MULTI-LAYER APPROACH TO MOTION PLANNING IN OBSTACLE RICH ENVIRONMENT

A Thesis

by

SUNG HYUN KIM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2008

Major Subject: Aerospace Engineering

MULTI-LAYER APPROACH TO MOTION PLANNING IN OBSTACLE RICH ENVIRONMENT

A Thesis

by

SUNG HYUN KIM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Raktim Bhattacharya
Committee Members,	Srinivas R. Vadali
	Darbha Swaroop
Head of Department,	Walter Haisler

May 2008

Major Subject: Aerospace Engineering

ABSTRACT

Multi-Layer Approach to Motion Planning
in Obstacle Rich Environment. (May 2008)
Sung Hyun Kim, B.S., University of Illinois at Urbana-Champaign
Chair of Advisory Committee: Dr. Raktim Bhattacharya

A widespread use of robotic technology in civilian and military applications has generated a need for advanced motion planning algorithms that are real-time implementable. These algorithms are required to navigate autonomous vehicles through obstacle-rich environments. This research has led to the development of the multi-layer trajectory generation approach. It is built on the principle of separation of concerns, which partitions a given problem into multiple independent layers, and addresses complexity that is inherent at each level. We partition the motion planning algorithm into a roadmap layer and an optimal control layer. At the roadmap layer, elements of computational geometry are used to process the obstacle rich environment and generate feasible sets. These are used by the optimal control layer to generate trajectories while satisfying dynamics of the vehicle. The roadmap layer ignores the dynamics of the system, and the optimal control layer ignores the complexity of the environment, thus achieving a separation of concern. This decomposition enables computationally tractable methods to be developed for addressing motion planning in complex environments. The approach is applied in known and unknown environments. The methodology developed in this thesis has been successfully applied to a 6 DOF planar robotic testbed. Simulation results suggest that the planner can generate trajectories that navigate through obstacles while satisfying dynamical constraints.

To My Father God

ACKNOWLEDGMENTS

I am indebted to my advisor Dr. Raktim Bhattacharya for his valuable comments and advice. I would also like to thank Dr. Srinivas Rao Vadali and Dr. Darbha Swaroop for their contribution as committee members. I would like to thank Luis Calixto for his graciousness to help building hardware, running experiments, and providing support. I would also like to acknowledge James Fisher for building the TeRK. All this work would have not been possible without constant encouragement and support from my loved ones.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

A. Motivation

Autonomous robot technology is being rapidly applied to many applications for daily civilian and military life. Cleaning robots, such as Roomba, clean the house autonomously with a touch of a button. Autonomous capabilities have also been applied to DARPA Grand Challenge competition where vehicles must navigate through a dense urban environment while obeying traffic laws. It would demonstrate the leading edge in sensor technology, algorithms, and computational devices to gather information, process them with accuracy and speed, create trajectories, and actuate the vehicle control system. The success of DARPA Grand Challenge is expected to lead the industry to the next step where the automobiles become completely autonomous.

Unmanned vehicles such as Unmanned Aerial Vehicle (UAV) and Unmanned Ground Vehicle (UGV) take a significant role in modern battle field environment to provide intelligence, targeting, and execution capabilities. Predator developed by General Atomics is a good example of versatile UAVs. It can perform surveillance in the region of interest for many hours and collect battle field intelligence using an array of sensors such TV camera and IR sensor. What makes the Predator such a valuable option for the military commanders is the absence of pilot and subsequent life support and cockpit systems. This significantly improves the cost effectiveness of the vehicle. Autonomous capabilities can also reduce the workload of the operators - as well as the number of operators. In addition, eliminating human in the loop

---

This thesis follows the style of *IEEE Transactions on Neural Networks*.

reduces the risk of errors. Wide applications of unmanned vehicles since the mid 1990s have generated a trend of development and procurement of unmanned vehicles from all over the world.

This thesis concentrates on a problem of trajectory generation. In the presence of obstacles, a motion planner is required to devise a scheme that incorporate the motion constraints, tracks the waypoints, and obstacle avoidance. For a UGV scenario, obstacles can be steep hills, buildings, another vehicle, or threats. For a UAV, obstacles can be no-fly zones, Surface-to-Air Missile or Anti-Aircraft Artillery infested territories, or populated areas. All robotic vehicles possess their own unique dynamics. A trajectory incompatible with the vehicle dynamics will lead to poor performance. Vehicles also have constraints in fuel capacity, actuator effort, and mission time.

Thus, the objectives of this thesis is to generate trajectories in real-time that will satisfy the following requirements:

1) avoid obstacles

3) dynamically feasible

3) optimal a performance index (control effort, distance, time, etc)

B. Background

Various motion planning algorithms with obstacle avoidance have been developed in the computer science community. Motion planning in polytopic obstacle environment has led to the development of Configuration space (C-space). It is the starting point of most of motion planning schemes because they usually occur in 2D or 3D environment where the obstacles can be expressed as polytopes. The idea behind the C-space is to "grow" obstacles by constructing a map of obstacles which takes into account of various vehicle configurations [1, 2]. This ensures that the vehicle at any configuration

never contacts the obstacles. C-space classifies the topology into the obstacle region C_{obs} and the available region C_{free} for motion planning. In a sense, configuration defines the feasible state space for the trajectories.

A generalized term for a topological graph to solve a motion planning problem is a roadmap. There are two major subdivisions of roadmap planning depending on the methodology. One is the probabilistic roadmap method (PRM) based on sampling approach through the use of a rapidly exploring dense trees (RDT). RDT starts from a vertex in the map and expands the tree through sampled nodes. First, a set of vertices that include start, goal, and sampled points in C_{free} is constructed. RDT initiates from the starting vertex. Then, it connects with the closest neighboring vertex. The same process is recursively executed to build a tree. Every connection is called a path which must be collision-free of all obstacles. PRM approaches the sampling procedure in a probabilistic manner and it contains a bias towards the goal point, reducing the number of sampled points. In a rapidly-exploring random tree (RRT), dynamics of the vehicle can be embedded into the path generation step of the RDT. This allows RRT to be a very potent method for a roadmap generation.

In a deterministic framework, visibility graphs, voronoi diagram [3], cell decomposition [4] are well known algorithms that generate roadmaps. Some do not classify cell decomposition as a roadmap method, but it essentially builds up to a roadmap. These approaches focus on using geometric information from the existing features in the environment and use techniques to extract trajectories. For instance, equidistant edges are calculated in a Voronoi diagram by calculating and comparing distance functions between vertices or edges of C_{obs}. Deterministic methods produce segments over the entire map. Therefore, only the necessary segments will make into the query in the graph search step.

Graph search algorithm evaluates the cost (measures can differ) and builds a

sequence of segments that spans from the initial point to the goal through vertices q_i. A deterministic roadmap typically lacks smooth transitions between segments, so that it requires a post-processing for dynamically compatible trajectories.

Graph search is not only used to assist building roadmaps, but it has also evolved significantly to make an impact on the motion planning community. The backbone of such development is the A* algorithm [5]. It utilizes heuristics and searching logics to execute start to goal shortest-path calculation, allowing for efficient fast planning and replanning of trajectories. D* [6] contains improvements to A* to enabling vehicles to sense and create trajectories in a partially known or unknown environment. D* is widely used in navigation of autonomous vehicles such as DARPA Crusher and Spinner to carry payload in a real world off and on-road environment where known and static environment is scarce. Therefore, D* opened up a door for autonomous navigation capabilities, applicable for indoor and outdoor platforms. Despite the speed and efficiency of the algorithm, the paths need to go through a refining process to make them dynamically feasible similar to roadmap methods.

The optimal control problem (OCP) is used to create optimal trajectories that minimizes a certain performance criteria. In a basic environment where the vehicle can traverse freely, an OCP can be formulated and solved to generate a trajectory that satisfies boundary, dynamic, and control constraints of the vehicle. As stationary obstacles are added to the environment, the number of trajectory constraints multiplies, increasing the complexity of the constraints. Recent research on numerical methods for trajectory optimization have generated tremendous improvement in stability and convergence of the numerical algorithms [7]. However, brute force OCP results in high computation time and ill-conditioned formulations.

To eliminate this shortcoming, recent papers propose motion planning with trajectory primitives [8]. These primitives are pre-computed straight and minimum

radius turn profiles which integrates the full vehicle dynamics. Finding a sequence of primitives to generate a path completes the motion planning. Obstacles can be avoided by checking for collisions. However, in a cluttered environment, extracting a correct sequence might not be feasible and the path can become excessively lengthy by detouring. [9].

Flores [10] presents the separation of concern for obstacle avoidance and trajectories generation where different stages of work are specialized by different planners. Likewise, this thesis suggests an approach that divides the motion planning problem into two major constraints: obstacle avoidance and vehicle dynamics. First, collision-free regions and possible paths are found from the obstacle avoidance stage. In addition, B-spline characterization [11] of the trajectory along with differential flatness reduces OCP constraints further. This enhancement can improve convergence and computational efficiency on trajectory generation level.

We call this approach a multi-layer approach. The division of stages in motion planning lets each part to concentrate and solve respective objectives in the most efficient manner possible. As a result, the trajectory generation becomes more computationally efficient and real-time implementable.

C. Definition of Problems

Basic motion planning problem consists of the world in $\mathcal{G} = \Re^2$ or $\mathcal{G} = \Re^3$. The robot must maneuver from the designated starting point q_{init} to the end point q_{goal}. Let $O = \{O_1, O_2, \ldots, O_n\} \subset \Re^2$ be a set of convex polygons acting as obstacles. Depending on the scenario, obstacles are dynamic or static. The goal of this thesis is to find and implement algorithms that concentrate on localized layers. Synergizing the advantages of algorithms for different layers will provide the maximum compu-

tational efficiency to achieve real-time performance of these autonomous capabilities. The layers and their objectives are described in fig. 1. High level planner executes geometric and graph search algorithms commonly used for motion planning without dynamics model. It is done in discrete domain, creating discrete trajectories in the map space. Mid level planner receives the waypoints and region of interest for trajectories to pass. Based on the given information, feasible sets are created to ease the generation of dynamically feasible trajectories. In addition, optimal control is applied using differential flatness and B-spline properties in a receding horizon control manner. In the last level, a robot is put through the test and a feedback control is performed to track the given trajectory and reach the goal.

D. Overview

The thesis is made up of several chapters that explain the motion planning procedure:
Chapter I: High level planner implements geometric algorithm to simplify the map and the graph algorithms provide discrete trajectories.
Chapter II: Mid level planner create a friendly environment for dynamically feasible trajectory generation through feasible sets.
Chapter III: Low level planner describes the experimental setup and vehicles used for the simulation and implementation of the algorithm.
Chapter IV: Simulation results are presented that include motion planning in both known and unknown environment.
Chapter V: A brief conclusion provides an overview and the performance of the multi-layer approach for motion planning.

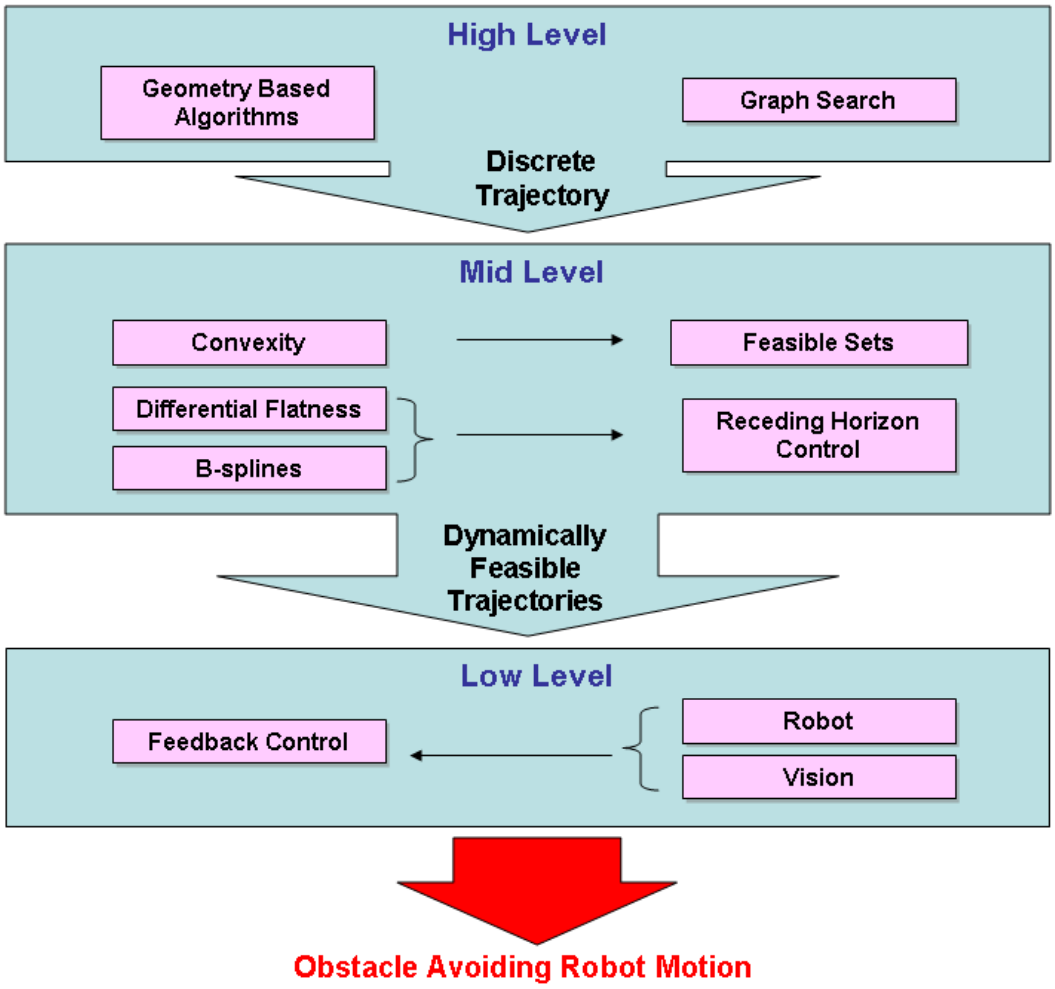


Fig. 1. Multi-layer concept: High and mid level planner create trajectories using specialized tools and the low level controls the robot with localization system

CHAPTER II

HIGH LEVEL PLANNING

When faced with a motion planning problem, the first challenge is to decide which direction the robot must move. Finding the most logical and effective sequence of points defines the objective of a high level planner. It designs paths by utilizing graph theories to help making the best choice of maneuver for the robot in a discrete domain. This section discusses the backbone of high level motion planning framework and reviews the existing algorithms and their applications in known and unknown environment. Appropriate implementations will be described to demonstrate the feasibility.

A. Configuration Space

Configuration space classifies map space. It uses Minkowski's sum to grow dimensions of the polygonal obstacle by adding possible orientations of the vehicle. Let $A \subset \mathcal{G}$ be a rigid body robot. Let $r = (x, y, \theta)$ be the configuration which represents the position and orientation. Many scenarios of $A(r)$ next to the obstacles are scrutinized by [12]. Considering the expansion of obstacles according to the addition of $A(r)$, C_{obs} is defined as

$$C_{obs} = \{r \in C \mid A(r) \cap \mathcal{O} \neq \emptyset\}$$

where $\mathcal{O}$ represents obstacles as mentioned in pg. 6. The rest of the space is obstacle free, hence $C_{free} = \mathcal{G} - C_{obs}$ obtained by set operations [3]. This provides the region where the vehicle can maneuver without any contact with obstacles.

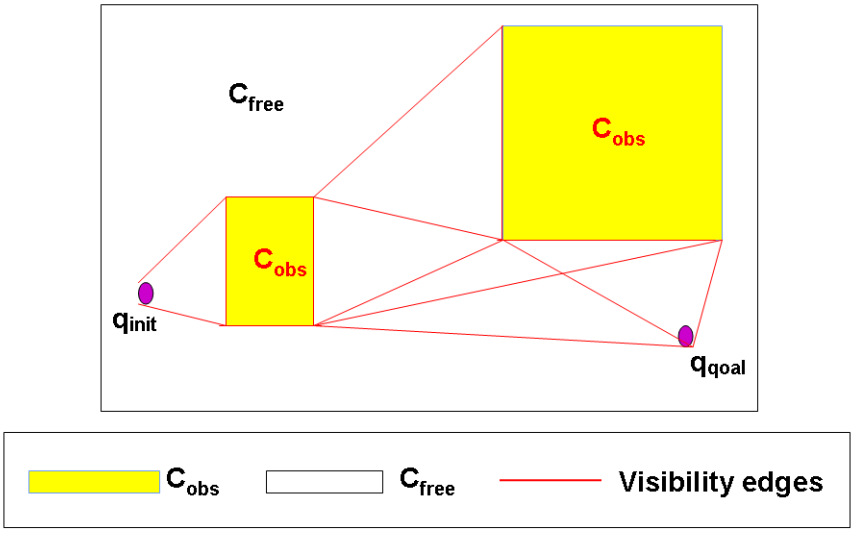


Fig. 2. Configuration space and the visibility edges

B. Known Environment

A roadmap can be constructed in a known environment (it is possible in unknown environment, but it needs to be updated to account for new realization of the environment). Among the roadmap algorithms, visibility graph and decomposition methods require a graph search algorithm to support paths generation, whereas a generalized voronoi diagram offers a different way of obtaining paths.

1. Visibility Graph

A visibility graph is a non-directed graph made up of vertices of C_{obs}, q_{init}, q_{goal}, edges of obstacles E_o, and general edges E_f that lie entirely in C_{free} [4]. The configuration space and edges are shown in fig. 2. With these nodes and edges, a graph search algorithm finds a set of sequential edges that connect q_{init} to q_{goal}. After calculating

the cost of traversal, a combination of edges that offer the least cost is designated as the optimal path in motion planning.

