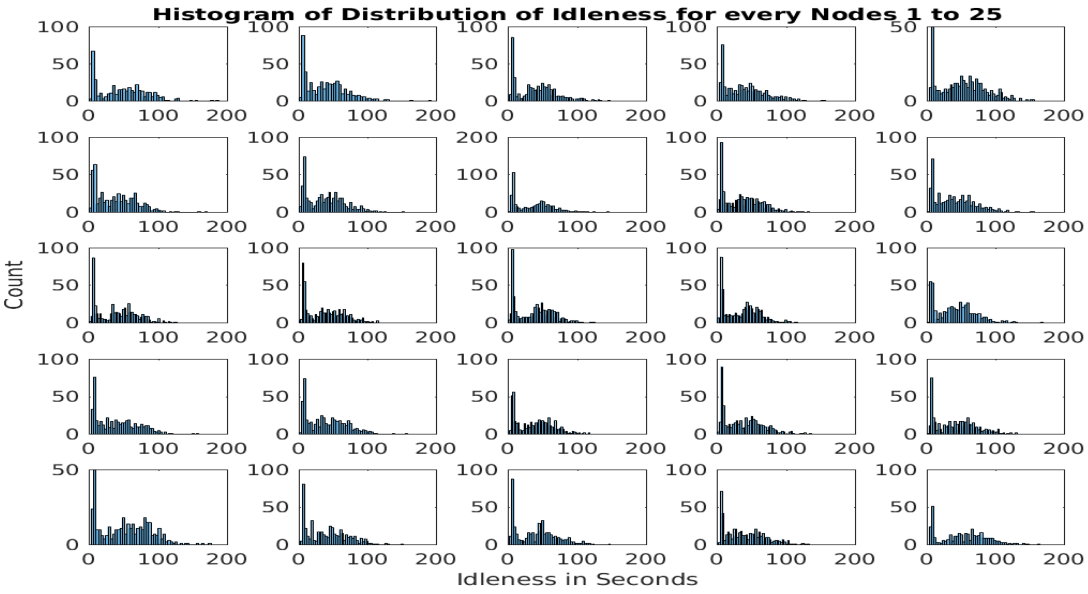
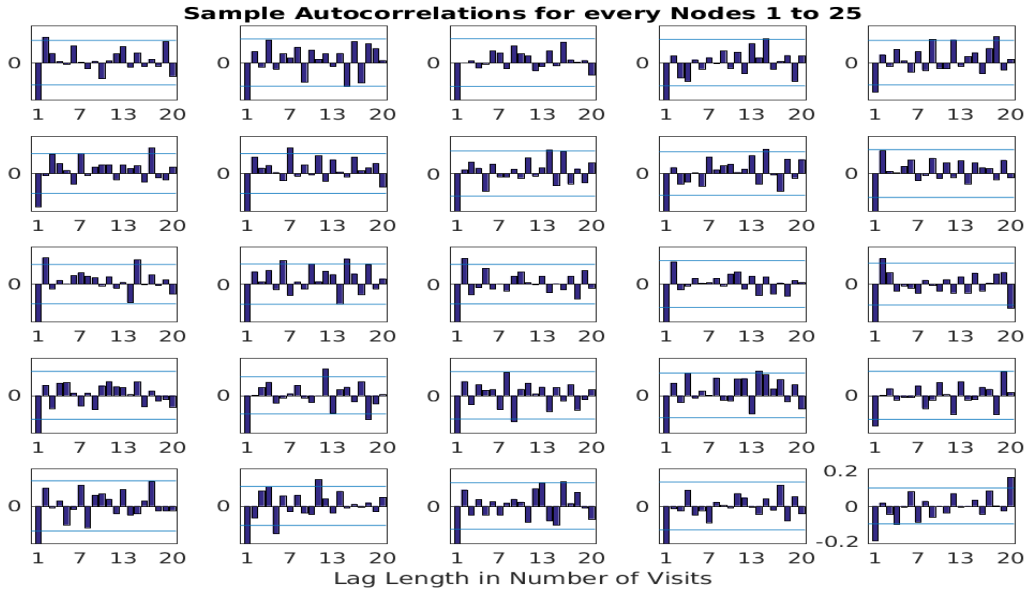


Figure 7.34: Correlation Analysis for different nodes



State Exchange Bayesian Strategy (SEBS)

Figure 7.35: Instantaneous Idleness (in secs) vs Time (in secs)

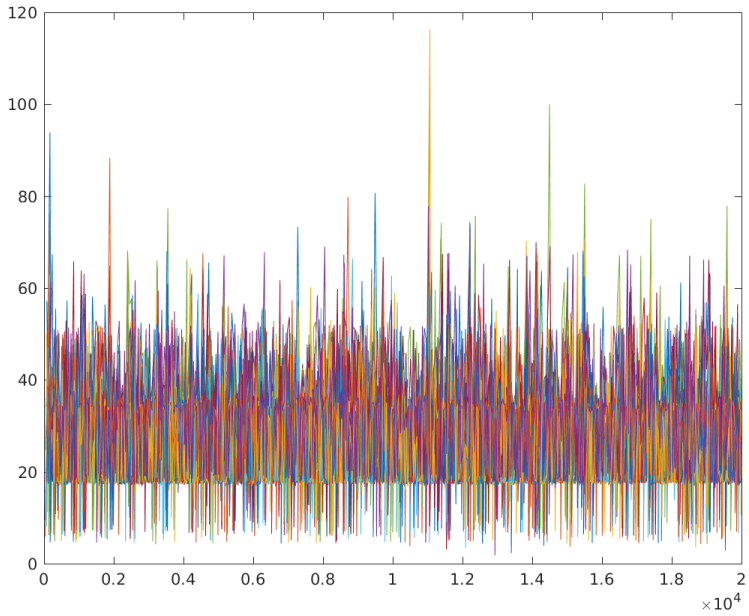
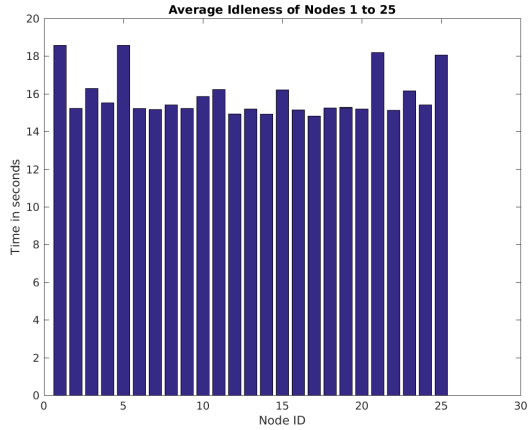


