JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 6, July 2009

Motion Planning in Obstacle Rich Environments

Sung Hyun Kim* and Raktim Bhattacharya†
Texas A&M University, College Station, TX 77843

DOI: 10.2514/1.38467

In this paper we present a multi-layer approach for motion planning in obstacle rich environments. The approach is built on the principle of separation of concern which partitions the motion planning problem into multiple independent layers. This enables design space exploration at each layer. We partition the motion planning algorithm into a roadmap layer and an optimal control layer. Elements of computational geometry are used to process the obstacle rich environment and generate a set of convex feasible regions, which is then 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 plans paths at a global level using coarse representation of the environment. The optimal control layer ignores the complexity of the environment and plans paths at a mid-level using fine representation of the dynamics and the environment. In this manner a separation of concern is achieved. This decomposition enables computationally tractable methods to be developed for addressing motion planning in complex environments.

I. Introduction

IT IS expected that future air force missions will shift from open sky operations to engagements in cluttered environments, which are expected to be uncertain and dynamic in nature. Engagement in such environments, demands a high level of autonomy to increase the precision and lethality of military power, with minimal risk and collateral damage. The success of such missions is highly dependent on the effectiveness and robustness of the algorithms that navigate the vehicles in unknown and dynamic environments. In recent times Congress has mandated that by 2015, one-third of the operational ground combat vehicles should be unmanned. This mandate from Congress has led to the DARPA Grand Challenge competition and its successful completion has proved feasibility of autonomy in such applications. This push towards autonomy has sparked considerable research in the area of motion planning.

Motion planning falls in a gray area among the disciplines of computer science, control theory, and artificial intelligence, and planning in perfectly known environments has been studied extensively. The starting point of most of motion planning schemes involve definition of a configuration space (C-space), where the environment is partitioned into regions with obstacles (C_{obs}) and regions without obstacles (C_{free}) [1–4]. The obstacles in this framework are assumed to be polytopic. Once the C-space has been defined, it is discretized, and various algorithms, based on fast graph-search algorithms such as A^* [5] and D^* [6], are used for motion planning in C_{free}. This ensures that the vehicle never contacts the obstacles. Motion planning in this framework results in a roadmap, which is a topological graph, that solves the motion planning problem.

Received 8 May 2008; accepted for publication 10 April 2009.

* Graduate Student, Department of Aerospace Engineering, Texas A&M University, College Station, TX 77843, USA

† Assistant Professor, Department of Aerospace Engineering, Texas A&M University, College Station, TX 77843, USA, raktim@tamu.edu.

Common methods, used to obtain roadmap, include probabilistic roadmap method (PRM) [4] and rapidly-exploring random tree (RRT) [4]. Both these methods are sampling-based method where C_{free} is discretized and a graph is created. The PRM method solves the motion planning problem by starting from a vertex in C_{free} and creating an expanding undirected graph with closest neighboring nodes. These nodes are obtained by randomly sampling C_{free}, with a bias towards the goal vertex. This reduces the number of sampling points. PRM is provably probabilistically complete, meaning that as the number of sampled points increases without bound, the probability that the algorithm will not find a path, if one exists, approaches zero. The final roadmap is then created by a Dijkstra's shortest-path query. In RRT method, expanding trees are used to efficiently search nonconvex high-dimensional spaces. It is expected that C_{free} for motion planning problems, in obstacle rich environments, will have this property. RRT is based on incremental sampling and searching approach that incrementally constructs a search tree, and gradually improves the resolution of the solution. A dense random sequence of samples from C_{free} is taken in the construction of the tree. An RRT can be intuitively considered as a Monte-Carlo way of biasing search into largest Voronoi regions. Some variations can be considered as stochastic fractals. RRTs have the capability of incorporating nonholonomic and kinodynamic constraints. Usually, an RRT alone is insufficient to solve a planning problem. Thus, it can be considered as a component that can be incorporated into the development of a variety of different planning algorithms.

In a deterministic framework, visibility graphs, Voronoi diagram [3], and cell decomposition [4] are well-known algorithms that generate roadmaps. These methods are often not classified 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 to extract trajectories. The backbone of such methods is the A^* algorithm. It utilizes heuristics and graph search algorithms to determine the shortest path from start q_{init} to goal q_{goal}. D^* contains improvements on A^*, 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. A deterministic roadmap typically lacks smooth transitions between segments, so that it requires a post-processing for dynamically compatible trajectories. Thus, in this framework, the dynamics of the vehicle is not considered. Consequently, the motion plans thus obtained are not dynamically feasible. However, these methods are computationally tractable and thus lend themselves well to near real-time implementation.