2. Decomposition

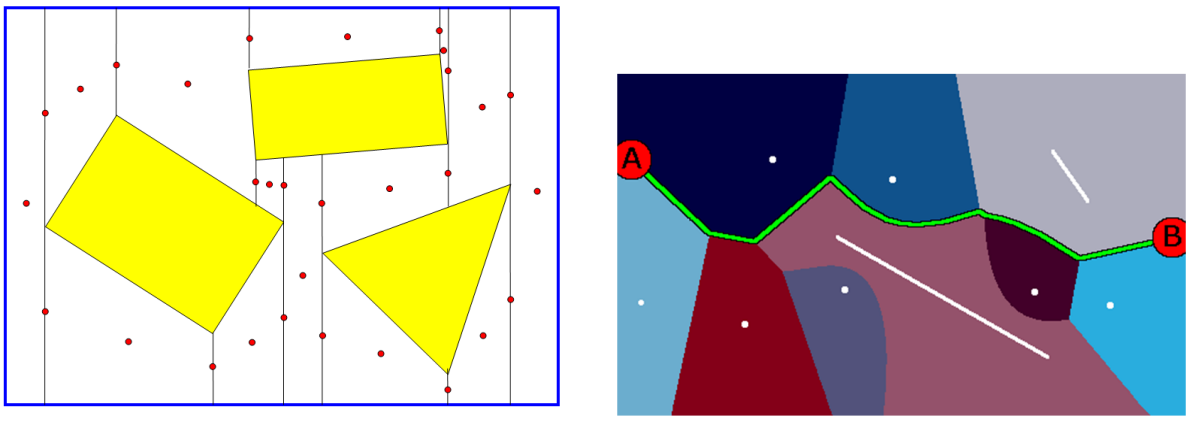
C_{free} in configuration space is normally a large non-convex polytope. The geometric complexity limits the extensive use of algorithms in the workspace since many geometric operations are based on convex hull assumption. Therefore, a convex decomposition prior to the roadmap generation can ease the searching process. Any n-dimensional decomposition of C_{free} consists of simplicial complexes [3]. Typical methods such as vertical, cylindrical decompositions, and triangulations result in simplices T in $\Re^2$. While visibility graph allowed the use of E_o as a path, decomposition methods use centroids of polygons and midpoint of edges as nodes (shown in red dots), avoiding contact with C_{obs} at all as found in fig. 3(a).

3. Generalized Voronoi Diagram

We define $P = \{p_1, p_2, \cdots, p_n\} \in \Re^2$ as voronoi sites. A voronoi region is formed by the following expression for all x in Euclidean space [13]:

$$V(p_i) = \{x : |p_i - x| \leq |p_j - x|, \forall j \neq i\}$$

Therefore, applications such as calculating cellular phone tower placement and grocery store location utilize voronoi diagrams. A generalized voronoi diagram (GVD) applies the same principle of the voronoi diagram, but it is extended to calculating maximum distances among vertices and line segments of polygonal obstacles. Three combinations of geometric features (vertex to vertex, edge to edge, vertex to edge) produce voronoi edges and arcs representative of equidistant points between obstacles. Equidistant segments for an edge to edge and vertex to vertex features are lines



(*a*) Vertical Decomposition (*b*) GVD (taken from [21])

Fig. 3. Variety of high level implementable algorithms

whereas a vertex to edge relationship results in a parabolic voronoi arc. Essentially, GVD creates medial axis paths in C_{free} as shown in fig. 3(b). A vehicle tracking GVD arcs is guaranteed to maintain maximum clearance from any nearby obstacles.

4. A* Algorithm

Whether the environment consists of a finite number of grids or a set of edges and vertices, the robot must decide on its traversal based on the known information. A*(pronounced A-star) is a well known algorithm for finding the shortest path between two nodes given traversable nodes or edges. The feature that differentiates it from other graph search algorithms such as best-first search and Dijkstra's algorithm [14] is the use of heuristics. It provides a distance estimate to the goal for the algorithm to direct the search better. A* also takes account of the cost $g(s)$ from the q_{init} to the current node s. The heuristic estimate $h(s)$ gives the cost from the current node to the q_{goal}. The measure of heuristics can differ from manhattan, diagonal, or Euclidean distance. Then, nodes with the least combined cost $f(s) = g(s) + h(s)$ are selected to create a path with the minimum $\Sigma f(s)$. Therefore, we can see that A*

algorithm neither searches for the goal comprehensively nor randomly. It makes an educated guess of the feasible paths. Readers are referred to the reference [5] for the details of the algorithm. It is proven that A* is complete and optimal.

5. Implementation

In this thesis, the modeled environment is very cluttered. Therefore, it is desired that decompositions such as vertical, cylindrical, etc, which generate a large set of polygons, are avoided to conserve the computation time. In this case, triangulation can offer relatively small number of grids, hence lower computation time for the planner.

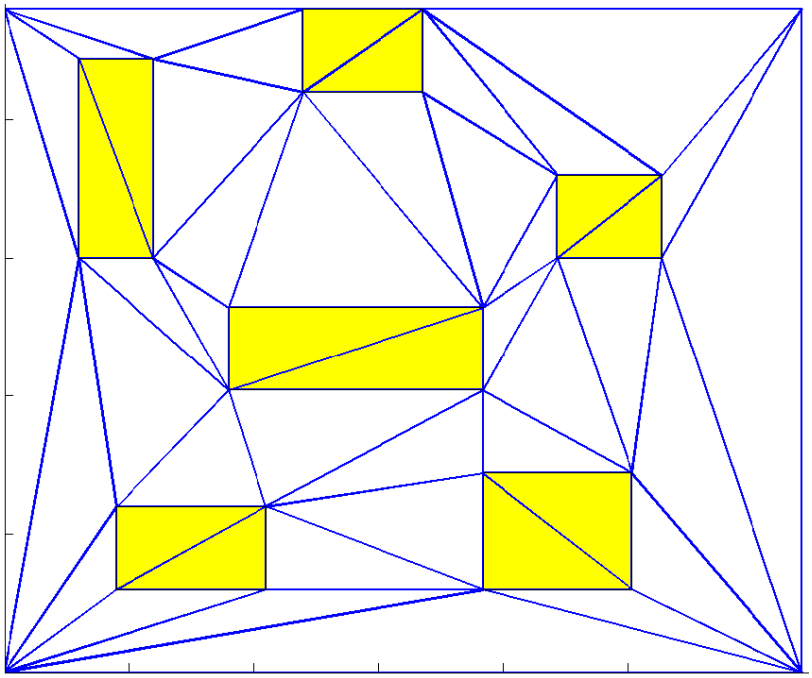


Fig. 4. Triangulation of C_{free}

Delaunay triangulation is the dual of voronoi diagram. Vertices of obstacles are voronoi sites $p_i \in P$ and connecting them would result in Delaunay triangles. It provides m triangular meshes of different sizes $\mathcal{G} = \bigcup_{i=1}^{m} T_i$ where T_i represents each

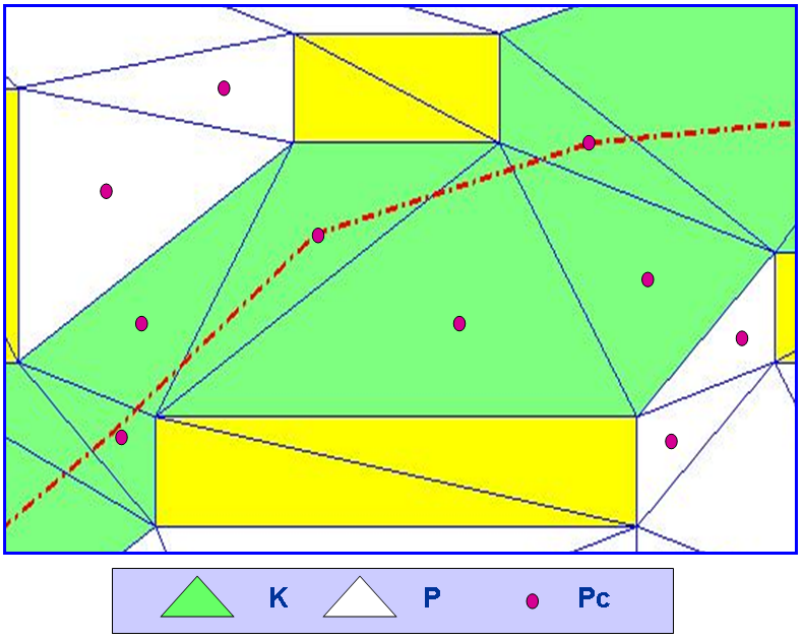


Fig. 5. A* algorithm in the map: The algorithm connects red dots, which are the centroids of decomposed triangles, for shortest path

mesh as shown in fig. 4. However, in highly non-convex environment, Delaunay triangulation cannot take into account of the restriction that edges of the triangle must not intrude the obstacles. Thus, a pruning procedure takes place to capture the polygons P_i which is the result of $P_i = T_i - C_{obs}$. For each P_i, corresponding centroids Pc_i are found.

A* algorithm tries to find a shortest path with the nodes shown as red dots in fig. 5. A set of centroids Pc are given as the nodes and a constraint is applied to the A* planner so that no path segment can intrude obstacles. Therefore, it can be proven that connections between Pc create a path S_s and it is the shortest path. Polygons in contact with the shortest path are designated as $\mathcal{K}$ and these are easily discovered. For the definition,

$$\mathcal{K} = \{P |\ P \cap S_s \neq \emptyset\}.$$

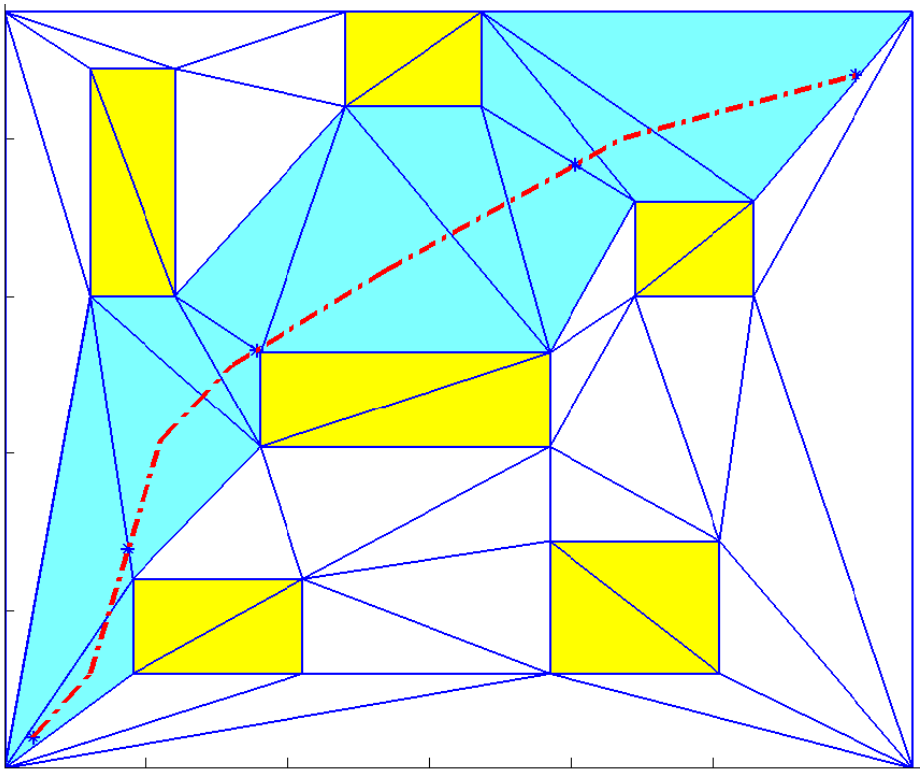


Fig. 6. Shortest path as a result of A* algorithm

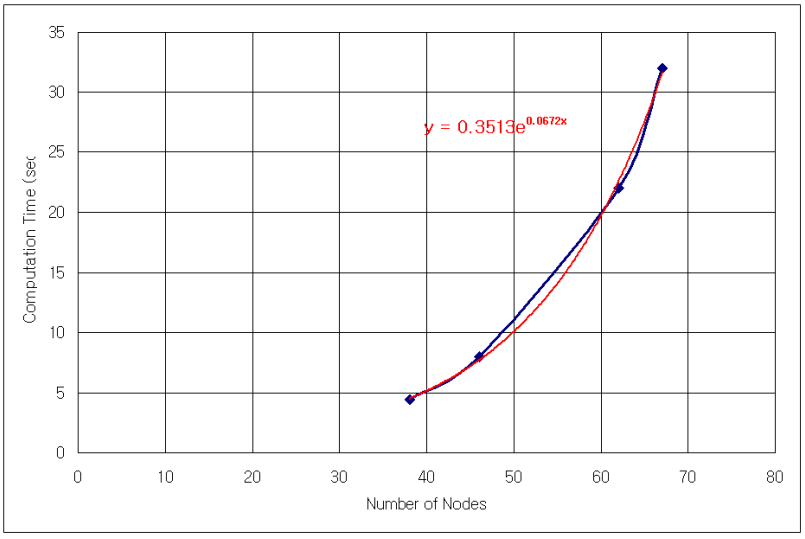


Fig. 7. A* algorithm computation time: Increasing number of nodes cause exponential computation time increase

Not only the polygonal grids are found, but this also lays the foundation of feasible sets which will be discussed in the next section. Note that the shortest path from q_{init} to q_{goal} in fig. 6 is not the globally optimal or shortest path. This originates from the sparse distribution of nodes provided to the search tree. At this point, we need to consider which factor is more important to this research: add more nodes and obtain shorter paths or decrease the computation time. Fig. 7 represents the relationship between computation time and number of nodes. The exponential increase of computation time strongly suggests that minimizing the number of nodes must be the priority to run the algorithm close to real time. Moreover, as explained earlier, the purpose of A* is to find triangles that lie on the path than the actual shortest path on the map. Therefore, the contribution of A* to the global optimality is minimal and in turn the computation time for less A* run time can be allocated to the mid level planner.

This combinatorial approach deviates from the traditional approaches explained in prominent motion planning literature. Two step procedure seems to require more computational effort compared to other methods. However, it can be observed that the effect of this drawback will diminish when creating dynamically feasible trajectories in the next section.

C. Dynamic or Unknown Environment

1. Incremental Search Algorithm

We will review prominent methods applied in real world vehicles. Please refer to reference [15]. Improvement has been made to A* to approach the motion planning in partially known or unknown environment. The motivation was that many robotic vehicles do attain only limited amount of information on terrain and obstacles. The

vehicle must utilize its on-board sensors and fuse the previous information to navigate relying on those. D* [6] was originally developed for such purpose. It finds the discrepancy between the given map and the real map or it constructs a map while traversing. It is often referred as backward A* where the search tree is constructed from the goal state to minimize the number of nodes whose cost change.

Another modification to the A* algorithm brought the Lifelong Planning A* algorithm [16], which is algorithmically very similar to A*, having better theoretical ground and simpler implementation. One particular departure or advance of LPA* from A* is the rhs value. It is one step lookahead value of the $g(s)$ value from A*. It is defined as

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

where s is a vertex. Measures such as $g(s)$, $h(s)$, and $f(s)$ are preserved in LPA*. If g value equals the rhs value, the vertex is called locally consistent. The algorithm maintains a priority queue which contains all the "inconsistent" vertices. The lowest "key" retains the highest priority in the queue. A key is defined as

$$k(s) = [k_1(s) \; ; \; k_2(s)]$$

where $k_1(s) = min(g(s), rhs(s)) + h(s, s_{goal})$ and $k_2(s) = min(g(s), rhs(s))$. Ties are broken arbitrarily. After all vertices become consistent, LPA* can trace back from the goal to the starting point with the least cost, providing the shortest (optimal) path. With LPA*, local changes of the map does not affect the entire vertices or cells. Only the local vertices whose costs need to be updated according to the change undergo recalculation and replanning. This makes LPA* very efficient if the computer allows to maintain a large set of data for the entire map. This application can be easily seen

on the applet [17].

While LPA* starts the search from the q_{init}, D* Lite initiates the search from q_{goal}. This characteristic reduces the changes in the priority queue. It also accounts for the vehicle movement. This means that D* Lite can encounter a dead end if the detection horizon cannot cover the dead end in the previous iteration.

Further introduction of Field D* [18] states the issue of suboptimal paths created by limited traversability between the nodes with preset angles in $\pi/4$ increments. It suggests an interpolation based solution to obtain a globally optimal path. Theta* cuts the computational time even more to create algorithms to be more real-time implementable [19].

2. Implementation

LPA* is implemented for navigation in an unknown environment. The C code was provided by Dr. Sven Koenig from University of Southern California. In addition to the code, a new routine was written to the maze generation part to include user defined maps. A special output was created to process the results from the code. However, the basic concept and its implementation follows the LPA* algorithm.

A choice of 8-connected graph or 4-connected graph is given to the user. The cost function is governed by the Euclidean distance. As the vehicle proceeds to the goal, sensors detect new obstacles and LPA* replans the path as shown in figs. 8 and 9. Sensors has a certain range (sensor horizon) ahead of the robot position, however, its horizon cannot provide the coverage for the whole area. For this thesis, we will assume that the sensor horizon is 10 grids in four directions, making the detection area a square. This finite sensor horizon causes multiple replanning episodes to occur where the trajectory is planned within the detected area and the robot moves according to the trajectory. At the final robot location in each replanning, the sensor builds a new

map and replans the trajectory for the next horizon. The planning horizon eventually covers the goal and the final planning can be done to reach the goal. Note that the detected region is split into half to only select the region the vehicle is headed towards. This region is called the region of interest (ROI) as shown in fig. 9.

High level algorithms in known and unknown environment are able to provide feasible paths. Although not always dynamically feasible, the information can be inherited to the mid level planner which efficiently extracts final trajectories.