Table 7.14: Mean and Standard Deviation of different nodes



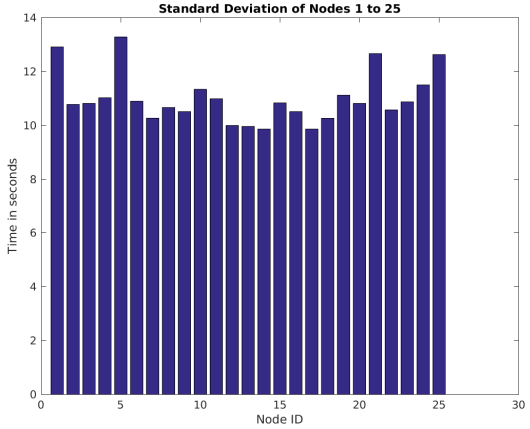


Figure 7.36: Histogram of Idleness values at different nodes

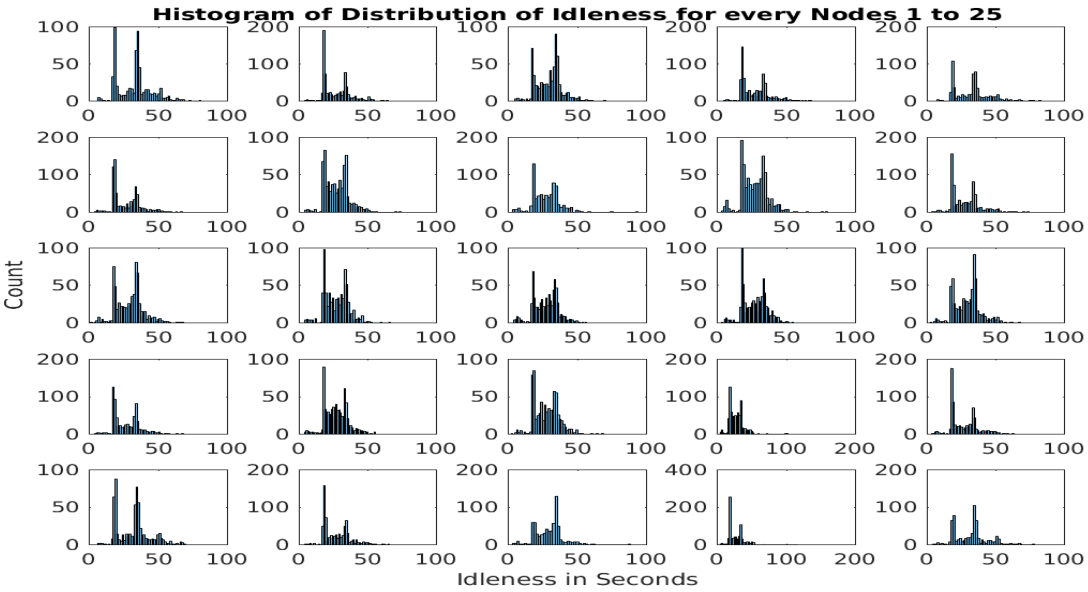


Figure 7.37: Correlation Analysis for different nodes

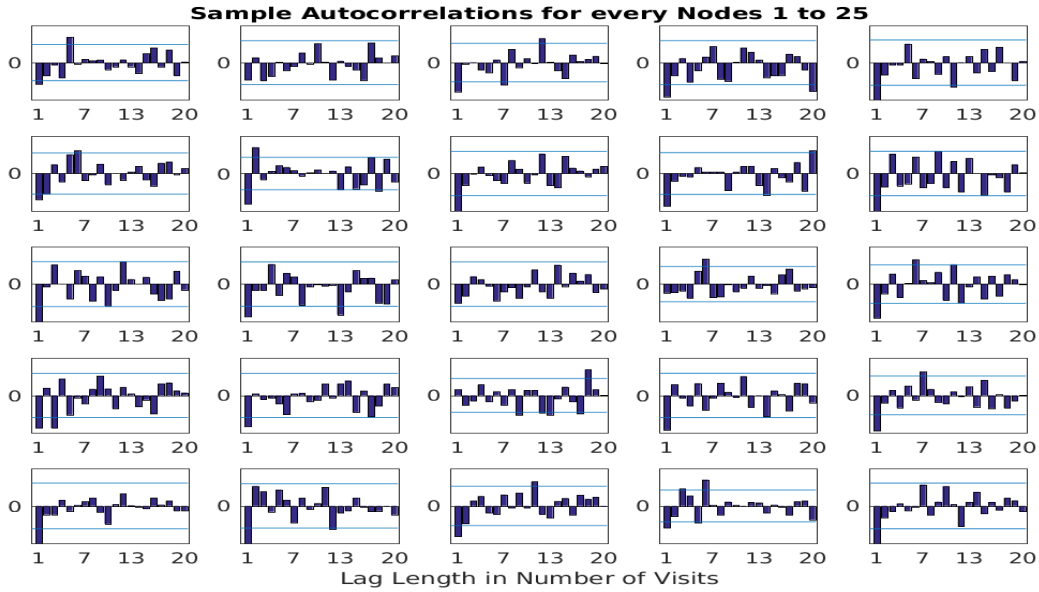


Figure 7.38: Instantaneous Idleness (in secs) vs Time (in secs)