For complex robotic systems, such as ground vehicles or unmanned air vehicles, such techniques are of limited value as the dynamics of these vehicles impose certain constraints that must be incorporated in the motion planning. Recent research has attempted to explicitly account for the dynamics of the robot at the planning stage to obtain dynamically feasible motion plans as extensions of the configuration space planning methods [7–9]. An alternate approach, that incorporates dynamics of the vehicle using optimal control problem (OCP) formulation to generate dynamically feasible trajectories, has recently received attention [10,11]. Additionally, recent research on numerical methods for implementing optimal control for trajectory optimization has generated tremendous improvement in computational efficiency and numerical stability [12], and provides adequate motivation for pursuing this approach. The work of Petit et al. [13] presents a simplification of optimal control by exploiting properties such as differential flatness. However, we note that most of the recent work under this formulation concentrates on trajectories with relatively few obstacles. Increasing number of obstacles causes more constraints to be enforced on the OCP, creating computational complexities and convergence issues. Therefore, for generating a real-time trajectory in densely populated obstacles, a monolithic formulation based on optimal control theory is prone to be infeasible.

To eliminate this shortcoming, recent papers propose motion planning with motion primitives [7]. These primitives are precomputed straight and minimum radius turn profiles which integrates the full vehicle dynamics. Determining a sequence of motion 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 may not be feasible and the path can become excessively lengthy by detouring [14]. Flores and Milam [15] presents a methodology based on separation of concern between obstacle clearance and trajectory generation. Elimination of the obstacles brings a significant reduction of OCP constraints. Differential flatness is used to additionally eliminate dynamical constraints of the OCP, reducing the complexity of OCP further.

In this paper, we present a hierarchical framework that blends the traditional motion planning ideas from the computer science community with optimal control. We model obstacles as polygons and using algorithms from computational geometry; we simplify the obstacle rich environment to obtain a set of obstacle free convex feasible regions. Within each feasible region, optimal control techniques generate local trajectories that are optimal with respect to a given cost function, while satisfying the dynamics of the vehicle. Formulating the motion planning problem in this hierarchical manner reduces complexity at each layer and results in the design of computationally efficient, real-time implementable algorithms, and dynamically feasible trajectories.

II. Problem Formulation

This paper concentrates on the problem of motion planning in the presence of many obstacles, satisfying the following requirements: 1) avoids obstacles, 2) is dynamically feasible, and 3) is optimal with respect to a performance index (control effort, distance, time, etc).

In this paper, we restrict ourselves to two-dimensional environments, but the technique presented here can be extended to plan motion in three dimensions as well. Extension of this algorithm to three dimension and the associated growth in complexity is discussed later in Sec. III.C.

The motion planning problem addressed here is described as follows. A mobile agent 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* representing static obstacles.

The goal of this research is to design and implement motion planning algorithms in a hierarchical manner as shown in Fig. 1. At the global level, the dynamics of the environment are represented at a low spatial and temporal resolution, using coarse discrete variables. The vehicle dynamics is completely ignored at this level. The output of the global planner is high-level commands in terms of discrete waypoints. We use A^* and D^* algorithms to achieve motion planning at this level. At the mid-level, the nominal dynamics of the vehicle and local characteristics of the environment are considered in motion planning. This middle layer uses the discrete waypoints from the global-level planner to form continuous reference trajectories that satisfy the dynamical constraints of the vehicle. Sensor information describing the surrounding of the vehicle is sent to the global layer, which uses it to modify the global environmental model, consequently affecting the motion planned at that level. We use optimal control formulation to achieve motion planning at this layer. The bottom layer robustly tracks the reference trajectories provided by the mid-level planning layer. Advanced algorithms already exist to design robust controllers for vehicle inner-loop control. We assume that such a controller is part of the vehicle under consideration and design of such controllers is not considered in this paper.