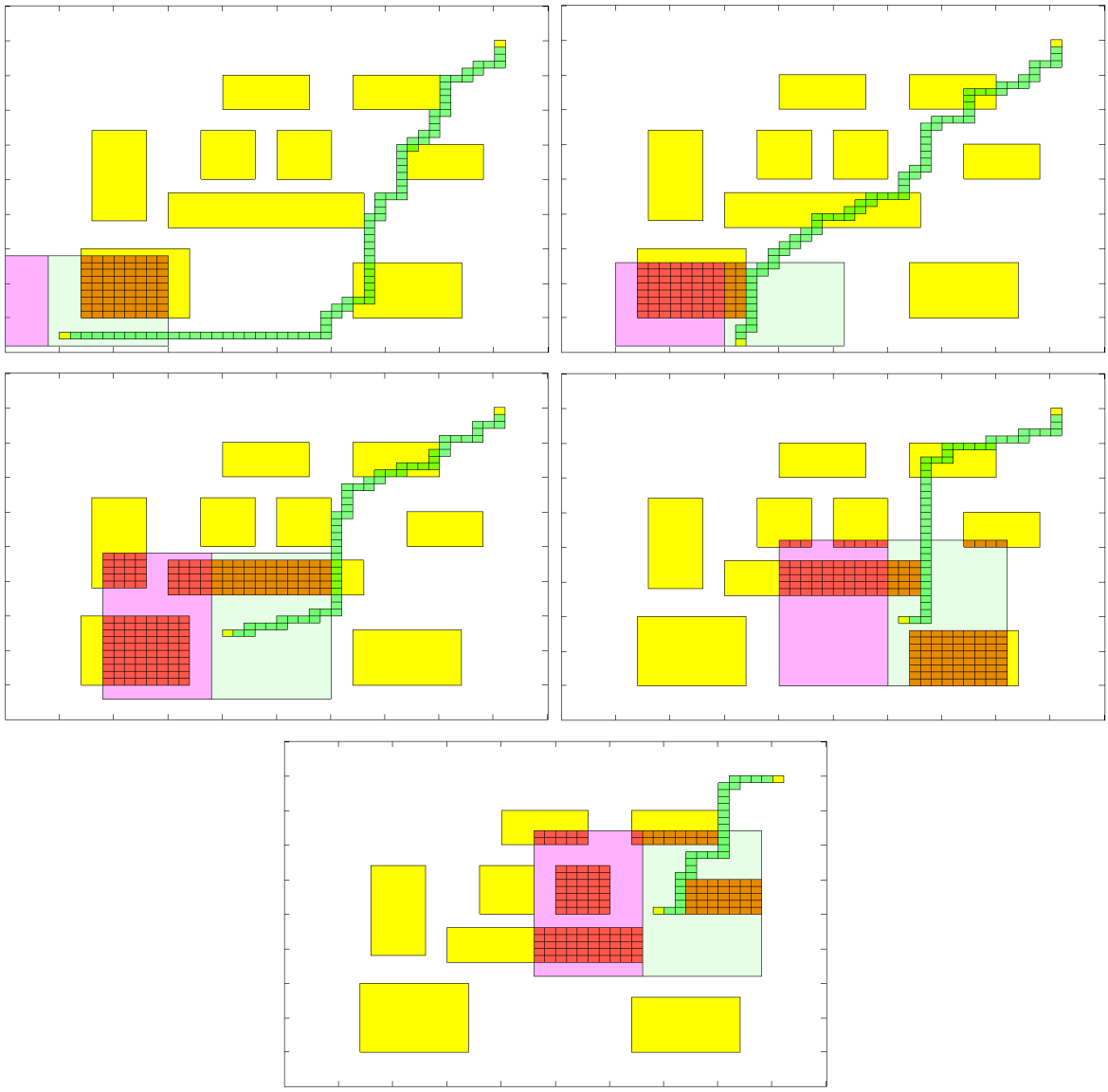


Fig. 8. Four connected grids: Original obstacles are shown in yellow. Green grids show the projected path calculated by LPA*. We can see that the green cells pass through obstacles beyond the sensor horizon. Red and orange areas show obstacle region recognized by the sensors.

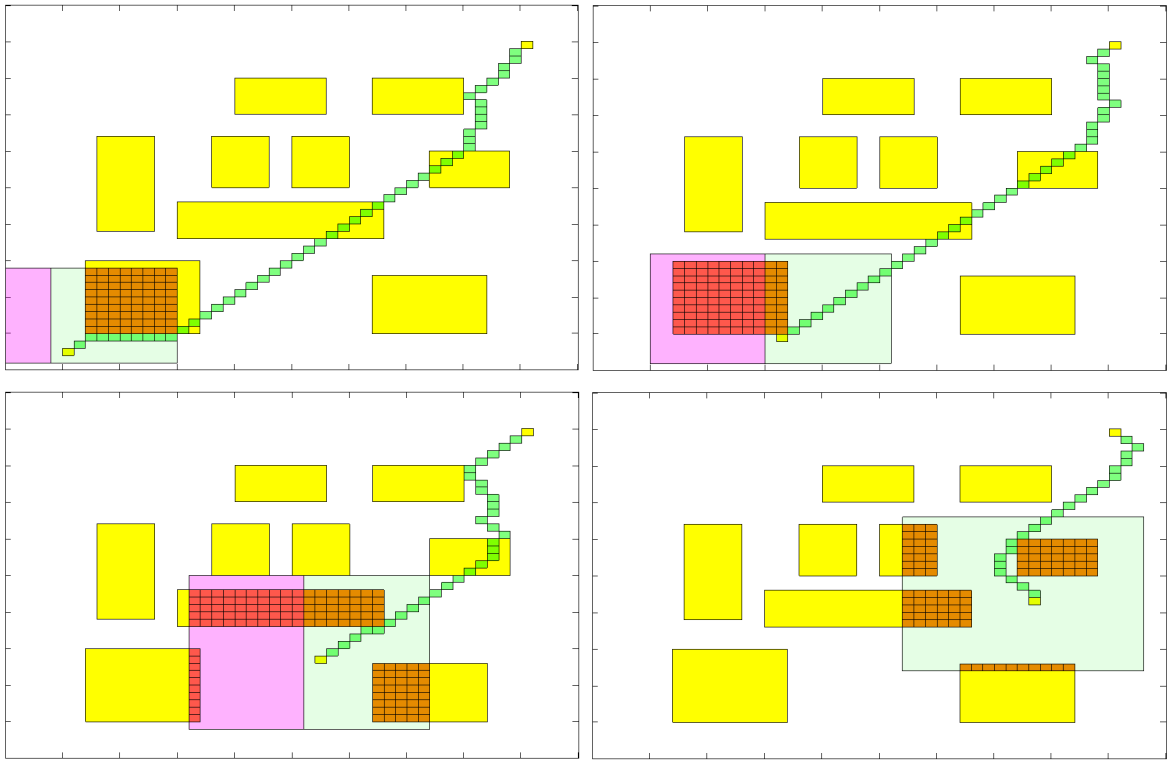


Fig. 9. Eight connected grids: Light green region is determined by the planner by evaluating where the projected path is headed to. Purple region represents unnecessary region in the sensor horizon, whose information is discarded during trajectory calculation. Paths from 8 connected grid tend to hug the obstacles due to the lowest cost.

CHAPTER III

MID LEVEL PLANNING

Although the high level planner provides a sequence of discrete points, no consideration for the dynamics of the vehicle is given at this stage. The purpose of the mid level planner is to build trajectories that follow prescribed waypoints while avoiding obstacles and satisfying the dynamic and actuator constraints of the vehicle.

First, feasible set generation is discussed. Details of the implementation is described in known environment, then the changes of procedure for the unknown environment is discussed. This is followed by applications of B-spline, OCP formulation, and Receding Horizon Control (RHC) are explained to generate paths.

A. Feasible Sets

Tracking trajectories made up of piecewise linear segments poses a challenge to the continuous vehicle motion; discontinuous velocity and acceleration vectors at the segment junctions are difficult to track. Although replacing the roadmap with an optimal control problem is a valid option, it is an expensive and computationally arduous process due to constraints that exponentially grow for an increasing number of obstacles and dimension of state space. For some environment where sharp turns are necessary due to narrow passes, OCP would suffer computationally to generate feasible trajectories because the constraints on the control with the states are very restrictive. Therefore, it is not suitable for a complex environment.

We propose the concept of feasible set. It can be defined as the following:
Given polygon $\mathcal{K}$ on the shortest path, a feasible trajectory τ must satisfy the relationship $\tau \subset FS$ where FS is a finite union of local $\mathcal{K}$. Recall that polygon $\mathcal{K}$ is a subset of C_{free} from the previous chapter.

The feasible set algorithm is a recursive algorithm as explained in algorithm A. The algorithm starts with the parent polygon $\mathcal{K}_p$ set as the one that include the start point q_{init}. From this point, a while loop contains the rest of the steps until there emerges a FS that includes q_{goal}. A child $\mathcal{K}_c$ can be found by examining neighboring polygons $\mathcal{K}_n$. $\mathcal{K}_c$ is the polygon that intersect shortest path S_s. Fig. 10 shows how $\mathcal{K}_c$ is selected among the candidates $\mathcal{K}_n$ and the incremental construction of feasible sets. The selection process for $\mathcal{K}_c$ ensures that all FS follow the S_s while neglecting irrelevant polygons present in the C_{free}. Then, the union of parent and child polygon is done in step 2. If this results in a convex polygon, the union is designated as the parent polygon (step 4). Otherwise, only $\mathcal{K}_p$ becomes a FS (step 6) and its $mathcal K_c$ inherits the title of parent polygon.

This operation incrementally builds feasible sets along the shortest path. Two properties of FS benefit the motion planning. One is its local convexity that guarantees trajectories to be in the feasible set if characterized by B-spline, due to the property that B-spline trajectories lie in the convex hull of the control points. The other property is that a FS usually include more than one polygon K. Even though a triangle is considered convex, generating trajectories in every triangle is computationally challenging. By making the largest convex polygon in the vicinity of the vehicle, the number of trajectory generation inside FS decrease significantly. Fig. 11 indicates that there are 11 triangles from the start to the goal. With the application of FS, there are only 4 convex polygons where trajectories are generated. Assuming the CPU time for a trajectory generation scheme is constant, this can reduce the computation time by more than 50%.

Algorithm A. *Feasible Set Generation*

1. while $FS \cap q_{goal} = \emptyset$
2. $\mathcal{K}_c = \{\mathcal{K} : \mathcal{K} \cap \mathcal{K}_p \neq \emptyset \quad , \mathcal{K} \cap S_s \neq \emptyset\}$
3. check for convexity of $(\mathcal{K}_p \cup \mathcal{K}_c)$
4. if true,
5. $\mathcal{K}_p = \mathcal{K}_p \cup \mathcal{K}_c$
6. else
7. $FS = \mathcal{K}_p$
8. $\mathcal{K}_p = \mathcal{K}_c$
9. end
10. end

1. Known Environment

In known environment, one time execution of A* algorithm provides all the necessary polygons to execute feasible set generation. Thus, feasible sets are generated all at once. Concerns arise because of sharp turns created by the dimension of the FS. If there is not sufficient lead space for the robot to turn, a very tight turn might not be possible and it can lead to infeasible controls. Appropriate measures are devised to add additional neighboring polygons in the direction of robot movement to give more space for a turn. A method of imposing a velocity constraint on the transition point from one FS to the next one can also direct the robot's heading to the direction of the next transition point, minimizing the effort to reach the next transition point.

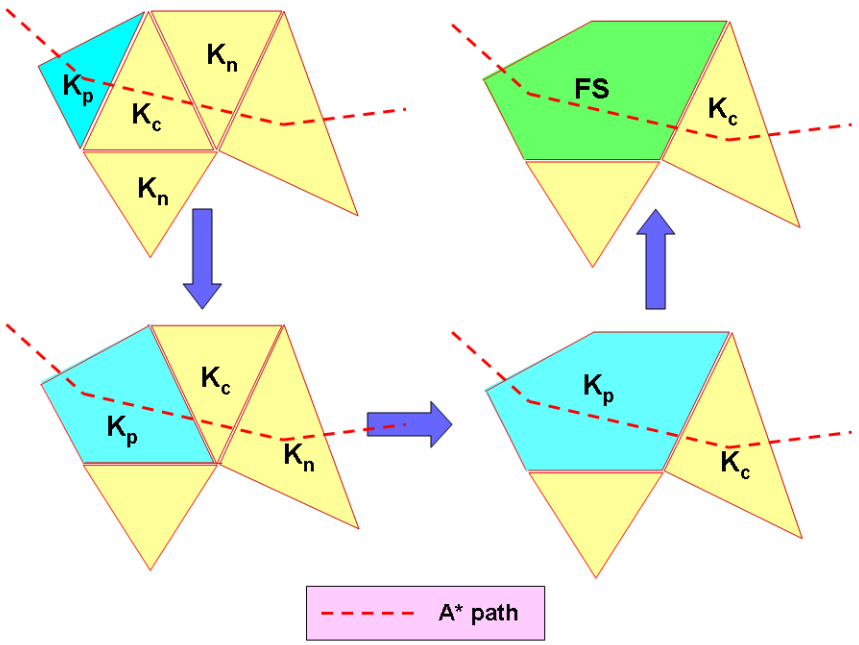


Fig. 10. Capturing FS: As explained in Algorithm A, $\mathcal{K}_p \cup \mathcal{K}_c$ is evaluated for the convexity and the subsequent decision whether to designate the union as a FS or to let the expansion continue is made

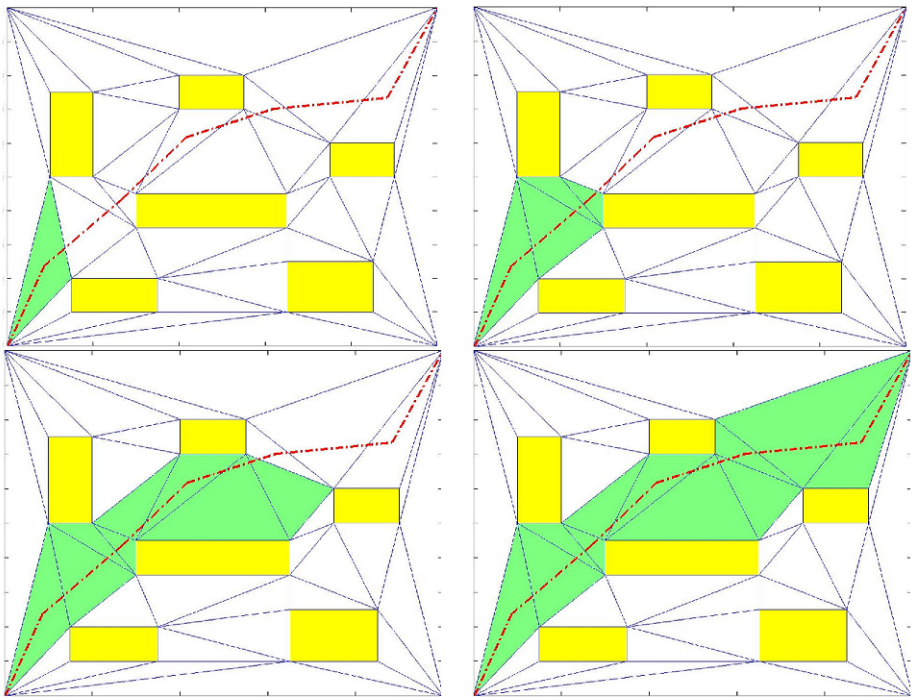


Fig. 11. Feasible Sets in known environment: A* algorithm is executed once (redline) and each subfigure is generated everytime a new FS (green) is generated

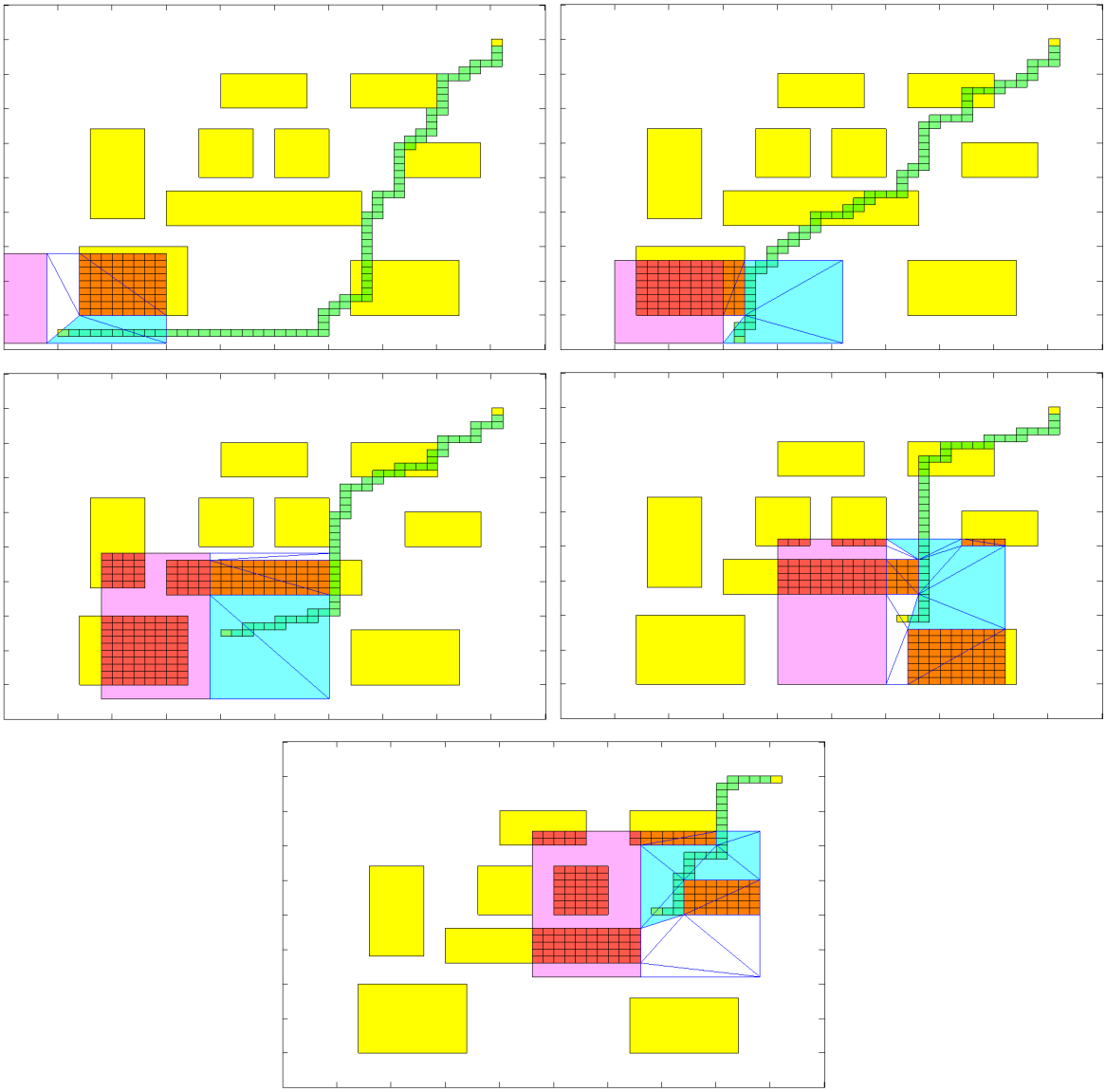


Fig. 12. Feasible Sets in unknown environment: LPA* generates FS (cyan) at every new detection of area

Algorithm B. *Known environment trajectory generation*

1. Obtain the map with obstacle information
2. Create C-space : C_{free} and C_{obs}
3. Triangulate C_{free} and prune the triangles : P
4. A* Execution : shortest path S_s
5. Find polygons of interest : polygon $\mathcal{K}$
6. Feasible Set Generation : FS
7. Trajectory Generation using B-splines and optimal control

2. Unknown Environment

Motion planning in unknown environment requires sensing and mapping of the surroundings as the vehicle maneuvers, therefore feasible sets are generated every time new information is gathered by the vehicle. The mechanism of feasible sets generation remains identical for this case, with a few differences. First, the use of A* algorithm is not required. LPA* provides the projected shortest path. The region inside the sensor horizon is triangulated region of interest and the polygons $\mathcal{K}$- relevant polygons on the projected path of the robot - are found. Then, the feasible set generator (explained in Algorithm A) takes $\mathcal{K}$ and creates FS. FS is generated multiple times as it is a subset of the detected area that move until the goal is reached due to the nature of finite detection horizon as presented in fig. 12.