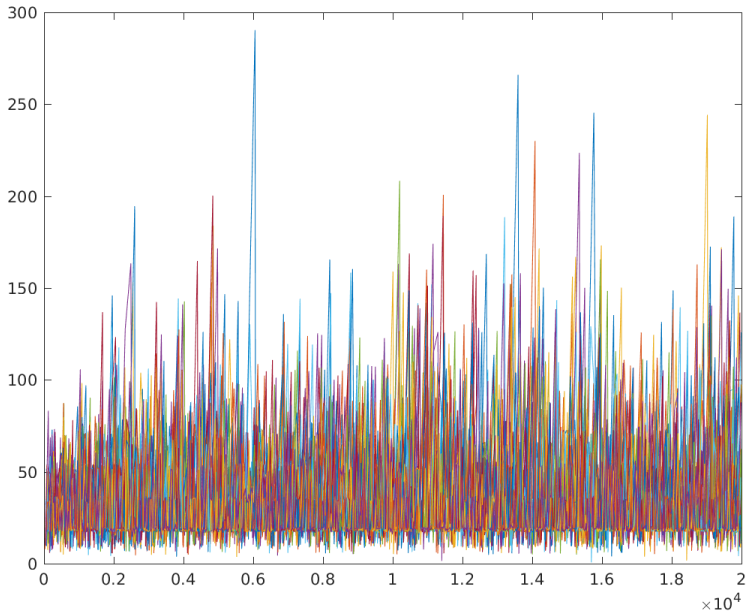
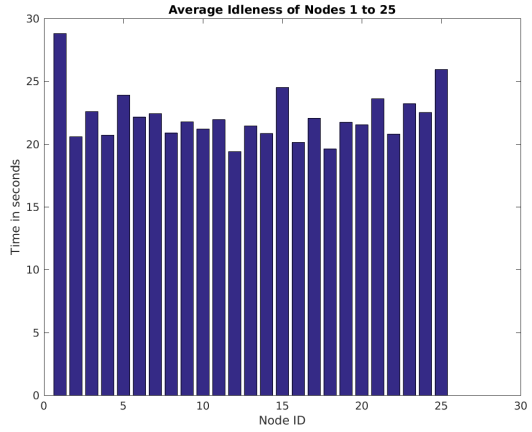


Table 7.15: Mean and Standard Deviation of different nodes



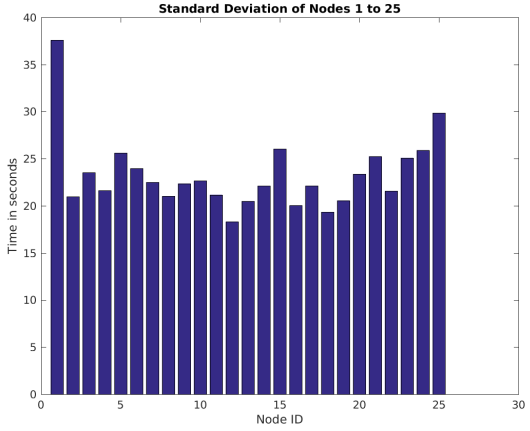


Figure 7.39: Histogram of Idleness values at different nodes

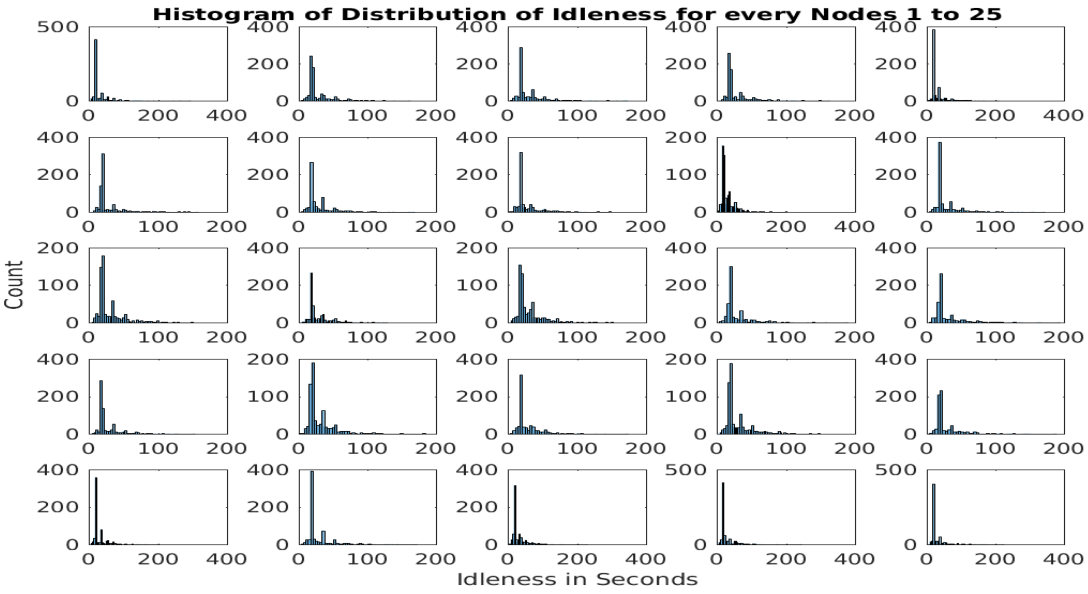
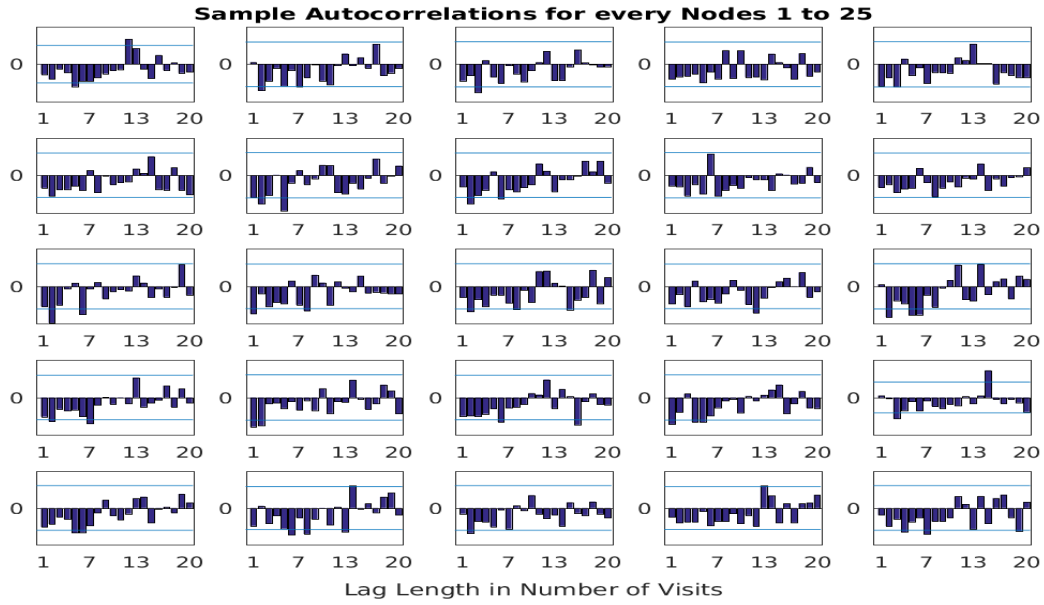