We do not address uncertainty in this research work, as motion planning without uncertainty is quite a complex problem in itself. However, in real-world applications, there will be uncertainty. This will largely be due to sensor

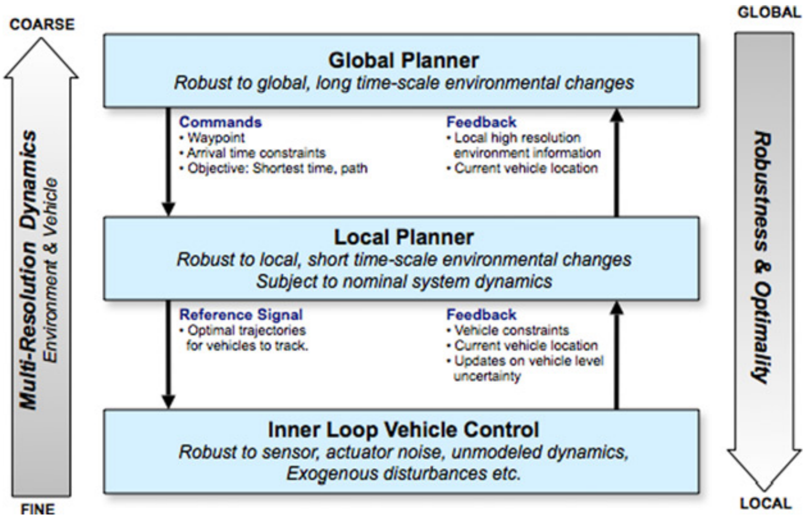

COARSE
Multi-Resolution Dynamics
Environment & Vehicle
FINE
Global Planner
Robust to global, long time-scale environmental changes
Commands
• Waypoint
• Arrival time constraints
• Objective: Shortest time, path
Feedback
• Local high resolution environment information
• Current vehicle location
Local Planner
Robust to local, short time-scale environmental changes
Subject to nominal system dynamics
Reference Signal
• Optimal trajectories for vehicles to track.
Feedback
• Vehicle constraints
• Current vehicle location
• Updates on vehicle level uncertainty
Inner Loop Vehicle Control
Robust to sensor, actuator noise, unmodeled dynamics, Exogenous disturbances etc.
GLOBAL
Robustness & Optimality
LOCAL

Fig. 1 Architecture for multi-layer motion planning.

errors. Uncertainty is expected to have different time and length-scale dynamics as well, and the proposed framework can be used to address robustness at the appropriate planning layers. The global layer would address robustness to long time and length-scale uncertainty such as location of threats and obstacles over a relatively larger geographical area. This information is expected to arrive dominantly from large-scale aerial imagery of the environment. The path planned a this level would avoid these regions in the map. The mid-level planner would connect the waypoints provided by the global planner, and avoid obstacles that may have emerged recently or are smaller in size that are not visible in the coarse resolution map available to the global planner. This information will be available to the mid-level planner as these will be detected by the short-range sensors on the vehicle. The bottom layer will robustly track the continuous time trajectory in the presence of sensor noise that corrupts vehicle state information such as position, velocity, etc. This can be easily done using the well developed robust control theory [16].

III. Hierarchical Motion Planning

A. High-Level Planner

The global-level planner discretizes the environment and defines a configuration space. The dynamics of the vehicle is completely ignored at this layer. The configuration space is created using Minkowski's sum to grow dimensions of the polygonal obstacle by adding possible orientations of the vehicle. Let $\mathcal{G}$ be the environment and $A(\rho) \subset \mathcal{G}$ be a rigid-body robot, where $\rho = (x, y, \theta)$ defines the configuration space with respect to the position and orientation of the robot. The space of obstacles $C_{\text{obs}} \subset \mathcal{G}$ is generated according to the addition of $A(\rho)$. C_{obs} is defined as [17]

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

where $\mathcal{O}$ represents set of obstacles as mentioned earlier. The rest of the space is obstacle free, hence $C_{\text{free}} = \mathcal{G} - C_{\text{obs}}$ obtained by appropriate set operations [18]. The set C_{free} provides the region where the vehicle can maneuver without colliding with the obstacles.

The next step is to generate a roadmap that connects q_{init} and q_{goal} in C_{free}. In this paper we consider motion planning for completely known and partially known environments. The mechanism for generating roadmaps is different for the two scenarios, which is described as follows.