In summary, the feasible set generation code that the mid level planner uses is the same for any case. Algorithm B. represents the skeleton of the high and mid level execution for the known environment. A slight modification needs to be made for the

motion planning in unknown environment. All the steps - with the replacement of step 4 with a simple search for FS candidates - must repeat several times until the goal is within the range of the sensor.

Next section provides the description of the B-spline characterization of the trajectory and its application to motion planning.

B. B-spline Trajectory Characterization

All previous steps involved obtaining C_{free} and useful subdivisions to provide feasible sets. The final step to motion planning is the trajectory generation using B-spline curves [15]. Following is the definition of a B-spline:

$$S(t) = \sum_{i=1}^{n} \Phi_i B_{i,r}(t) \ ,$$

where Φ_i are the n control points and $B_{i,r}$ denotes B-spline basis functions with a degree r. t represents a non-decreasing knot sequence $t = [t_1, \ldots, t_l]$. The number of control points is determined by the formula

$$n = N_i(r - s) + s$$

where $s \leq r$, N_i is the number of piecewise polynomials in the curve, and s is the smoothness condition which indicates the curve will retain C^{s-1} continuity. The basis function is calculated using Cox-de Boor recursion formula

$$B_{i,0}(t) := \begin{cases} 1 \ if \ t_i \leq t \leq t_{i+1} \\ 0 \ otherwise \end{cases} ,$$

$$B_{i,r}(t) := \frac{t - t_i}{t_{i+r} - t_i} B_{i,r-1}(t) + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}} B_{j+1,r-1}(t).$$

In two dimensional space, we define $x(t)$ and $y(t)$

$$x(t) = \sum_{i=1}^{n} \alpha_i B_{i,r}(t) \ ,$$

$$y(t) = \sum_{i=1}^{n} \beta_i B_{i,r}(t) \ ,$$

where α and β denote control points in x and y coordinates. A unique property of B-spline we can exploit is that a B-spline curve remains inside the convex hull of its control points. We can translate this relationship mathematically as,

$$A_c S(t) \leq b_c$$

where A_c is a m by n matrix. The rows of A_c reflect the number of edges convex hull of control points creates and the columns n represents the dimensionality of space. In $\Re^2$, A_c has two columns. $S(t)$ is a set of $(x(t), y(t))$ that satisfy the constraint. Our goal to keep trajectory $S(t)$ inside the FS can be achieved by an additional constraint which is very similar to the previous equation

$$A_f \rho \leq b_f$$

where subscript f denotes the relation with FS and ρ represents control points (α, β). For the optimization process, we parameterized α and β. Fig. 13 shows a graphical representation of the concept.

A continuous trajectory $S(t)$ can be characterized to satisfy the vehicle dynamics. This enables us to simplify hard trajectory constraints on the states; address only FS regardless of obstacles. Other properties such as local support - change in one control point does not alter the whole curve - ease shaping the trajectory. These advantages bring path planning easier than conventional OCP with brute force constraints enforcement.

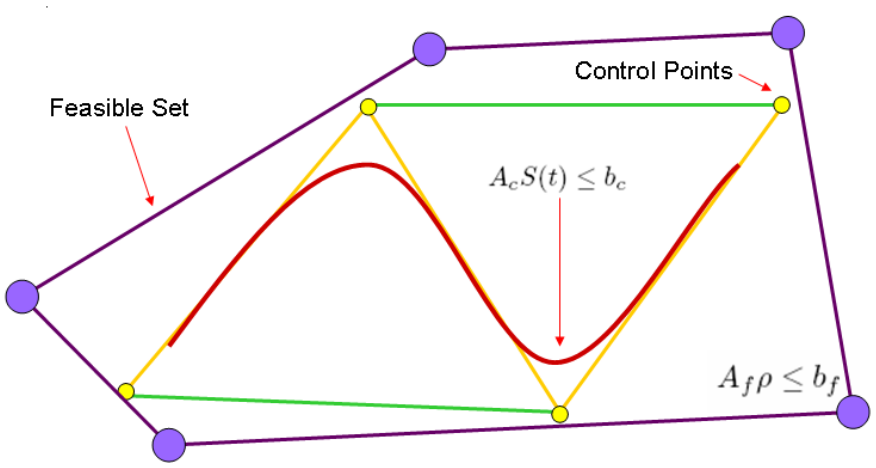


Fig. 13. Use of B-splines: Trajectory remains in the convex hull of control points which also reside inside feasible sets

C. Trajectory Generation

The dynamics and cost are not specified in this section and the detailed formulations will be discussed along with the introduction of testbed. We follow the standard formulation for the OCP:

Minimize

$$J = \phi_0(x_0, u_0, t_0) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt + \phi_f(x_f, u_f, t_f) \ ,$$

subject to dynamics

$$\dot{x} = f(x(t), u(t)) \ .$$

The initial, final, and trajectory constraints are

$$lb_0 \leq \psi_0(x_0, y_0, t_0) \leq ub_0 \ ,$$
$$lb_f \leq \psi_f(x_f, y_f, t_f) \leq ub_f \ ,$$
$$lb(t) \leq \psi_t(u(t)) \leq ub(t) \ .$$

Feasible sets are enforced as trajectory constraints. Therefore, it is expressed as

$$A_i \rho \leq b_i$$

As the trajectory migrates through the sequence of N feasible sets, A_i and b_i also change,

$$\begin{bmatrix} A_1 & 0 & \dots & 0 \\ 0 & A_2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \dots & A_N \ . \end{bmatrix} \begin{pmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix} ,$$

where $\rho_i = [\alpha_i \ \beta_i]^T$. After OCP is formulated, we parameterize states x and y into B-spline curves. Now OCP has become an optimization problem for coefficients α_i and β_i at every time step.

$$z = \begin{Bmatrix} \alpha_1 \\ \dots \\ \alpha_{Nc} \\ \beta_1 \\ \dots \\ \beta_{Nc} \end{Bmatrix}$$

Nc is the total number of collocation points where all constraints are enforced and the cost function is evaluated. Pseudo-spectral and other collocation methods can offer better effectiveness for calculations as presented in [8, 16].

The OCP formulation was then transcribed into a nonlinear programming (NLP) problem. To implement the SNOPT solver, the formulation was altered as the following:

$$min \ F(z)$$

subject to constraints:

$$LB \leq \left\{ \begin{array}{c} z \\ Az \\ C(z) \end{array} \right\} \leq UB$$

The resulting α and β from the NLP generate state and control output.

D. Receding Horizon Control

Receding horizon control (RHC) repeatedly solves for controls for a finite horizon T. At time t, a set of control inputs $U_1 = [u_t \dots u_{t_f}]$ are produced. Instead of executing the control U_1 for its whole length, RHC recalculates the controls at the next time step $t+1$ to obtain new controls. Executing and creating controls this way minimizes the error propagation throughout the trajectory. Fig. 14 shows a graphical representation of the approach. In this thesis, OCPs are solved in receding horizon manner.

E_f is defined as a shared edge between two consecutive feasible sets: FS_i and FS_{i+1}. Therefore, $E_f = FS_i \cap FS_{i+1}$. We could select the point where the shortest path S_s made a contact with E_f to be the waypoint WP_i as presented in fig. 15.

In unknown environment application, S_s is replaced by a line that connects the centroids of FS_i and FS_{i+1}. If the FS_i is the first one in the detected region, the centroid is replaced by the starting point. If the region of interest only contains one FS, such method is not achievable. In this case, a point near the end of the FS can be chosen as a waypoint through the projected path. By taking 3 grids closer to the start grid, the waypoint will be located inward towards the first waypoint. This change of scheme is motivated because high level planner has a tendency to create projected path next to the obstacles, making the trajectory generation difficult for the OCP. By taking this step, the waypoints are generated near the center of the FS,

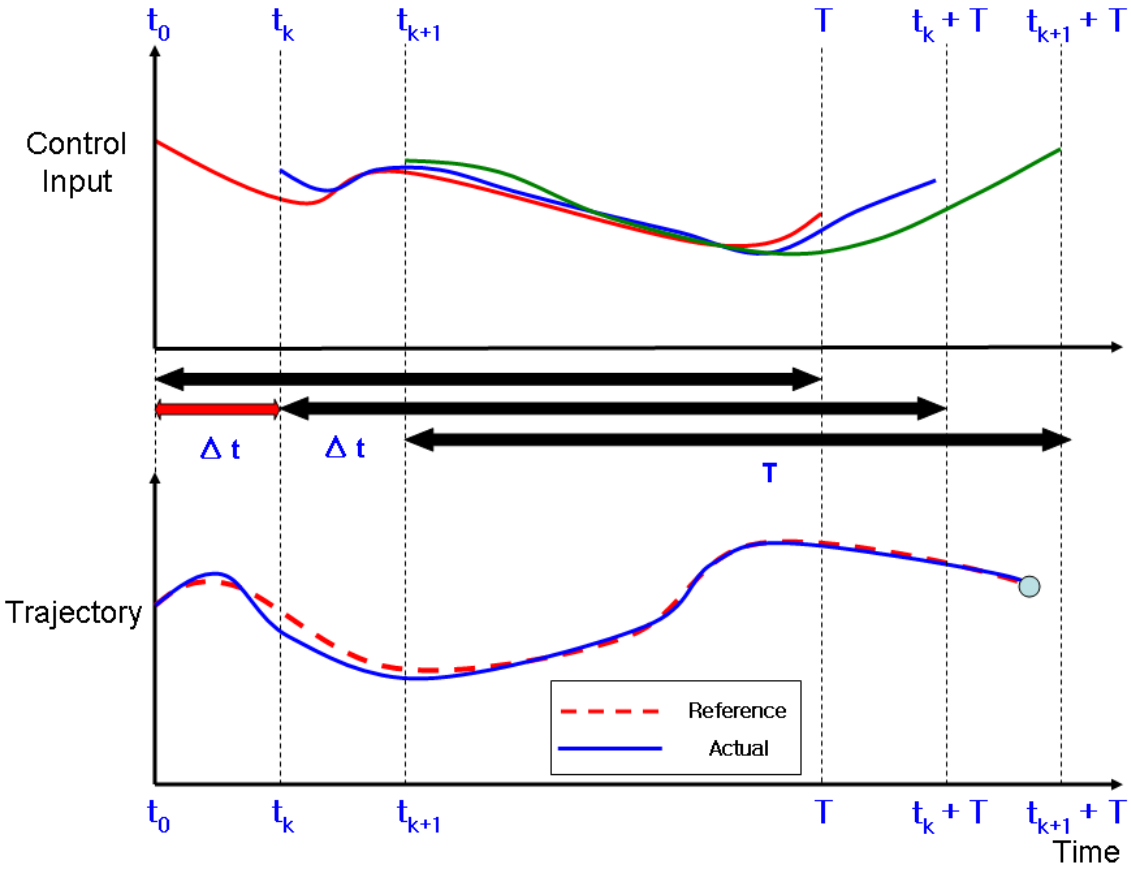


Fig. 14. Use of RHC framework to track a reference trajectory

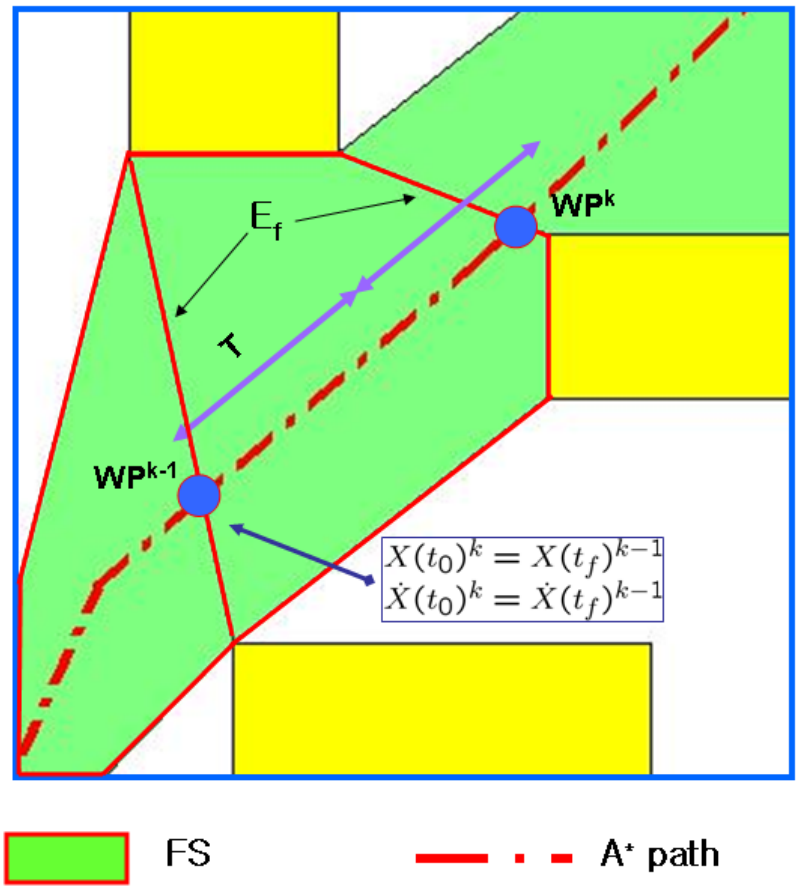


Fig. 15. Boundary conditions during the transition between FS in known environment

which generally has ample separation from the wall. This prevents trajectories being placed too close to the obstacles. From fig. 16, it can be seen that the last waypoint for the region of interest is the centroid of the last FS in order.

At current position (x_c, y_c), the vehicle would be inside of a feasible set FS_i heading for WP_i. When the vehicle or trajectory reached WP_i, a new waypoint WP_{i+1} would be immediately selected to follow in the feasible set FS_{i+1}. We formulated a fixed final time OCP for each horizon length T:

Minimize

$$J = (x(t_f) - x_f)^2 + (y(t_f) - y_f)^2.$$

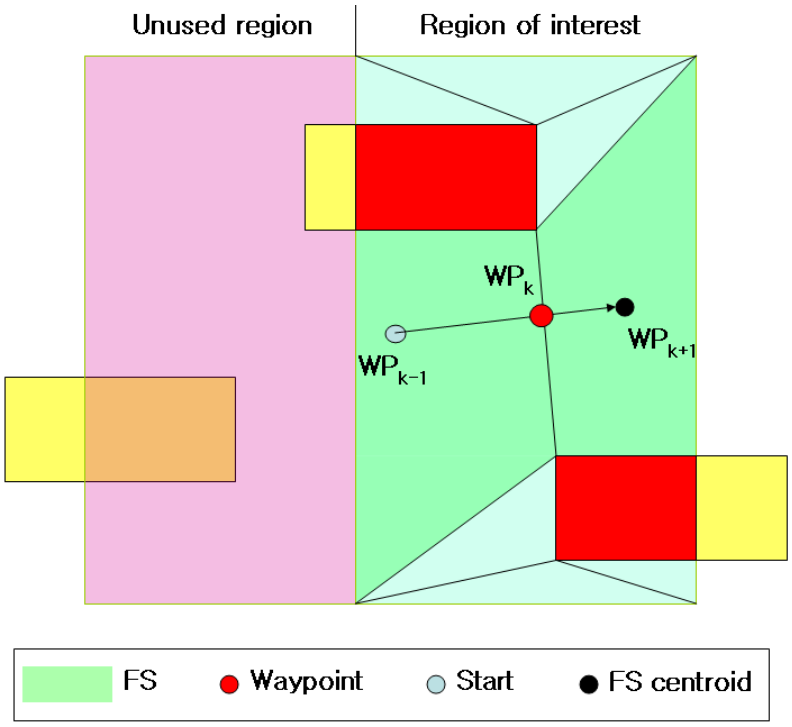


Fig. 16. Boundary conditions when generating trajectories in new FS

This formulation reflects that the cost is directly related with the proximity of the vehicle's position at final time t_f to the specified target on the convex polygon. Therefore, if x_f is not reachable, it induces a high cost.