Figure 7.40: Correlation Analysis for different nodes



Dynamic Task Assignment Greedy (DTAG)

Figure 7.41: Instantaneous Idleness (in secs) vs Time (in secs)

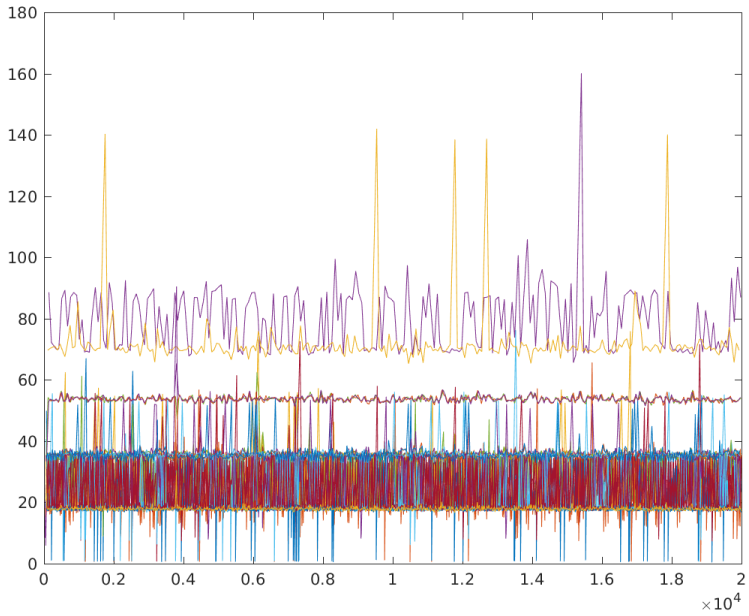
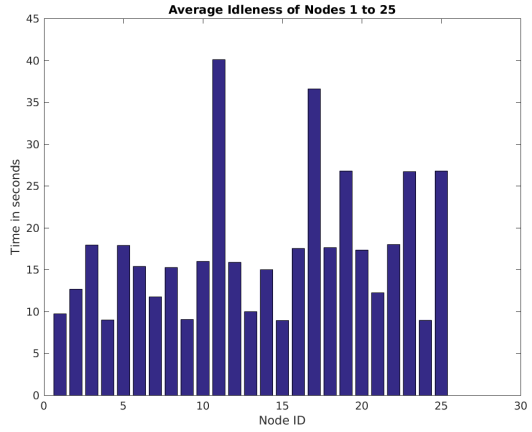