1. Known Environments

For known environments, two methods are commonly used. The first method is visibility graph and decomposition-based methods. The other approach is based on Voronoi diagrams and Delaunay triangles. The roadmap is then generated using a graph search algorithm that connects q_{init} to q_{goal}. In this paper we use the approach based on Delaunay triangulation. Figure 2 illustrates an example environment with six obstacles, modeled as polygons. The vertices of the obstacles are the Voronoi sites $\mathcal{P} = \{P_1, \ldots, P_n\}$, and connecting them would result in m Delaunay triangles $\mathcal{T} = \{T_1, \ldots, T_m\}$. The edges of these triangles however can intrude the obstacles, thus a pruning procedure is applied to capture polygons $P_i = T_i - C_{\text{obs}}$. For each P_i, let us represent the corresponding centroid as P_{c_i}. The set $\{P_{c_i}\}$ represents a graph over which a graph search algorithm, such as A^*, is performed to obtain the shortest path (S_s) in C_{free} and thus avoids all the obstacles in $\mathcal{G}$. The initial and the desired final location of the mobile robot is incorporated in the heuristic function. Once the shortest path in $\mathcal{G}$ has been generated, we define $\mathcal{K}$ to be the set of all polygons that contain the shortest path, i.e.,

$$\mathcal{K} = \{P_i | P_i \cap S_s \neq \emptyset\} \tag{1}$$

The shortest path and the corresponding set $\mathcal{K}$, for the example scenario, is illustrated in Fig. 3.

Figure 3b illustrates that the optimality of the path S_s depends on the coarseness of the graph over which the shortest path is determined. Adding more nodes will improve the optimal solution but will require more computational time. Figure 4 shows the exponential increase in computational time with respect to the number of nodes in the graph.

2. Unknown Environments

In unknown environments, it is assumed that the robot has local information about its immediate neighborhood. Motion planning in this framework requires incremental search algorithm, which makes use of current information to update the motion plan. The motivation is that many robotic vehicles only acquire limited amount of information

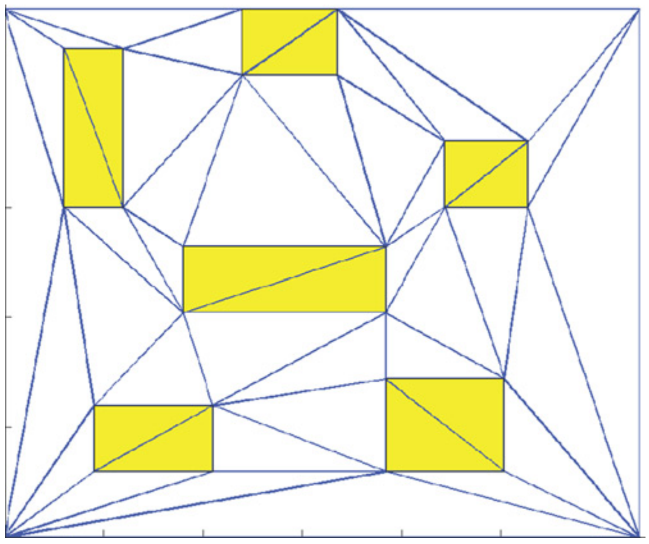
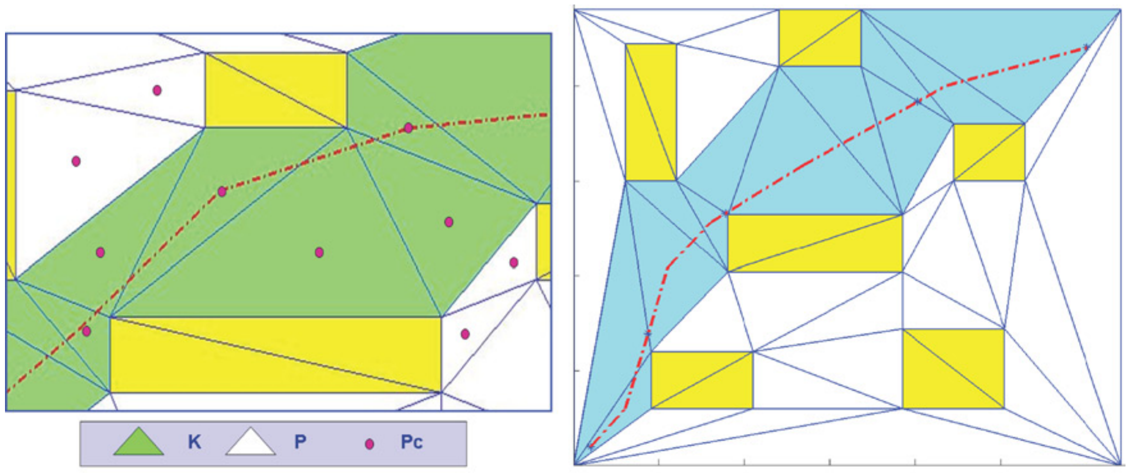


Fig. 2 Triangulation of C_{free}.