The initial condition of one trajectory segment are

$$X(t_0)^k = (x_c, y_c)^k \quad \dot{X}(t_0)^k = (\dot{x}_c, \dot{y}_c)^k \ ,$$

The subscript c denotes current and the superscript k indicates the iteration number. To maintain trajectory smoothness, initial conditions for position and velocity were defined. The final position X_f was specified as well, whereas the velocity $\dot{X}(t_f)^k$ remained free. In the next iteration, all the final position and velocity are inherited

to the $k+1^{th}$ iteration as the initial condition.

$$X(t_0)^{k+1} = X(t_f)^k \quad \dot{X}(t_0)^{k+1} = \dot{X}(t_f)^k \ .$$

The only exceptions to this inheritance occur at q_{init} and q_{goal}.

Trajectories were planned in the interval $[t_c, t_c + T]$, where T is the planning horizon. Within a FS, if the trajectory did not reach the final position before T has passed, more iterations were executed with the same destination waypoint. Yet, the initial conditions continuously changed for new iterations to ensure a smooth trajectory.

OPTRAGEN [17] was used to obtain optimal trajectories for each planning horizon. OPTRAGEN has a built-in transcription method to NLP, so that cost and constraints, types of collocation and trajectory representation can be specified without directly coding the NLP solver.

CHAPTER IV

LOW LEVEL PLANNING

For the purpose of verification and validation of the motion planning algorithms, an experimental setup has been built. This includes a robot, vision system mounted on top of the experiment station, and a computer that collects all available information, calculates trajectories, and commands the robot.

A. System Integration

There are three components that govern the low level robot control. Fig. 17 shows a vision system with a camera that provides images and image processing software which processes the image in SIMULINK. Acquired information through the vision system is fed into the trajectory generation package software in MATLAB to create the map of the environment and execute motion planning. Depending on the condition of the environment - known or unknown - the software also uses C code. After all trajectories and controls are generated, trajectory package sends commands to the robot control software written in Java to give appropriate commands. Through wireless connections, the robot receives command inputs, then executes them. Camera again detects the robot and the feedback loop is complete to control the robot. The environment for executing different programming languages is built on MATLAB due to its compatibility with Java and ease of use.

B. Vehicle System

The vehicle used for the system is Telepresence Robot Kit (TeRK) developed by the Carnegie Mellon Robotics Institute and Charmed Labs. The heart of the system

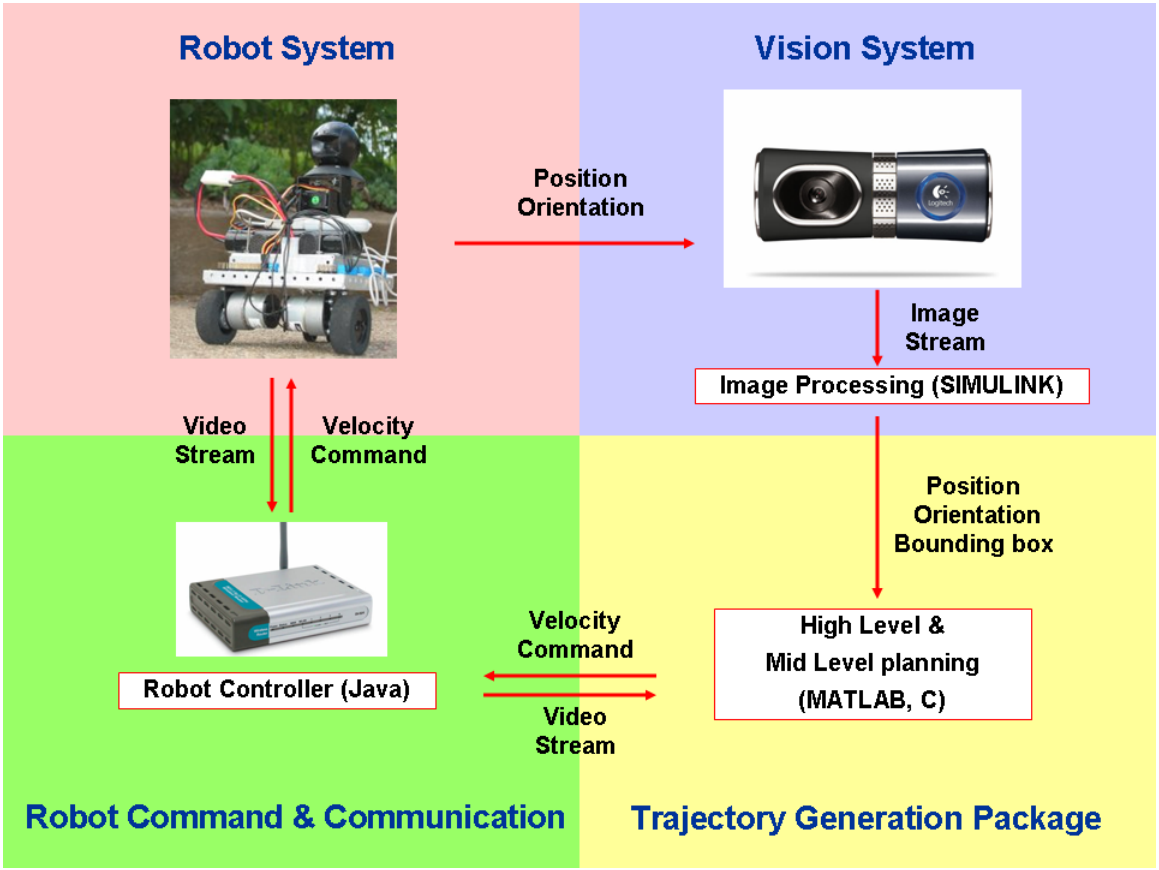


Fig. 17. Systems Overview

is the Qwerk embedded computer. It features a 200MHz ARM9 RISC processor with 32 MB of SDRAM and 8 MB flash memory. It is equipped with 4 closed-loop motor controllers, 16 RC-servo controllers, 16 programmable digital I/Os along with analog inputs, USB ports, and ethernet port. Two independent motors drive the robot and they are individually controlled, able to make turns of any radius. Its wireless networking capability allows the robot to move without tether or attachment to any object while receiving commands and transmitting information. The Logitech Quickcam STX mounted on the robot can provide real-time video stream to the computer.

C. Vision System

The primary purpose of an overhead vision equipment is to detect the obstacles and moving robot in the plain field. To conduct motion planning in a static and known environment, obstacles must be detected once to provide the accurate map. In a dynamic or unknown environment, the map must be updated to account for changes in the map, making detection and mapping usable for a limited duration of time.

The vision system is equipped with the Logitech QuickCam Ultra Vision. The resolution was set at 320 x 240 with the frame rate of 30 fps. Its wide angle lens allows for larger coverage area of the experimental environment compared to conventional webcams. The localization algorithm was built using SIMULINK video processing blocksets and default blocksets. Since all trajectory generation process is executed in MATLAB environment, conducting image processing in the same environment could eliminate the hassle of changing data types and other work. OpenCV is also a powerful tool to do the same job, yet the SIMULINK featured all necessary operations for this setup.

Here's the explanation of the algorithm. First, an image of a plain view of the experimental field is taken. Another image is taken when obstacles and the vehicle are placed in the map. Two images are subtracted from each other to find the difference. Note that both images are converted to intensity images (showing black and white). Therefore, the more contrast objects retain with respect to the background, the easier it is to detect the object. The floor was covered with white sheet, so that all obstacles and vehicle must carry non-reflective black markers on their top. After the difference has been found, autothreshold feature uses Otsu's method to convert the intensity image into a binary image. Before the new image is analyzed, morphological structuring elements are created. This effectively detects the neighborhood pixels with the binary number '1' and imposes a user-defined shape onto the area. Square has been selected for this thesis. These regions are called the region of interest (ROI). Blob analysis blockset takes ROIs and process them to find out their properties such as area, centroid, major-axis, etc. Area, centroid, and bounding box provide enough information to provide identifications, coordinates, and size of the objects in the post-processing stage of image processing. The implementation of the vision system on SIMULINK is shown in figs. 18 and 19.

The maximum number of blobs are specified, say 10. Blobs are then sorted from the largest to the smallest while their indices in the array are remembered. Indices are utilized to extract the centroid and bounding box coordinates for the corresponding blob areas. Orientation of the robot is found by attaching two differently sized (consecutively sized) black masks on the robot. By using two blobs, the vector between these can be found, hence the orientation is recovered.

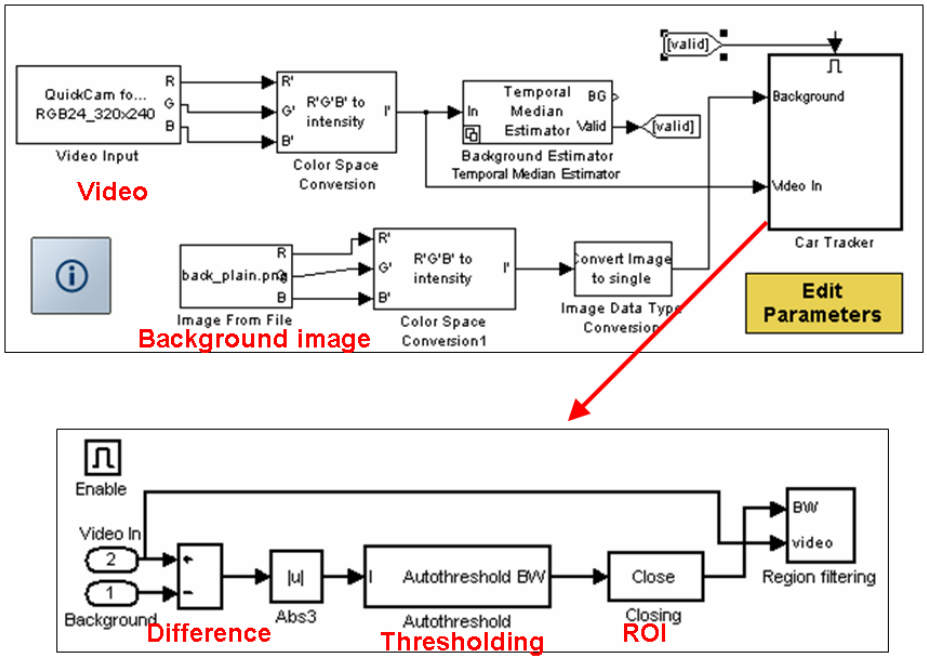


Fig. 18. First two steps of image processing: First block diagram receives background image and video frame, then converts them to intensity images. The second block subtract the images and obtain region of interest

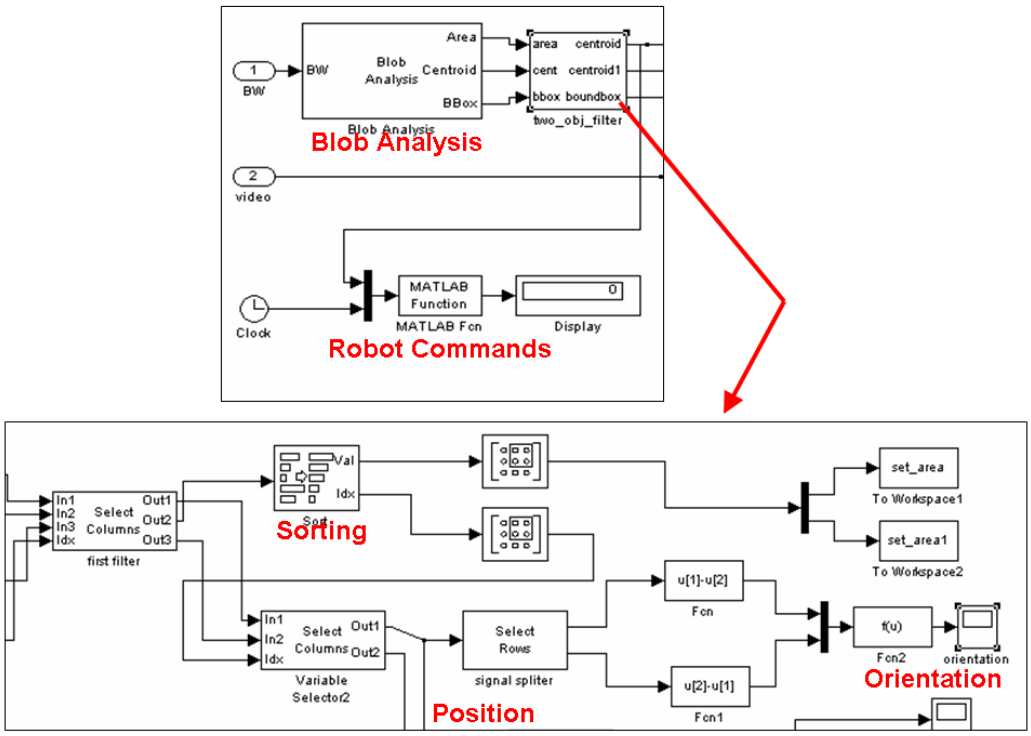


Fig. 19. Last two steps of image processing: Blob analysis is carried out and position and orientation is obtained through the lower block diagram. Then, the information is sent to the robot controller where the sequence of commands are determined

D. Dynamical Models

Several dynamical models have been utilized to generate simulations to study the feasibility of creating dynamically feasible trajectories.

1. TeRK

TeRK is the primary vehicle used in the experimental setup. It is a differentially driven vehicle. The dynamics of the vehicle is described as following:

$$\dot{x} = \frac{r}{2}(u_l + u_r)\cos\theta$$
$$\dot{y} = \frac{r}{2}(u_l + u_r)\sin\theta$$
$$\dot{\theta} = \frac{r}{L}(u_r - u_l).$$

Constants are

$$L = 5.5\ in$$
$$r = 1\ in$$
$$u_l = 6\ in/s$$
$$u_r = 6\ in/s$$

where r is the radius of the wheel, u_l and u_r are the speed of the left and right motors, and L is the distance between wheels. This system is differentially flat, therefore instead of directly parameterizing x, y, u_l, u_r, we described the system in terms of flat outputs:

$$z_1 = x \quad z_1^{(1)} = \dot{x} \quad z_1^{(2)} = \ddot{x}$$
$$z_2 = y \quad z_2^{(1)} = \dot{y} \quad z_2^{(2)} = \ddot{y}$$

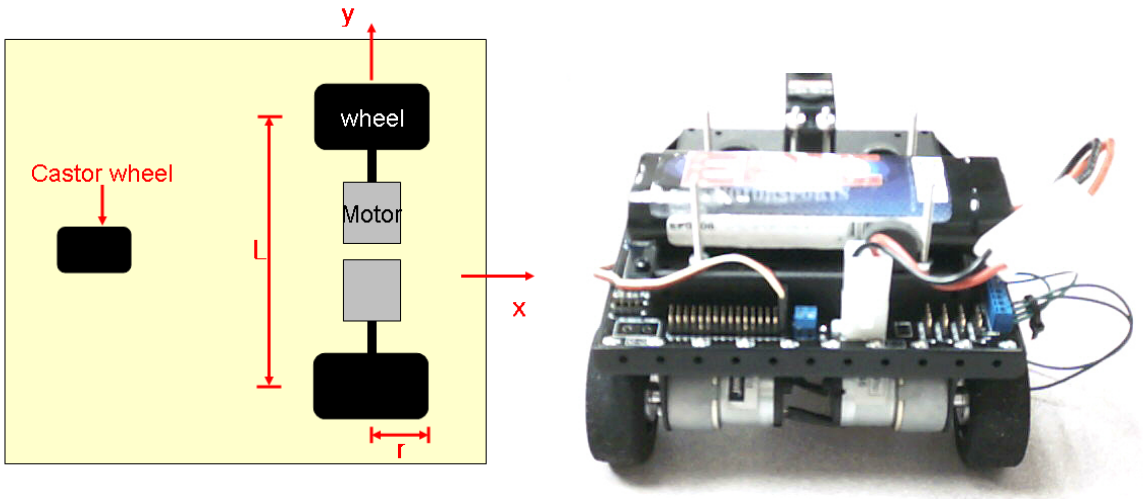


Fig. 20. Bottom view and the picture of the TeRK: Two motors and two wheels along with other ports on top of the board are visible

This parametrization allowed for an algebraic expression for θ and wheel velocities u_l and u_r.

$$\theta = tan^{-1}(\tfrac{\dot{y}}{\dot{x}}) = f_1(z_1, z_2, z_1^{(1)}, z_2^{(1)})$$
$$u_l = \dot{x}(1+\dot{y}^2/\dot{x}^2)^{1/2} - 1.5(\dot{x}\ddot{y} + \dot{y}\ddot{x})/(\dot{x}^2+\dot{y}^2)$$
$$= f_2(z_1, z_2, z_1^{(1)}, z_2^{(1)}, z_1^{(2)}, z_2^{(2)})$$
$$u_r = \dot{x}(1+\dot{y}^2/\dot{x}^2)^{1/2} + 1.5(\dot{x}\ddot{y} + \dot{y}\ddot{x})/(\dot{x}^2+\dot{y}^2)$$
$$= f_3(z_1, z_2, z_1^{(1)}, z_2^{(1)}, z_1^{(2)}, z_2^{(2)})$$

The schematic and picture of TeRK (fig. 20) shows the setup of the wheels and motors underneath the robotic platform

2. Multi-Vehicle Wireless Testbed Model

It is a 3 degree of freedom vehicle with translations in x,y and rotation θ. There are two fans on the port and starboard side of the vehicle that produce up to 4.5 N of force. For further information on the testbed, readers are referred to [22]. We exploited its differential flatness to simplify its dynamics. The dynamics of the vehicle

is described as following:

$$m\ddot{x} = \eta\dot{x} + (F_R + F_L)\, cos\theta$$
$$m\ddot{y} = \eta\dot{y} + (F_R + F_L)\, cos\theta$$
$$J\ddot{\theta} = \psi\dot{\theta} + (F_R - F_L)\, r_f \ .$$

Constants have been specified as

$$m = 5.05\ kg$$
$$J = 0.05\ kgm^2$$
$$r_f = 0.132\ m$$
$$\eta = 4.5g\ kg/s$$
$$\psi = 0.064g\ kgm\ ,$$

where g denotes gravitational acceleration. The flat outputs were:

$$z_1 = x \quad z_1^{(1)} = \dot{x} \quad z_1^{(2)} = \ddot{x}$$
$$z_2 = y \quad z_2^{(1)} = \dot{y} \quad z_2^{(2)} = \ddot{y}$$