Table 7.16: Mean and Standard Deviation of different nodes



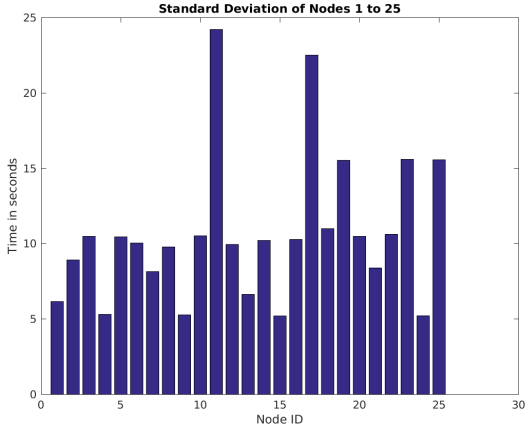


Figure 7.42: Histogram of Idleness values at different nodes

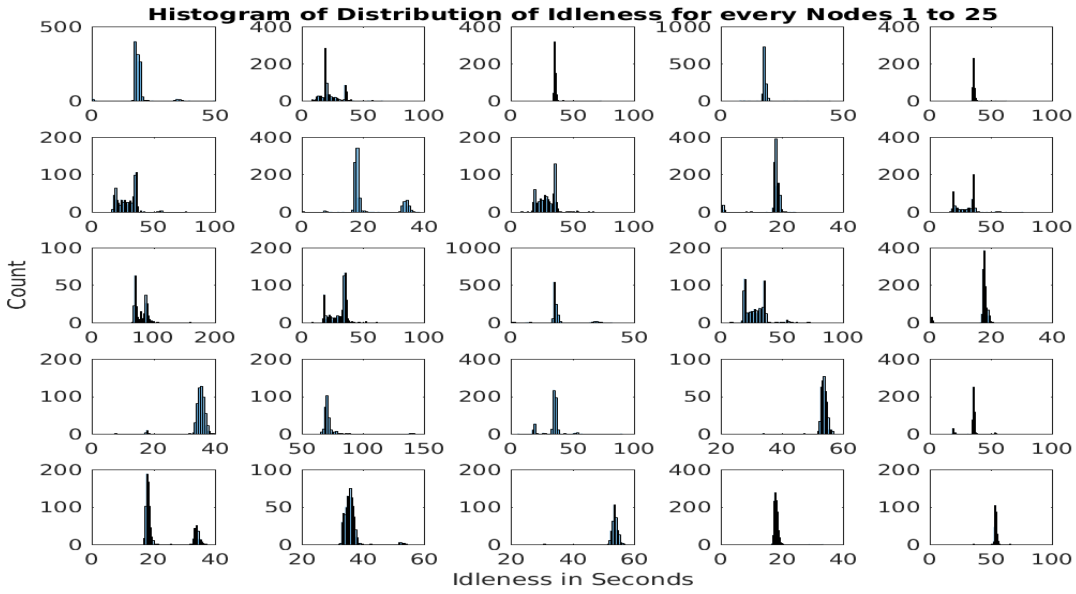


Figure 7.43: Correlation Analysis for different nodes

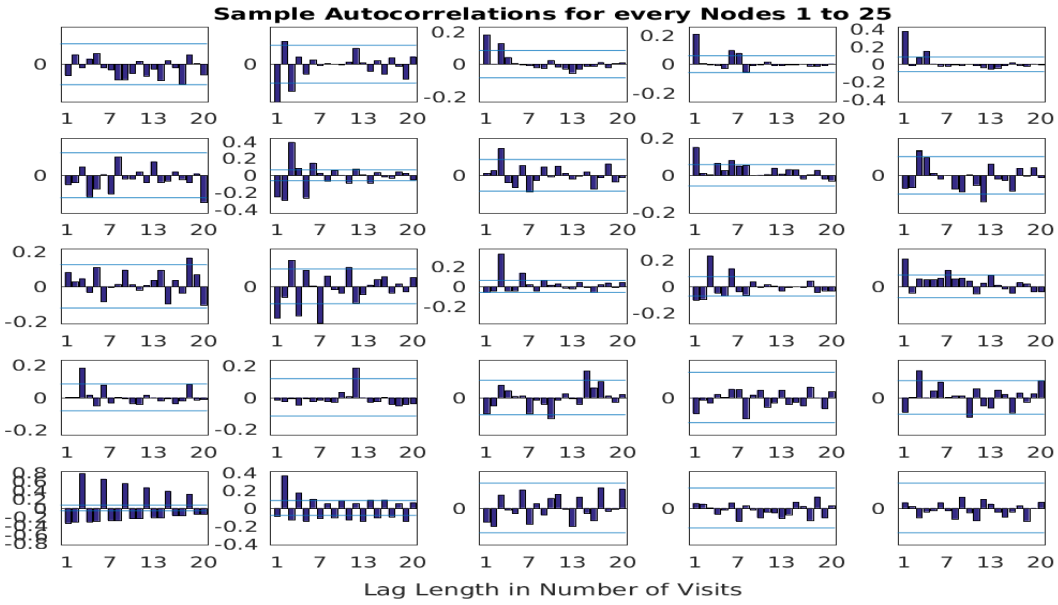


Figure 7.44: Instantaneous Idleness (in secs) vs Time (in secs)

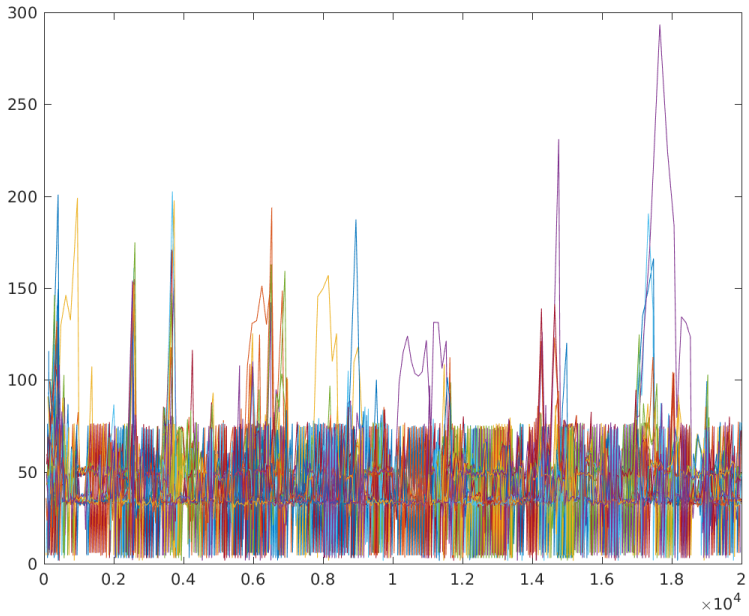
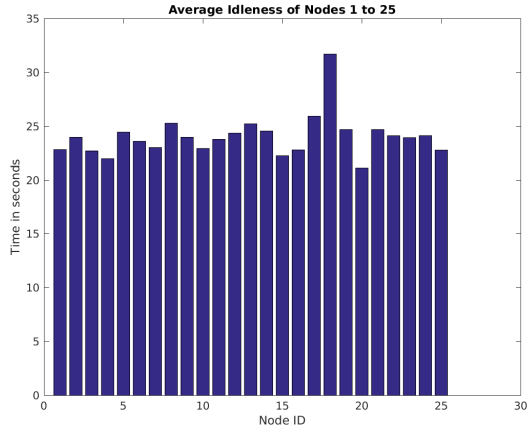