(a) Shortest path obtained using A^* algorithm on $\{P_{c_i}\}$. (b) Shortest path connecting q_{init} and q_{goal} in $\mathcal{G}$.

Fig. 3 Global-level motion planning using Delaunay triangulation and graph search algorithm.

on terrain and obstacles and fuse current and previous information for navigation. D^* was originally developed for this purpose. It finds the discrepancy between the given map and the real map and updates the map while traversing.

Another algorithm for generating paths in partially known environments is the lifelong planning A^* algorithm (LPA^*) [19], which is algorithmically very similar to A^*, but has better theoretical guarantees and is simpler to implement. One particular improvement of LPA^* from A^* is the RHS value. It is the one-step look-ahead value of the heuristic function $g(s)$. It is defined as

$$\text{RHS}(s) = \begin{cases} 0 & \text{if } s = s_{\text{goal}} \\ \min\limits_{s' \in \text{Succ}(s)} (g(s') + c(s, s')) & \text{otherwise} \end{cases}$$

where s is a vertex and Succ(s) is the adjacent vertex in the graph $\{P_{c_i}\}$, as defined in Sec. III.A.1. Heuristic functions or measures such as $g(s)$, $h(s)$, and $f(s)$ are preserved in LPA^*. If $g(s) = \text{RHS}(s)$, the vertex s is called locally

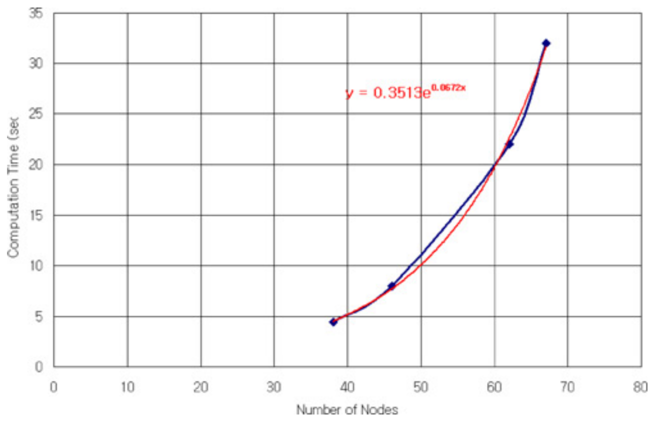


Fig. 4 Computational time of *A algorithm as a function of number of nodes.**

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 $k_1(s) = \min(g(s), \mathrm{RHS}(s)) + h(s, s_{\mathrm{goal}})$ and $k_2(s) = \min(g(s), \mathrm{RHS}(s))$. 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 path. With LPA^*, local changes of the map does not affect the entire graph. Only the local vertices whose costs need to be updated according to the change undergo recalculation and replanning. This makes LPA^* very efficient if storage is not an issue.

In the research work presented here, we used LPA^* to perform motion planning in unknown environments. The `ANSI C` code was provided by Dr. Sven Koenig from University of Southern California. We next demonstrate the application of LPA^* algorithm to the high-level planning problem. We consider two scenarios. In the first scenario, the motion of the robot is restricted to orthogonal directions in a two-dimensional rectangular grid. In the second scenario, motion along the diagonal directions are also allowed. The heuristic 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. 5 and 6.

In the results shown, it is assumed that the sensors have a finite range ahead of the robot position and cannot provide the coverage for the whole area. For this example, we assumed that the sensor horizon is 10 grids in four directions, making the detection area a square. This finite sensor horizon causes multiple replanning to occur.

B. Mid-Level Planner

The global-level planner provides a sequence of discrete points without considering the dynamics of the vehicle. 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. The main ingredients of the mid-level planner is construction of local feasible sets (FSs) from the set $\mathcal{K}$, defined in the previous section. The FS defines local subspaces of the configuration space which is convex and is devoid of obstacles. It is then used to constrain the trajectories of the vehicle, in the OCP formulation. The trajectories of the system are parameterized by B-splines and the OCP is transcribed to a nonlinear programming problem (NLP), which is solved using commercially available NLP solvers. The local, dynamically feasible paths are generated in a receding horizon control (RHC) framework.