This parametrization allowed for an algebraic expression for θ and forces F_L and F_R.

$$\theta = f_1(z_1, z_2, z_1^{(1...4)}, z_2^{(1...4)})$$
$$F_R = f_2(z_1, z_2, z_1^{(1...4)}, z_2^{(1...4)})$$
$$F_L = f_3(z_1, z_2, z_1^{(1...4)}, z_2^{(1...4)})$$

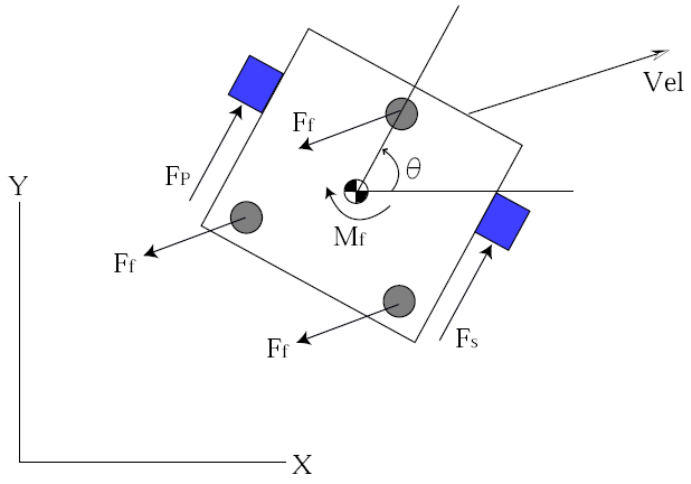


Fig. 21. Caltech MVWT: Thrust of the fans and friction from the wheels

3. Dubin's Car

Dubin's car best represents the kinematics of automobiles. It has limitation of turning radius due to the steering angles and the vehicle dimensions [23].

$$\dot{x} = ucos\theta$$

$$\dot{y} = usin\theta$$

$$\dot{\theta} = \frac{u}{L}tan\phi.$$

Constants are

$$L = 4.5\ in$$

$$u_l = 4.5\ in/s$$

$$u_r = 4.5\ in/s$$

E. Robot Controller

A Java object called "myRobot" is created from the SimpleRobotclient class. This class contains functions that govern the actions and movements of the robot as well as other simple tasks. It also creates a GUI client, where the robot can be connected

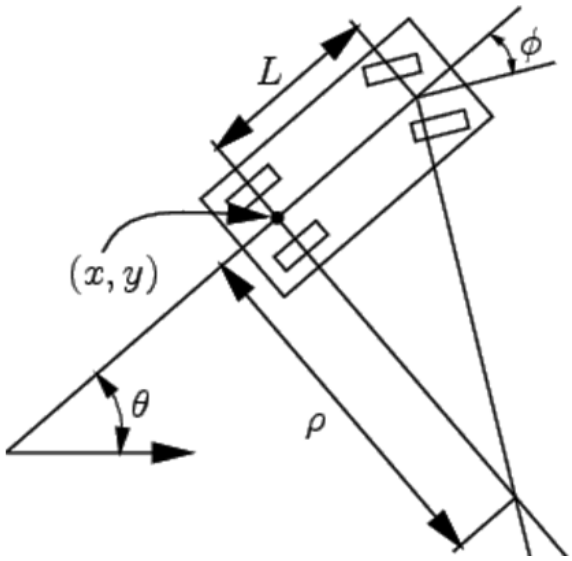


Fig. 22. Dubin's car dynamics taken from [4]

wirelessly to the computer. After the connection is established, instead of running a compiled code prescribed to do certain tasks, commands are given each time the position and the orientation of the robot from the image processing unit. After calculating the robot position error, MATLAB subroutine connected to the SIMULINK can direct the robot back to the reference trajectory by giving velocity commands on the differential motors.

F. Tracking Results

Since the robot system identification has not been completed and the localization system only relies on the vision system, accurate measurements of robot movement and how the motor reacts to the commands are difficult to obtain. Therefore, a simple control law must be devised to facilitate the immediate need for a controller. The vehicle was allowed to track a simple reference trajectory with two commands. These are simple "turn left" and "turn right" control commands where the robot is turning

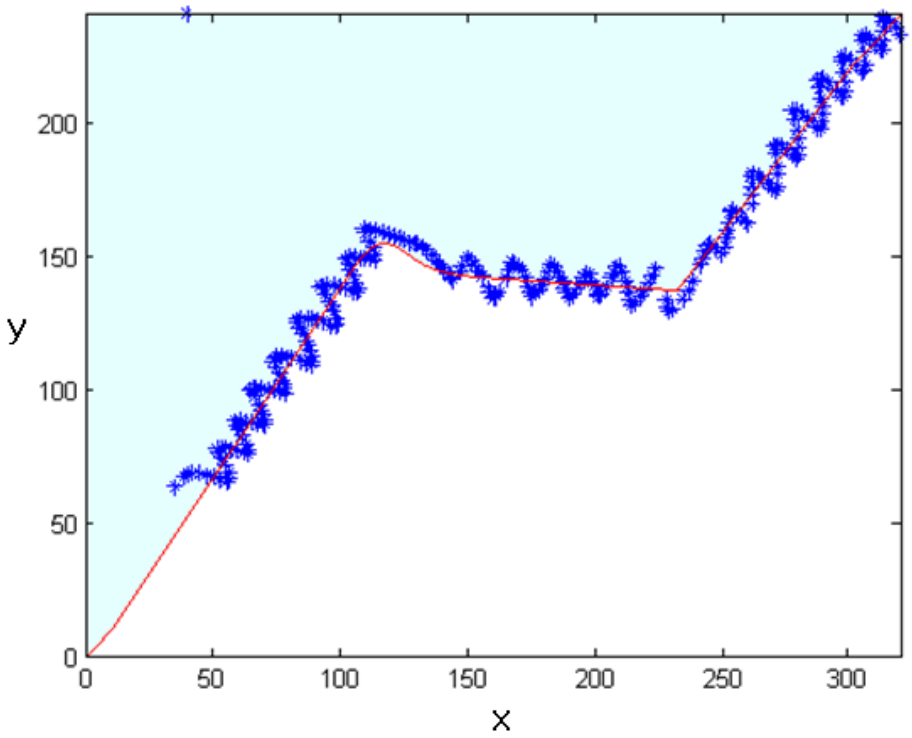


Fig. 23. TeRK robot trajectory tracking: With a primitive controller, the robot could still track a simple trajectory

at a specified rate towards the direction. As shown in fig. 23, the robot was able to track the trajectory with a tendency to overshoot.

CHAPTER V

RESULTS

This section demonstrates the application of multi level motion planning to dynamical systems.

A. Known Environment

All obstacles were assumed to be stationary and non-changing. Starting point is always near the lower left corner of the map and the motion planner must navigate the vehicle through the obstacles to reach goal in the upper right corner of the maps. Originally, the map was designed to provide pixel by pixel value for the vision system, therefore its size is 320 by 240. Considering the real experimental plain field size of 12 ft by 8 ft, the speed was scaled, so that the maximum speed of the robot at 6 in/s would be expressed as 14 pixels/s. The maximum speed for the dubin's car was arbitrarily set to 10 pixels/s, equivalent to about 4 in/s.

In fig. 24, blue curve in the left subfigure is the trajectory. We can see that wheel speeds u_l and u_r on the TeRK do not exceed the 14 units in the lower figure. u_r goes below 0 for less than a second. TeRK is capable of having negative values for the motors, meaning they are running backwards. However, in this simulation, $0 < u_l, u_r < 14$. If the speed of a motor is 0, TeRK is turning with the static motor as a pivot point.

In a narrower environment such as in fig. 25, TeRK was able to avoid obstacles without violating any control constraints. In contrast, dubin's car struggled to keep the steering angle under 45 degrees and it exceed the control limits by twice the amount in two instances shown as 2 peaks in fig. 26. Those instances are when the car makes a hard left turn and then a right turn to enter and exit the cluttered area.

Not only hard turns contribute to this error, but the formulation of the ϕ angle can affect the result. If the signs of $\dot{x}$ and $\dot{y}$ are different due to slight infeasibility of the first derivatives falling under zero, it will create extreme ϕ values. The trajectory generated for MVWT show no violation of constraints in fig. 27; F_L and F_R remained under 4.5 N. Unlike other vehicles, MVWT obtained very smooth trajectory. We can also observe that there is a time period with low F values. This occurs during the narrow pass because the narrow pass is a small area, but a whole FS is assigned to the region to account for the sharp turns. Having a constant horizon length T, the planner slows down the MVWT by exerting less force than in other regions of the map where the traveling distance is long.

B. Unknown Environment

To demonstrate the planning in partially known environment, 8- connected LPA* was used. 4-connected LPA* can also be used, however it lacks the capability to direct the vehicle, since the ties are broken arbitrarily. Going through a L shape trajectory from upper left corner to the bottom right incurs the same cost as taking the diagonal way because 4-connected graph cannot support diagonal movement. It can do zigzag movement to imitate a diagonal trajectory, yet the cost is essentially the same with L shape movement.

The motion planning is done on a 50 by 50 map. This is a smaller map than the one for the known environment, so that the TeRK wheel velocity is scaled down to nearly 3.5 pixels/s. The speed for dubin's car remained at 10 pixels/s. To generate trajectories with simulation, obstacle map was given to the trajectory planner for a specific region that the sensor was able to detect. In all scenarios, sensor horizon was set at 10 grids. On the figures, after the vehicle reaches a location where the goal is