Table 7.17: Mean and Standard Deviation of different nodes



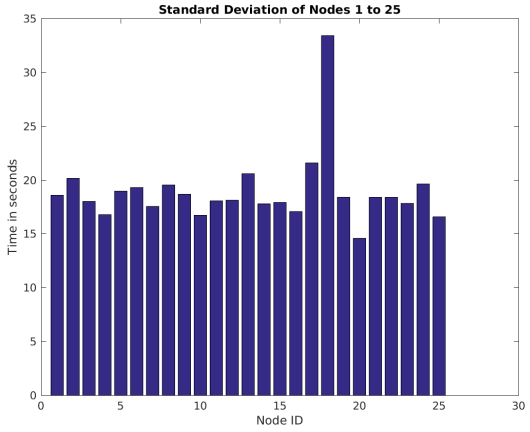


Figure 7.45: Histogram of Idleness values at different nodes

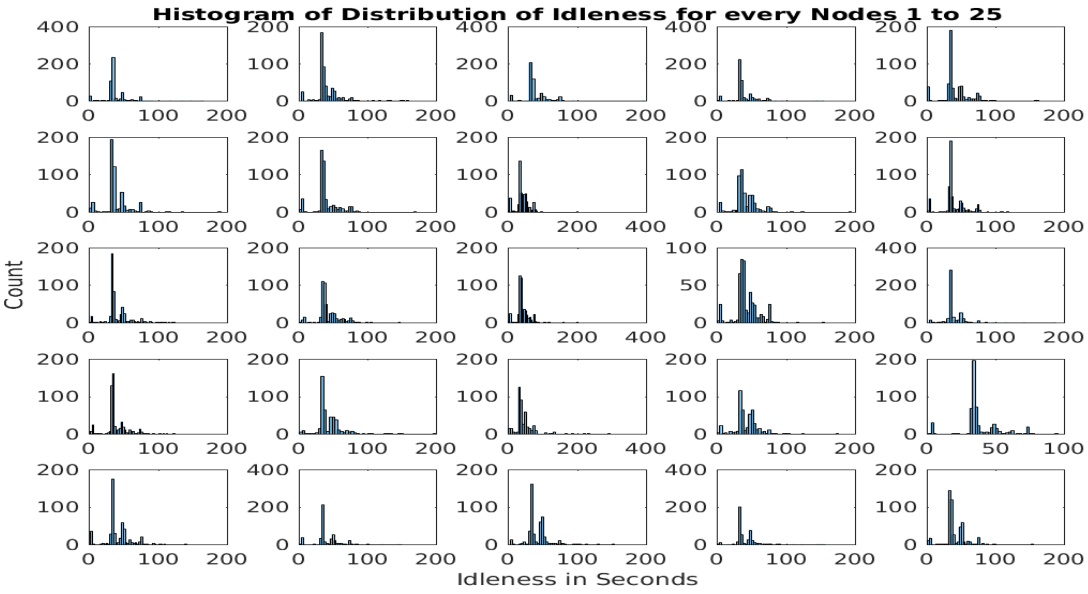
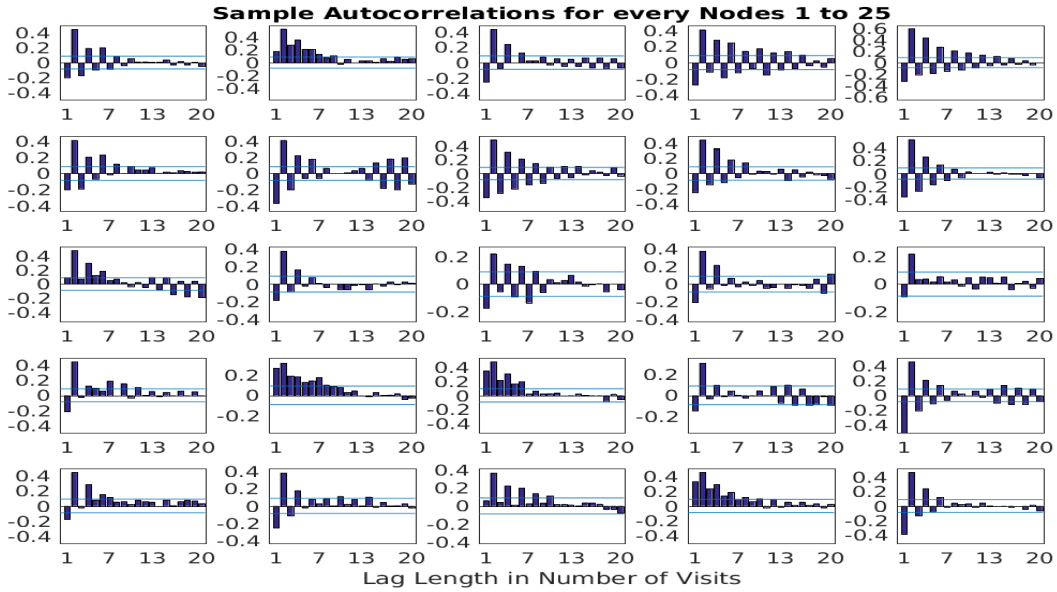


Figure 7.46: Correlation Analysis for different nodes



7.6.4 Comparisons

Table 7.18: Comparison of Mean and Standard Deviations of all the algorithms

Algorithm	Global Idleness Mean	Global Idleness Standard Deviation
RAND	58.90	69.67
CR	32.77	30.00
HCR	46.32	43.24
HPCC	44.86	40.18
CGG	36.49	36.62
MSP	22.12	17.16
GBS	31.15	24.09
SEBS	15.89	10.99
CBLS	22.18	23.32
DTAG	17.33	10.65
DTAP	24.03	18.91