1. Feasible Sets

Using OCP formulation to generate dynamically feasible trajectories is a viable option for designing the mid-level planner. However, the computational complexity in solving such problems, with high-state dimension and large number of constraints, is quite high. For environments with several obstacles, it is clear that an OCP formulation will result in large number of constraints and may render this approach computationally infeasible.

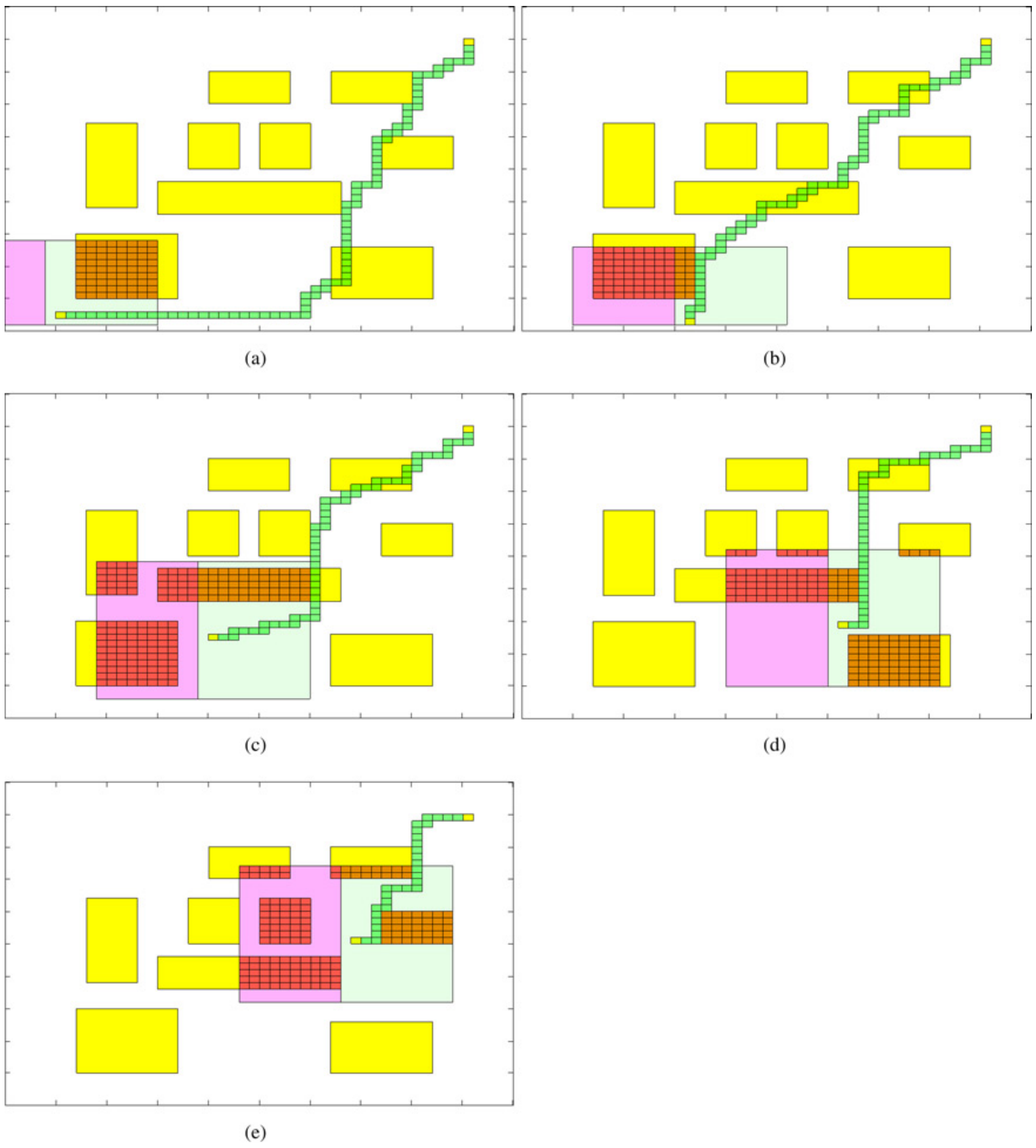

(a) (b) (c) (d) (e)

Fig. 5 Results from LPA* with robot restricted to move in four directions.