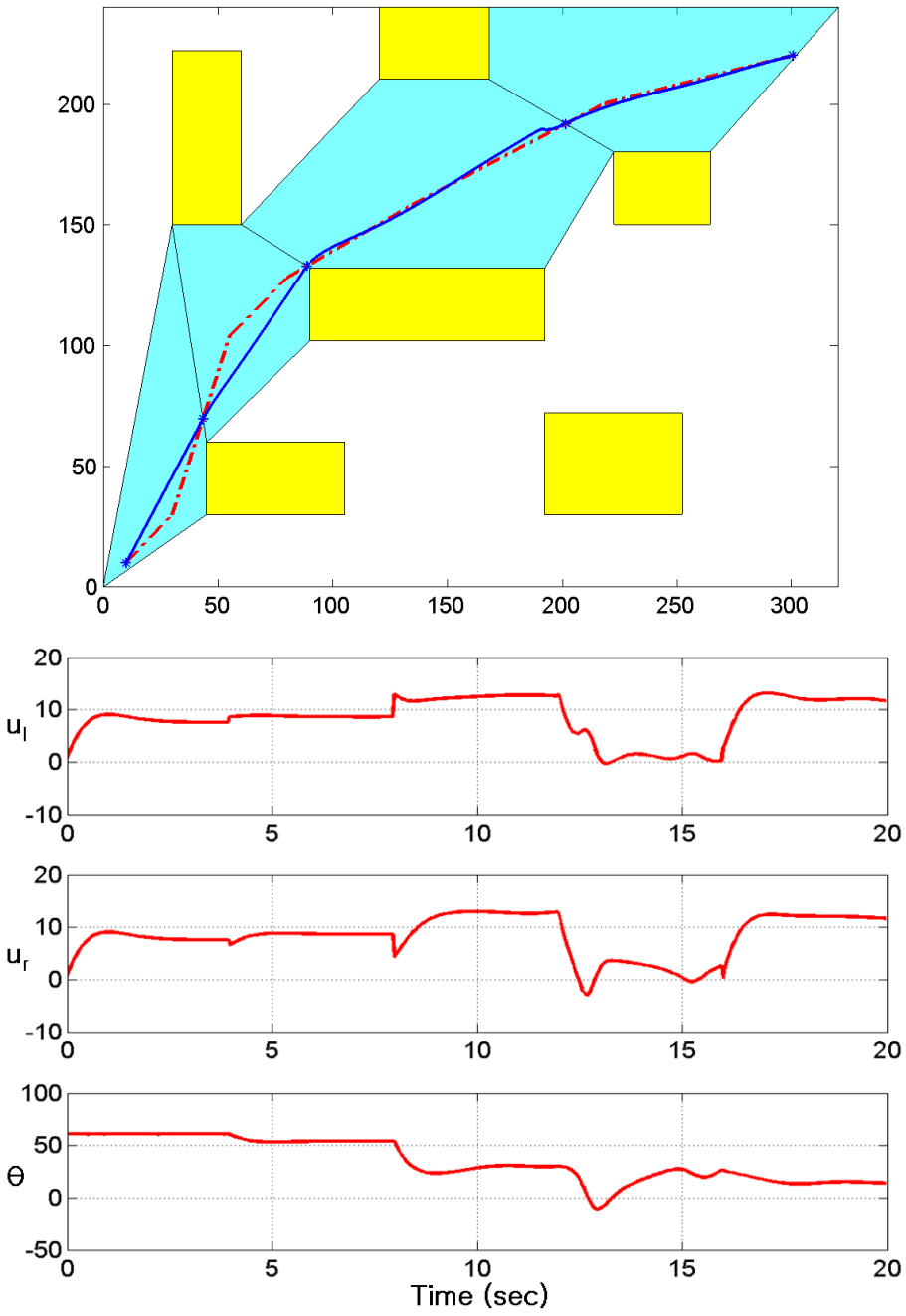


Fig. 24. Known environment map 1 with TeRK. T = 4 sec

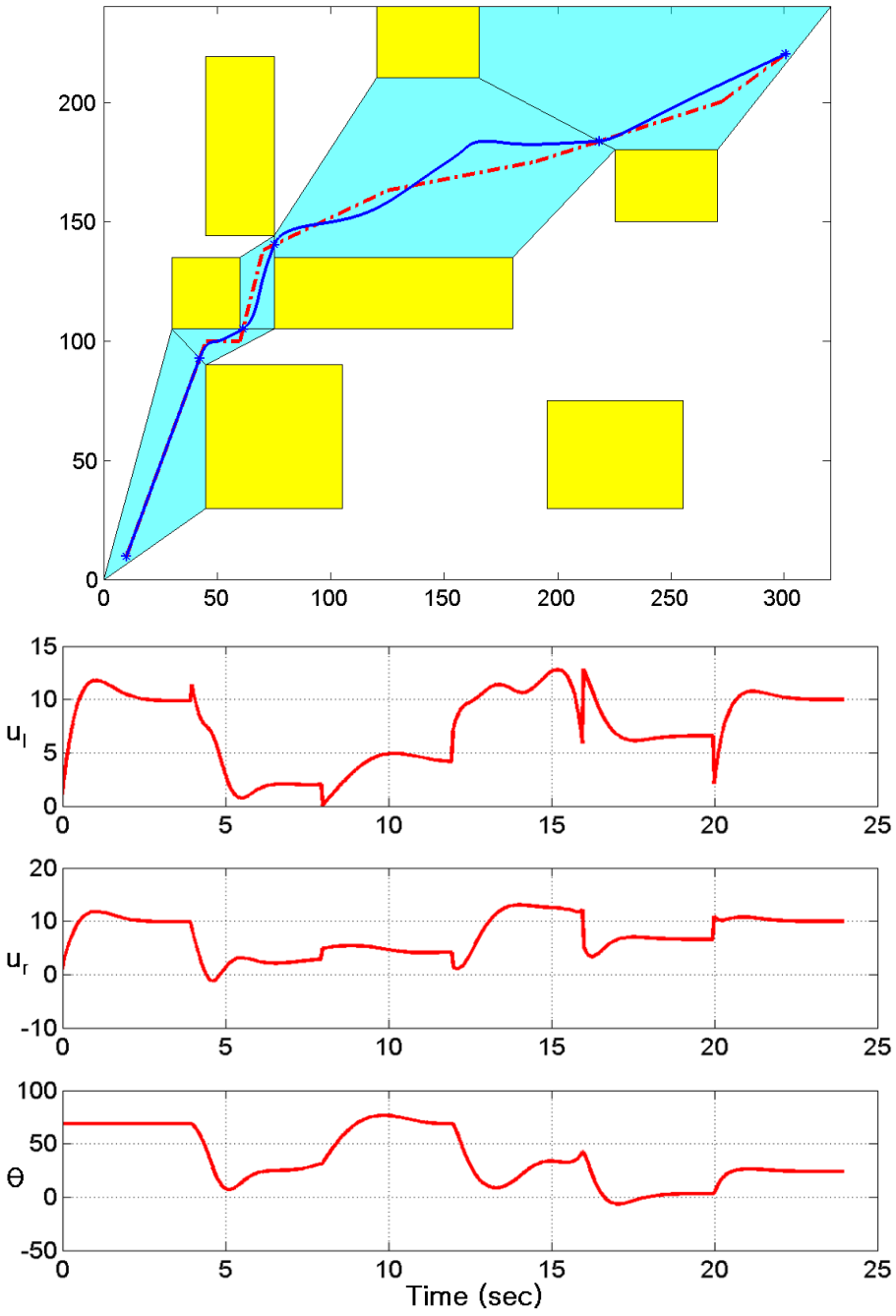


Fig. 25. Known environment with TeRK, T = 4 sec

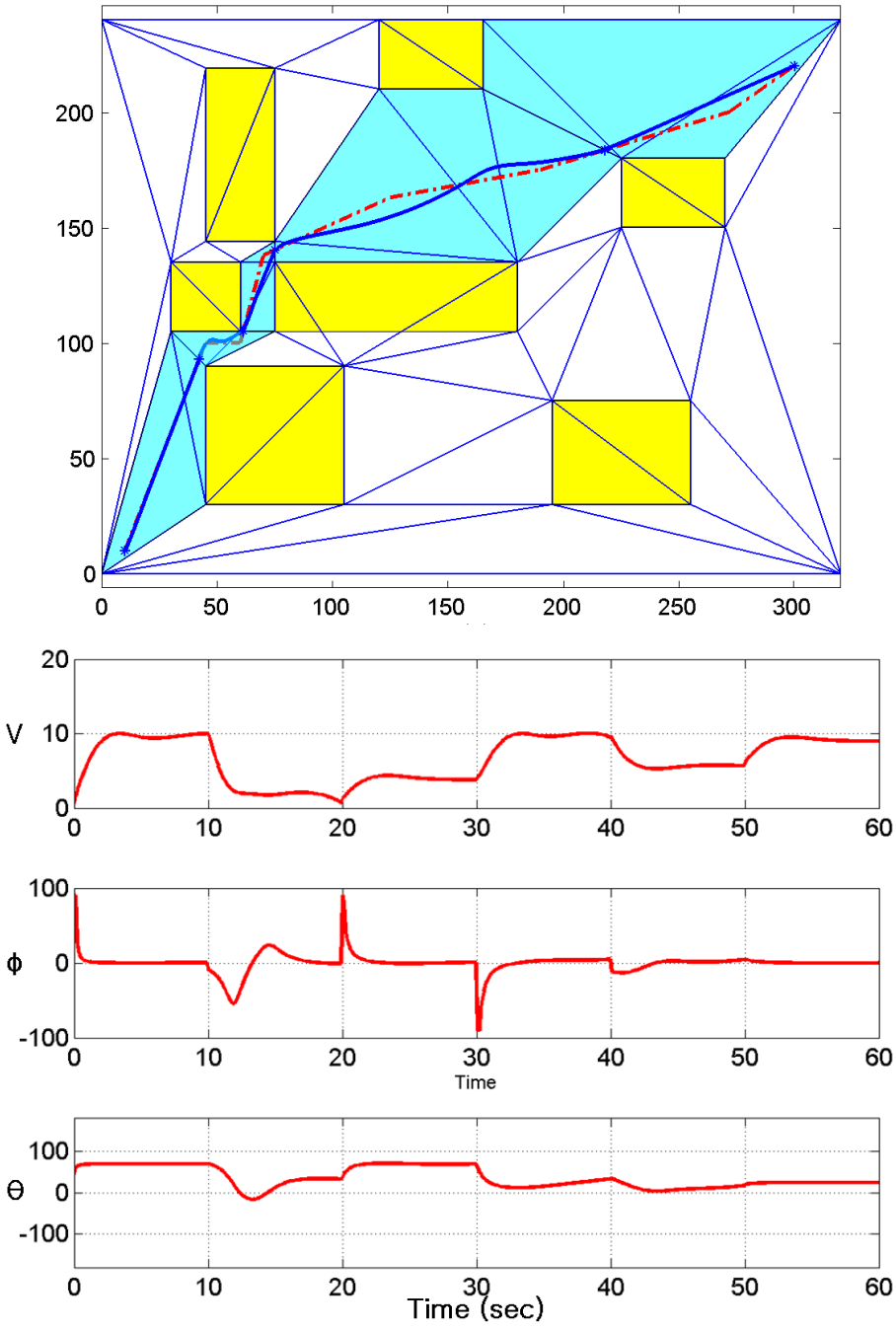


Fig. 26. Known environment with dubin's car, T = 8 sec

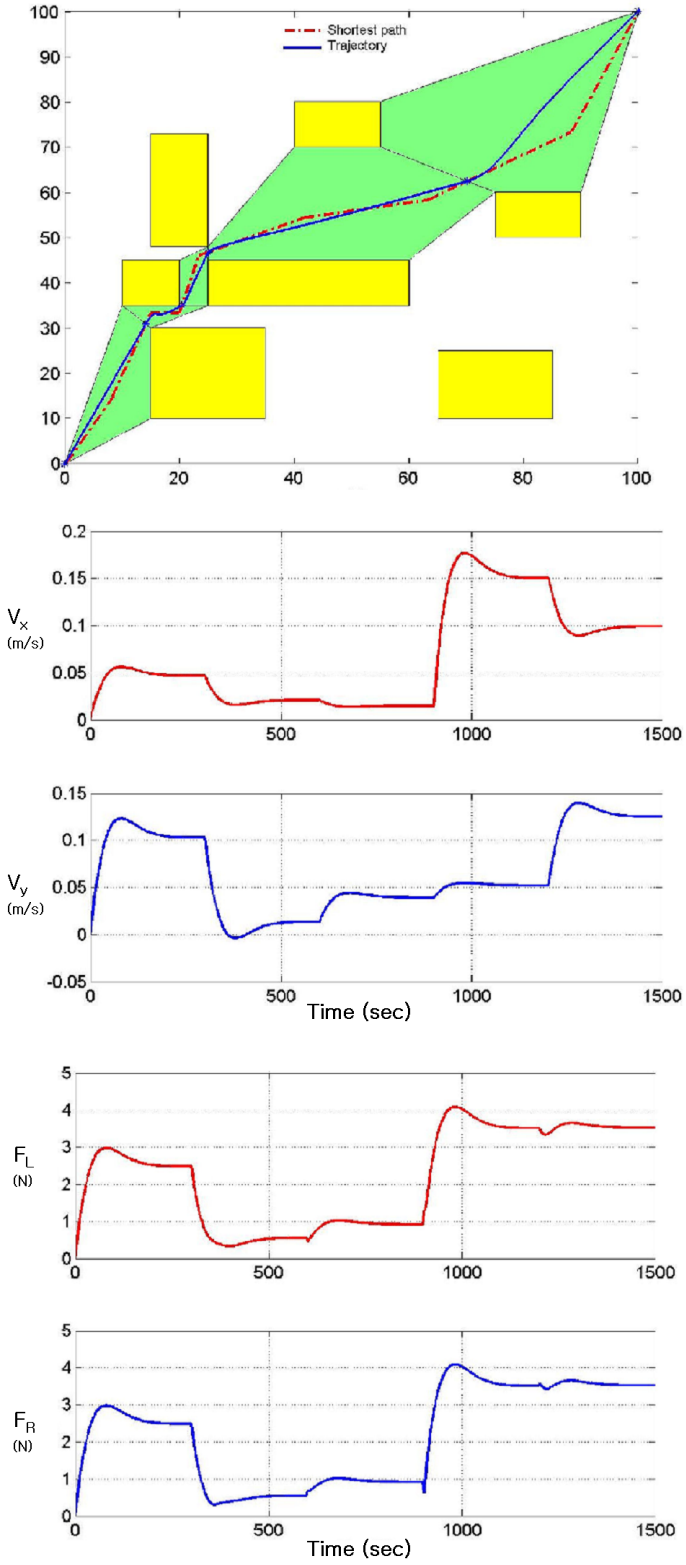


Fig. 27. Known environment MWVT, T = 300 sec

visible and the feasible set includes or touches the goal (marked as a yellow grid), the trajectory generation finishes.

Next two figures show trajectories for TeRK and the dubin's car. In fig. 28, velocity bounds are satisfied for TeRK. Dubin's car also generated obstacle free path while keeping the controls in the boundary. The maximum angle near 45 degree is achieved near the middle obstacle in the upper region as shown in fig. 29. In general, the trajectory is straight to the goal, minimizing the risk of intruding other obstacles.

A different map was utilized to demonstrate another planning episode in figs. 30 and 31. Sharp corners of the trajectory are shown in both TeRK and dubin's case when entering the cluttered area and making the turn for the goal after exiting the area. However, the first turn right turn towards the obstacles cannot be found on the control profile whereas the steep left turn after exiting is clearly seen with ϕ overshoot. This instance is also visible in the velocity profile due to the sudden drop near 2.5 second mark, yet the steering is nowhere to be found. The most probable cause of the control disappearance is from failing to achieve the continuity of the velocity as specified in the OCP and OPTRAGEN. Even though the initial condition for position and velocity was given, the initial velocity could have been ignored and be set to zeros. This can explain several dips towards zero for u_l and u_r at almost every 5 seconds (if not, at times that are multiples of 5).

The final map demonstrates similar characteristics with the previous. For the dubin's case in fig. 33, 90 degree right turn in the middle of the trajectory is not shown in the control and only three significant left turns are recorded on the control. In fig. 32, both u_l and u_r have negative velocities from 22 to 25 second interval. However, it is difficult to verify whether there was any backward motion.

All the results in the unknown environment retained issues with discontinuous velocity vector from one FS to another. This can be compensated by implementing