Table 7.19: Level of Interaction between robots and decision making

Algorithm	Communication	Decision Making
RAND	Nil	Local
CR	Low	Local
HCR	Low	Local
HPCC	Low	Global
CGG	Nil	Offline
MSP	Nil	Offline
GBS	Low	Local
SEBS	High	Global
CBLS	High	Global
DTAG	Low	Global
DTAP	High	Global

Table 7.20: Level of Interaction between robots and decision making. *Nil* - no interaction. *Low* - idleness shared. *High* - idleness as well as robot's target node info shared. *Local* - only neighbours considered. *Global* - all nodes considered. *Offline* - path is decided a priori.

7.6.5 Conclusion

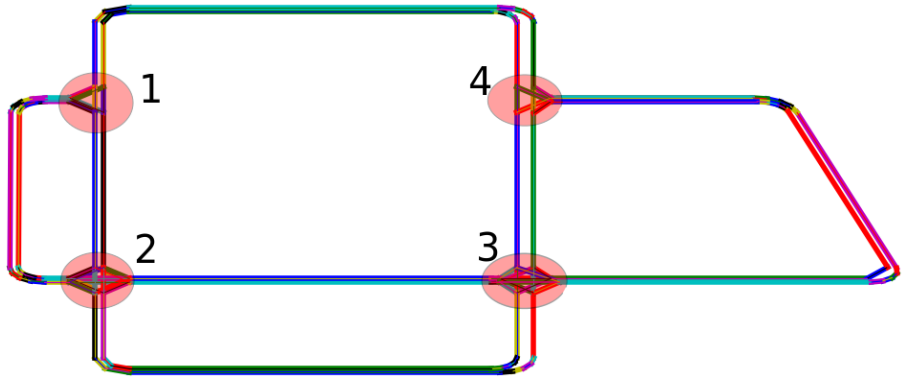
This work is only a preliminary step in addressing the defined problem of *Patrolling Campus-like Urban Environments using Multiple Car-like Robots*. This report attempts at setting up a direction for future work.
The simulation was done on a grid of 25 equally spaced nodes which is not the most ideal representation of a road network. Since, the robot locomotion was implemented using navigation stack, the area as a whole was classified into free and occupied cells. Since there were large number of interferences (robots coming in each others' way) during every simulation, the concern of emulating real-life road traffic was addressed to some extent. Also, the nodes in the layout could be classified as internal nodes and boundary nodes with varying degrees, thus enabling a scenario that captures both sets of dynamics.
The analysis carried out points out clearly that the strategies with low global average idleness are the ones in which the robots choose the node to be visited next from the entire graph. From amongst those with low average idleness, the Concurrent Bayesian Learning Strategy (CBLS) had the best result for correlation analysis. The auto-correlation function value was negligible for lag lengths greater than zero for all the nodes. This suggests that for an observer, observing the robot visits at the nodes over a period of time, there is no decipherable pattern at any of the nodes when CBLS algorithm is implemented. As a first proposal, *Concurrent Bayesian Learning Strategy* (CBLS) is suggested to address the stated problem.

7.7 Integration of Multi Robot Patrolling Algorithm with the Autonomous Driving

Since the analysed algorithms for multi robot patrolling are based on graph structures, the underlying layout of the road network had to be represented as a graph network.

7.7.1 Graph Structure of the Layout

Figure 7.47: The graph structure for the test layout

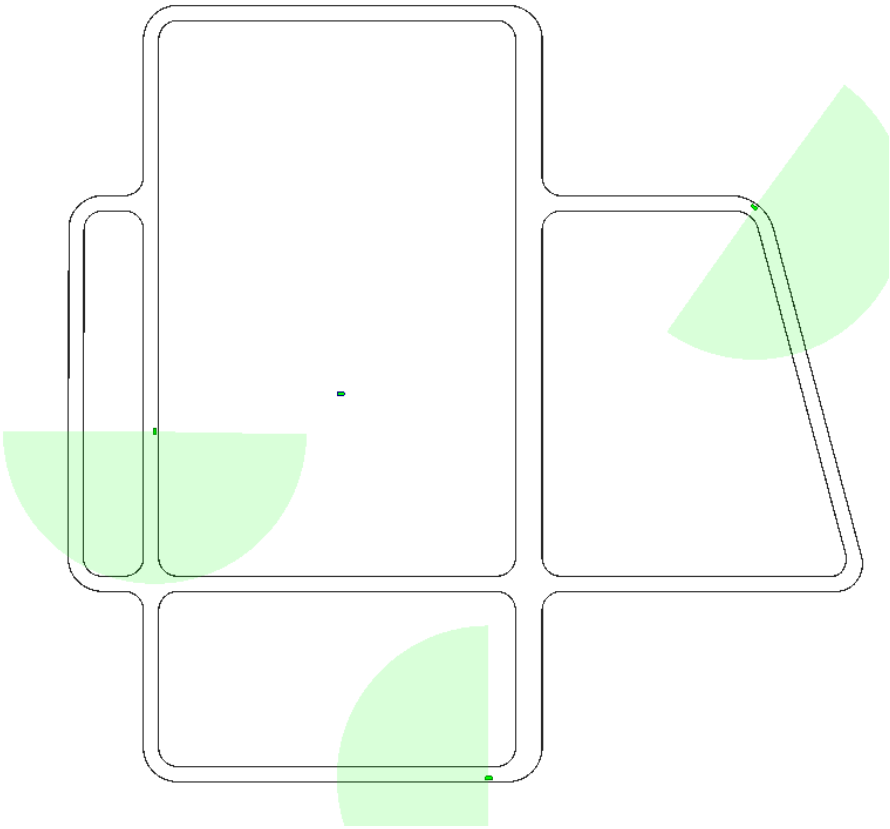


The graph structure representation has the intersections of the road network as nodes and the road segments connecting two intersections as edges. Each node has a number of way-points (classified as entry or exit based on the type of way-point) and each lane of road was a directed edge going from exit way-point of a particular node to entry way-point of one of its neighbouring nodes.

7.7.2 Patrolling Algorithm Integration

As pointed out in the previous section, the 'Concurrent Bayesian Learning Strategy (CBLS)' had the best performance amongst the analysed algorithms and hence the same was implemented in conjunction with the autonomous driving architecture.

Figure 7.48: Multiple active vehicles running on the CBLS strategy



Chapter 8

Conclusion

The stated objectives of the said project could be broadly outlined by the following points:

- A convenient road network representation for motion planning tasks in a campus like environment
- An autonomous driving strategy to navigate the environment (including ascertaining the environment state information, path planning and trajectory tracking tasks covering obstacle avoidance)
- A multi robot patrolling strategy requiring that the strategy be unpredictable for a complete state observer.

The above listed tasks have been addressed to the best of abilities and resources.

- A RNDF styled representation was adopted for road network representation with waypoints being the elementary object of interest.
- The state information was assumed to be available post processed after the sensory stage. The occupancy grid based obstacle information was achieved but not integrated and tested to our satisfaction due to lack of computing resources.
- A heuristic path planning algorithm was developed to go between given two way-points in the road network with the said path forming a C^1 continuous Bezier Curve.
- Faithful tracking of the achieved path could not be done in the given time frame and the available standard curve tracking literature had additional requirements on the curve in terms of smoothness etc.
- A preliminary analysis of benchmark algorithms addressing multi robot patrolling problem was done with the point of view of 'unpredictability criterion' requirement as proposed.
- The best performing algorithm 'Concurrent Bayesian Learning Strategy (CBLS)' was integrated into our simulations.

Overall, the work carried out is potentially a big step forward in terms of getting the research off the ground in the field of 'closed environment multi robot planning' for our research group and we would like to thank Centre for Artificial Intelligence and Robotics Laboratory of the DRDO for giving us this opportunity.

Bibliography

[1] Darpa Grand Challenge - Route Network Definition File (RNDF) and Mission Data File (MDF) Formats
grandchallenge.org/grandchallenge/docs/RNDF_MDF_Formats_031407.pdf

[2] A Sample RNDF
http://www.grandchallenge.org/grandchallenge/docs/Sample_RNDFv1.5.txt

[3] The 18 companies most likely to get self-driving cars on the road first
http://www.businessinsider.in/RANKED-The-18-companies-most-likely-to-get-self-driving-cars-on-the-road-first/1-Ford/slideshow/57996754.cms

[4] A Survey of Motion Planning and Control for Urban Vehicles
Brian Paden, Michal Cap, et al.

[5] Robot Operating System (ROS)
wiki.ros.org

[6] The Stage Robot Simulator
rtv.github.io/Stage

[7] PROUD-Public Road Urban Driverless-Car Test.
Alberto Broggi, Pietro Cerri, et. al.
IEEE Transactions on Intelligent Transportation Systems, 16(6):3508–3519, 2015.

[8] The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, volume 56. Springer, 2009.
Singh Sanjiv, Buehler Martin, Iagnemma Karl.

[9] A Review of Motion Planning Techniques for Automated Vehicles.
David Gonzalez, Joshue Perez, Vicente Milanes, and Fawzi Nashashibi.
IEEE Transactions on Intelligent Transportation Systems, 17(4):1135–1145, 2016.

[10] Realtime motion planning methods for autonomous on-road driving: State-of-the-art and future research directions.
Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka.
Transportation Research Part C: Emerging Technologies, 60:416–442, 2015.

[11] EureCar turbo: A self-driving car that can handle adverse weather conditions.
U Lee, J Jung, S Shin, Y Jeong, K Park, D H Shim, and I s. Kweon.
2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 2301–2306, 2016.

[12] Lost and Found : Detecting Small Road Hazards for Self-Driving Vehicles.
R. Pinggera, P., Ramos, S., Gehrig, S., Franke, U., Rother, C., and Mester.
In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1099–106, Daejeon, Korea, 2016.

[13] Detecting Unknown Obstacles for Self-Driving Cars: Fusing Deep Learning and Geometric Modeling.
Sebastian Ramos, Stefan Gehrig, Peter Pinggera, Uwe Franke, and Carsten Rother.
Icra, 2017

[14] Planning for Autonomous Cars that Leverage Effects on Human Actions.
Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan.
Proceedings of Robotics: Science and Systems, 2016.

[15] Indians spend more time behind the wheels than Chinese, Aussies: Survey
https://timesofindia.indiatimes.com/india/Indians-spend-more-time-behind-the-wheel-than-Chinese-Aussies-Survey/articleshow/46337734.cms

[16] On the Performance and Scalability of Multi-Robot Patrolling Algorithms
D. Portugal et. al.

[17] Multi-agent Patrolling: An Empirical Analysis of Alternative Architectures
A. Machado et. al.

[18] Combining Idleness and Distance to Design Heuristic Agents for the Patrolling Task
A. Almeida et. al.

[19] MSP Algorithm: Multi-Robot Patrolling based on Territory Allocation using Balanced Graph Partitioning
D. Portugal and R. Rocha

[20] Distributed multi-robot patrol: A scalable and fault-tolerant framework
D. Portugal and R. Rocha

[21] Concurrent Bayesian Learners for Multi-Robot Patrolling Missions
D. Portugal et. al.

[22] Distributed on-line dynamic task assignment for multi-robot patrolling
A. Farinelli et. al.