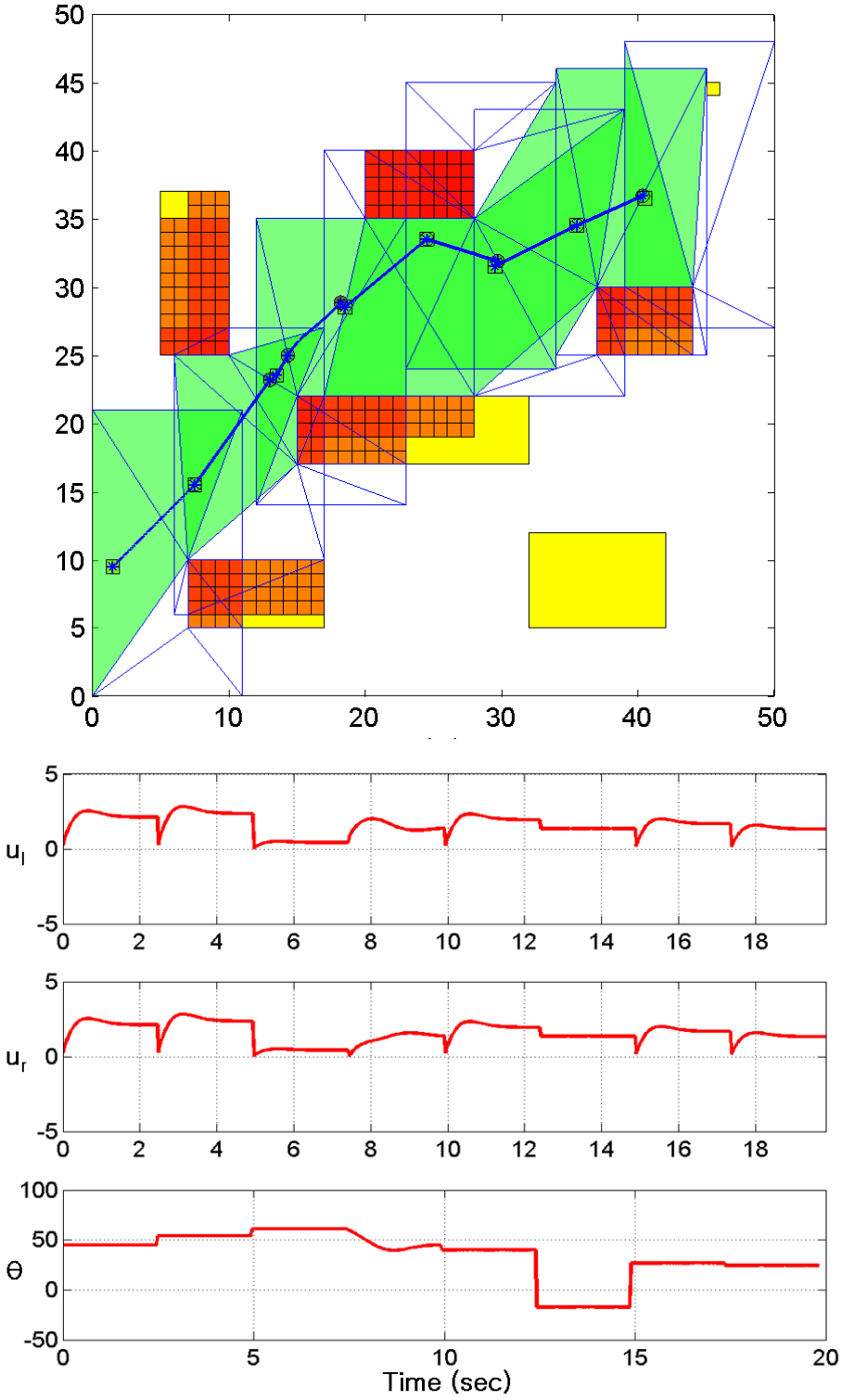


Fig. 28. Unknown environment with TeRK, T = 5 sec, Map A

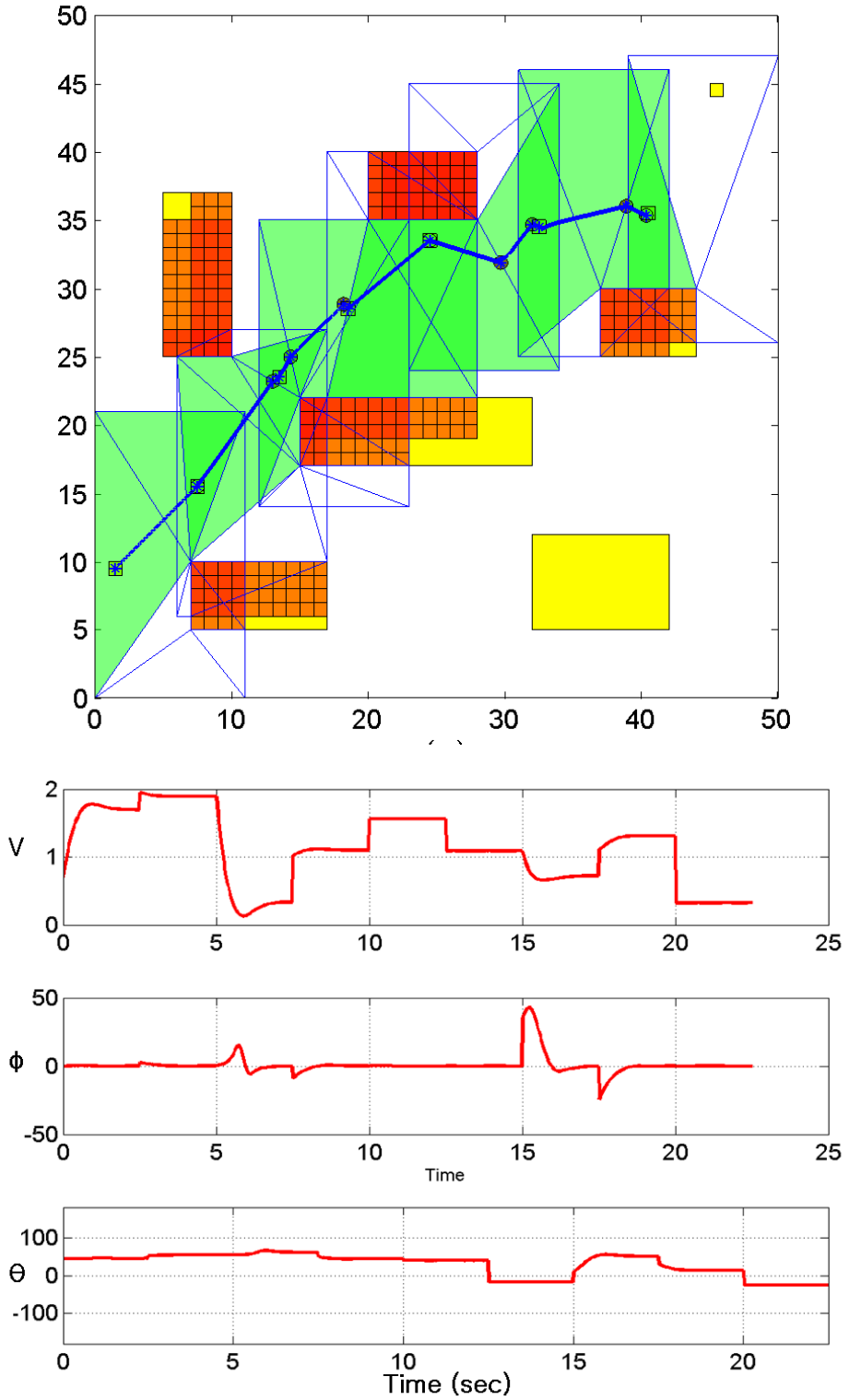


Fig. 29. Unknown environment with dubin's car, T = 5 sec, Map A

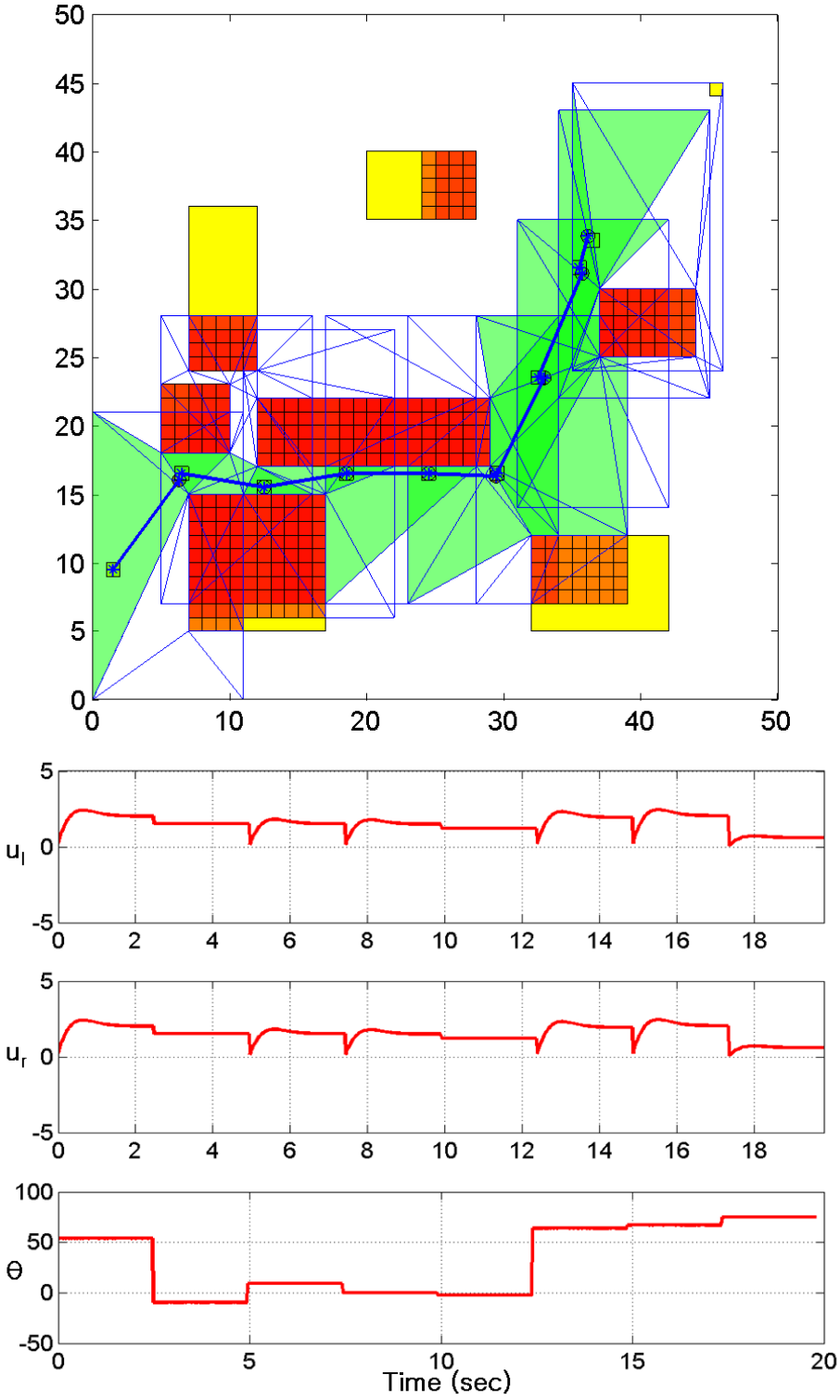


Fig. 30. Unknown environment with TeRK, T = 5 sec, Map B

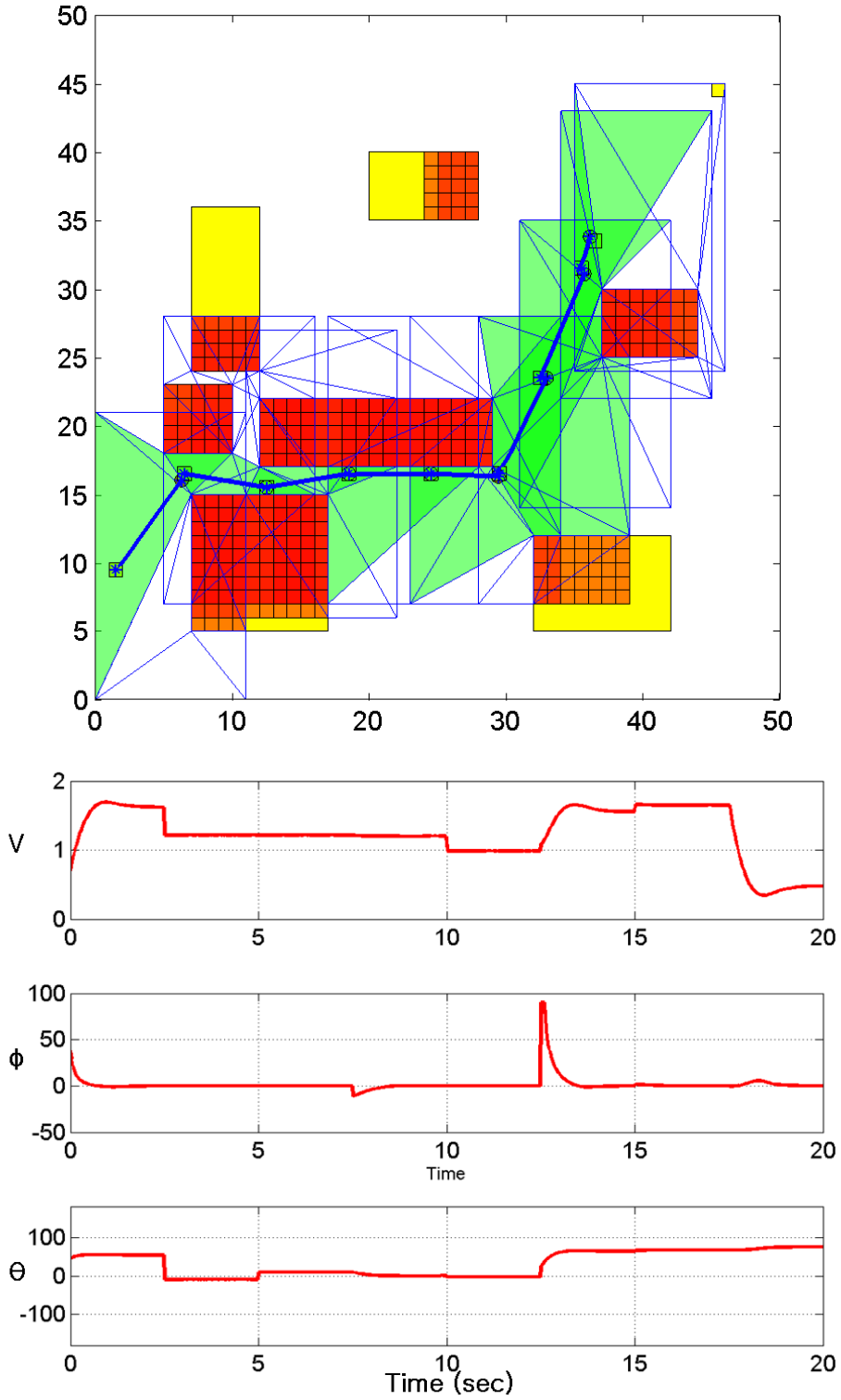


Fig. 31. Unknown environment with dubin's car, T = 5 sec, Map B

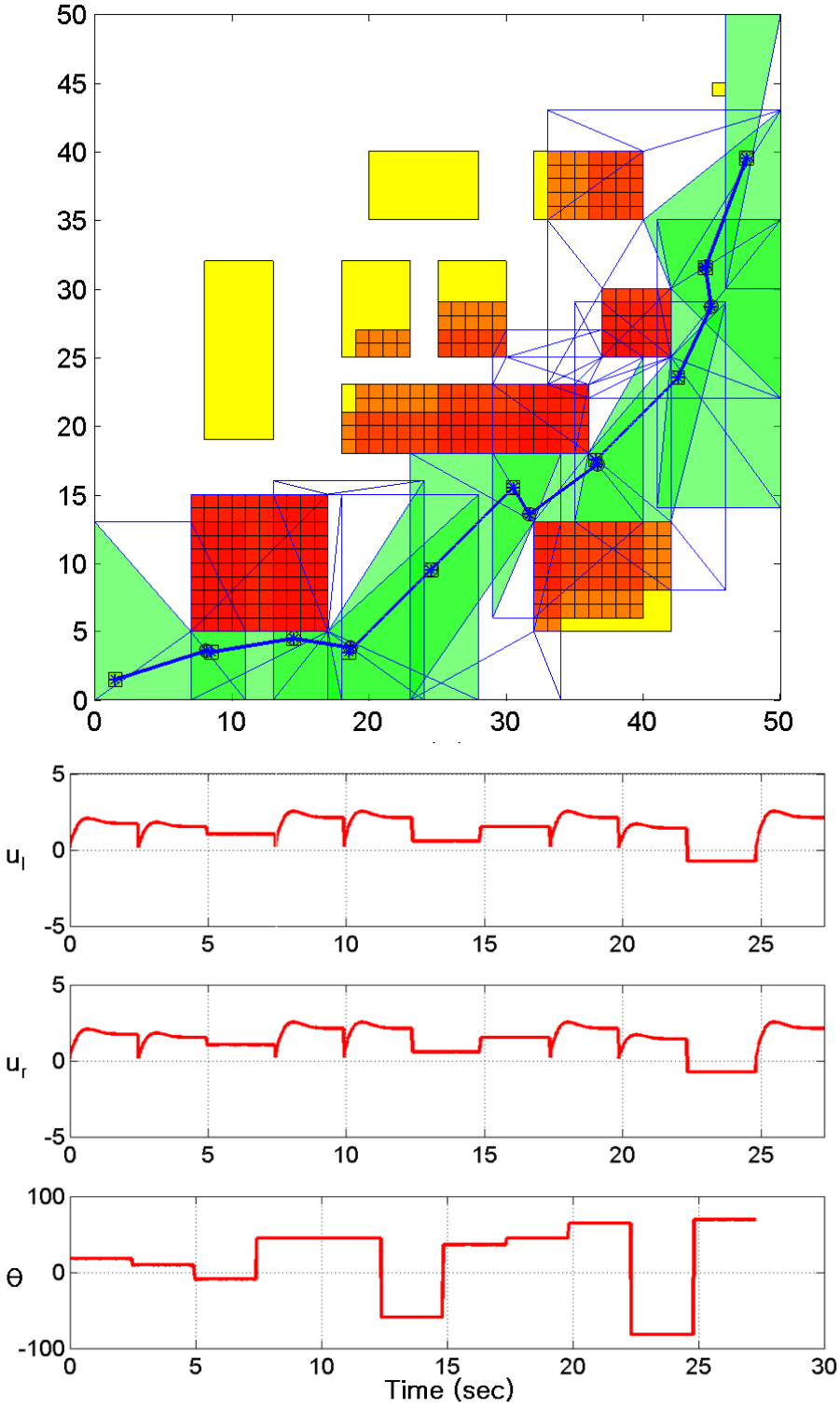


Fig. 32. Unknown environment with TeRK, T = 5 sec, Map C

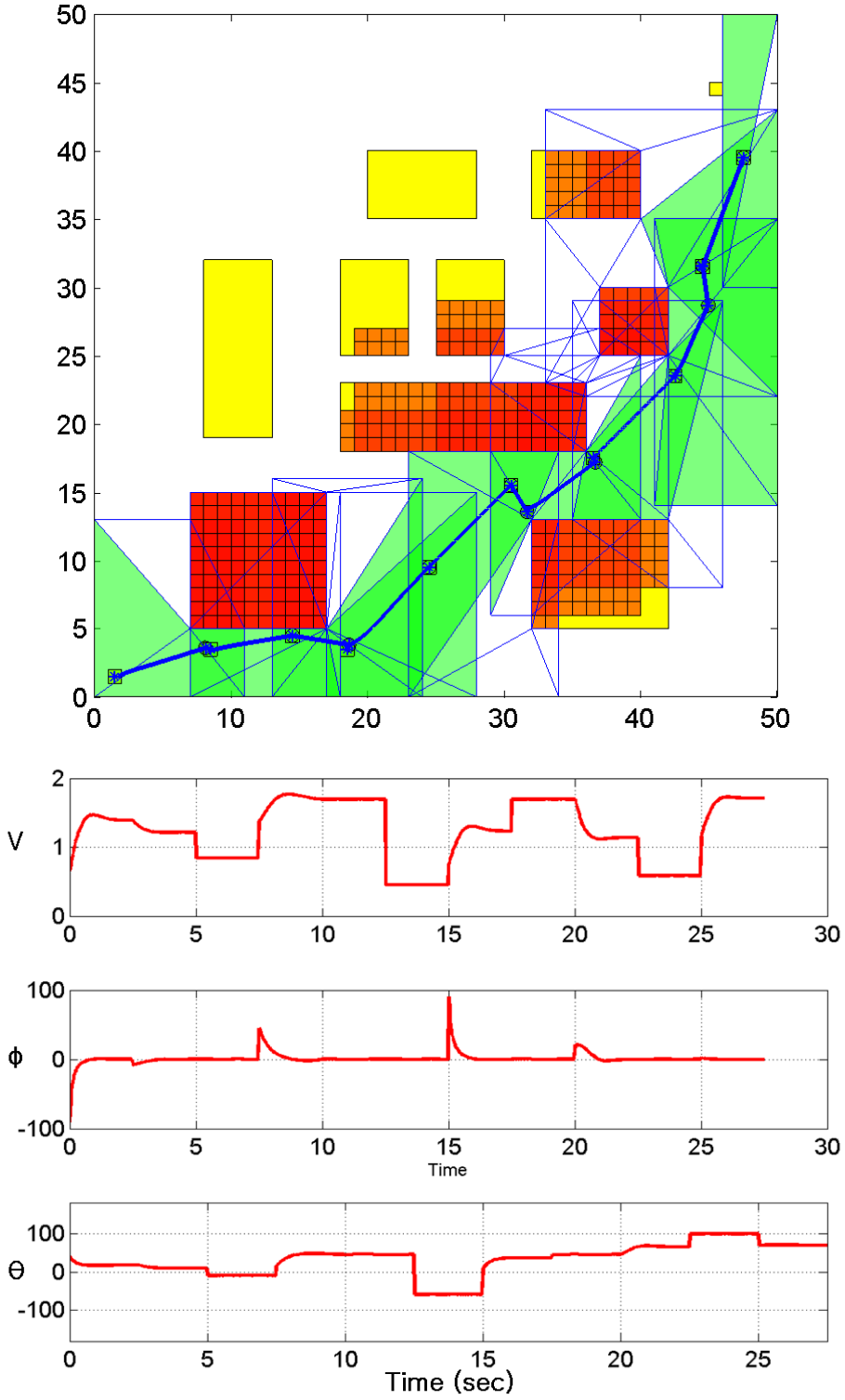


Fig. 33. Unknown environment with dubin's car, T = 5 sec, Map C

smoothing techniques locally where problems occur. Then, the vehicle can track more refined trajectory and be able to reach the goal without experiencing infeasible controls.

CHAPTER VI

CONCLUSION

This thesis has presented a multi-layer approach to reduce the complexity of motion planning by separating the problem into two parts: obtaining simplified obstacles and generating a feasible region in obstacle rich environments and solving for optimal control problem to create dynamically feasible trajectories. Feasible sets were successfully created inside the obstacle free region for known and unknown environment. Continuous time trajectories within the bounds of the feasible sets were also obtained using optimal control theory. Application of the approach to vehicle models show successful trajectory generation without violating constraints. The major advantage of this approach is the significant improvement in the computational efficiency due to the reduction in the complexity in path constraints and dynamics.

Further investigation to devise more computationally efficient algorithms for high level and mid level planner will enhance the performance of this approach. In addition, the experimental setup will fully acquire the capability to control the TeRK platform with system identification and control laws to track trajectories and follow command sequences. Stability analysis of motion planning in the RHC framework will provide insights on how to effectively build trajectories to satisfy all constraints.

REFERENCES

[1] T. Lozano-Perez and M. A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, 22(10):560-570, 1979.

[2] K. Kedem, M. Sharir, "An Efficient Algorithm for Planning Collision-Free Translational Motion of a Convex Polygonal Object in 2-dimensional Space Admist Polygonal Obstacles," in *Proceedings of the First Annual Symposium on Computational Geometry*, pp.75-80, 1985.

[3] J.C. Latombe, *Robot Motion Planning*. Norwell, MA: Kluwer Academic Publishers, 1991.

[4] S. M. Lavalle, *Planning Algorithms*. Cambridge, UK: Cambridge University Press, 2006.

[5] P. E. Hart, N. J. Nilson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions of Systems Science and Cybernetics*, 4(2):100-107, 1968.

[6] A. Stentz, "Constrained Dynamic Route Planning for Unmanned Ground Vehicles," in *Proceedings of the 23rd Army Science Conference*, 2002.

[7] J. T. Betts, "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, 21(2):193-207, 1998.

[8] E. Frazzoli, M. A. Dahleh and E. Feron, "Real-Time Motion Planning for Agile Autonomous Vehicles," in *Proceedings of American Control Conference*, 1(1):43-49, 2001.

[9] J. Bellingham, Y. Kuwata and J. How, "Stable Receding Horizon Trajectory Control for Complex Environments," in *Proceedings of American Control Conference*, 1(1):902-907, 2004.

[10] M. E. Flores and M. B. Milam, "Trajectory Generation for Differentially Flat Systems via NURBS Basis Functions with Obstacle Avoidance," *American Control Conference*, Minneapolis, MN, June 2006.

[11] C. De Boor, *A Practical Guide to Splines*. Berlin, Germany: Springer-Verlag, 1978.

[12] D. Leven and M. Sharir, "Planning a Purely Translational Motion for a Convex Object in Two-Dimensional Space Using Generalized Voronoi Diagrams," *Journal of Discrete and Computational Geometry*, 2(1):9-31, 1987.

[13] J. O'Rourke,*Computational Geometry in C*. Cambridge, UK: Cambridge University Press, 1998.

[14] E. Dijkstra, "A Note on Two Problems in Connection with Graphs," *Numerische Mathematik*, 1:269-271, 1959.

[15] D. Ferguson, M. Likhachev and A. Stentz, "A Guide to Heuristic-Based Path Planning," in *Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling(ICAPS)*, 2005.

[16] S. Koenig, M. Likhachev and D. Furcy, "Lifelong Planning A*," *Artificial Intelligence Journal* , 155(1-2):93-146, 2004.

[17] S. Koenig, "Demonstration of Lifelong Planning A* (Java Applet), http://idm-lab.org/project-a.html, Aug 2007.

[18] D. Ferguson and A. Stentz, "Using Interpolation to Improve Path Planning: The Field D* Algorithm," *Journal of Field Robotics*, 23(2):79-101, 2006.

[19] A. Nash, K. Daniel, S. Koenig and A. Felner, "Theta*: Any-Angle Path Planning on Grids," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1177-1183, 2007.

[20] R. Bhattacharya, "OPTRAGEN: A MATLAB Toolbox for Optimal Trajectory Generation," *IEEE Conference on Decision and Control*, San Diego, CA, December 2006.

[21] K. E. Hoff III, T. Culver, J. Keyser, M. Lin, D. Manocha, "Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, 33:277-286, 1999.

[22] L. Cremean, W. B. Dunbar et al., "The Caltech Multi-Vehicle Wireless Testbed," in *Proceedings of the 41st IEEE Conference on Decision and Control*, 1(1):86-88, 2002.

[23] L. E. Dubins, "On Curves of Minimal Length with a Constraint on Average Curvature and with Prescribed Initial and Terminal Positions and Tangents," *American Journal of Mathematics*, 79:497-516. 1957.

VITA

Sung Hyun Kim was born in Seoul, South Korea. He received his B.S. in aerospace engineering from the University of Illinois at Urbana-Champaign in 2005. He joined the Department of Aerospace Engineering at Texas A&M University in August 2005 and graduated with his Master of Science in December 2007.

Address:

Department of Aerospace Engineering, 3114, TAMU,

College Station, TX 77843-3114, USA.

Chair of Committee: Dr. Raktim Bhattacharya

E-mail Address: niceamos@gmail.com