We propose the concept of a FS. A FS is a set of convex polygons that is the convex cover of $\mathcal{K}$. Recall that $\mathcal{K}$ is the set of Delaunay triangles intersected by the trajectory generated by the global-level planner (see Eq. 1). The algorithm for recursive construction of a convex cover of $\mathcal{K}$ is summarized in Algorithm A. The algorithm starts with the triangle $\mathcal{K}_p \in \mathcal{K}$, which is the Delaunay triangle containing the present location of the robot. A child $\mathcal{K}_c$ can be found by examining triangles adjacent to $\mathcal{K}_p$ in $\mathcal{K}$. If the union of $\mathcal{K}_p$ and $\mathcal{K}_c$ is convex, then $\mathcal{K}_p$ is redefined to be $\mathcal{K}_p \cup \mathcal{K}_c$, else $\mathcal{K}_p$ is added to FS and $\mathcal{K}_c$ is defined to be $\mathcal{K}_p$. Figure 7 illustrates the incremental construction of FSs.

In a known environment, a single execution of A^* algorithm provides all the necessary polygons to execute Algorithm A and generate FS. Thus, FSs are generated all at once. The recursive construction of FS is shown in

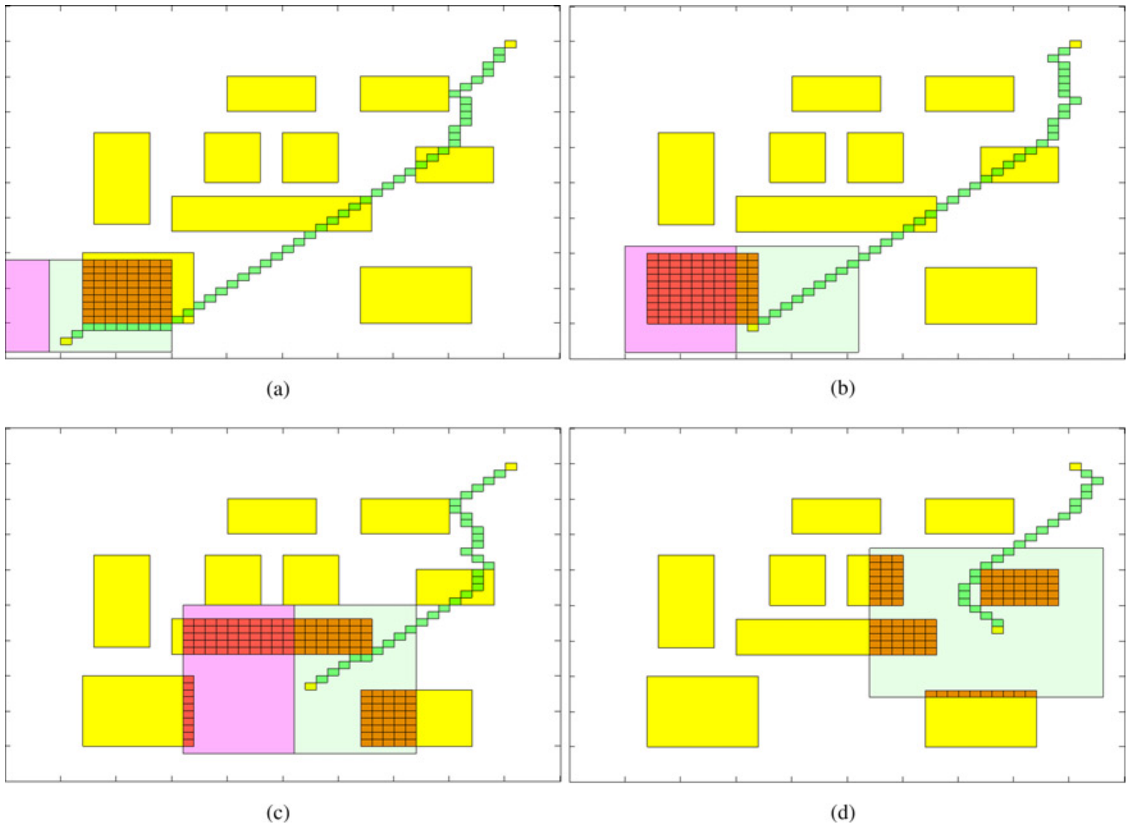


Fig. 6 Results from LPA* with robot restricted to move in eight directions.

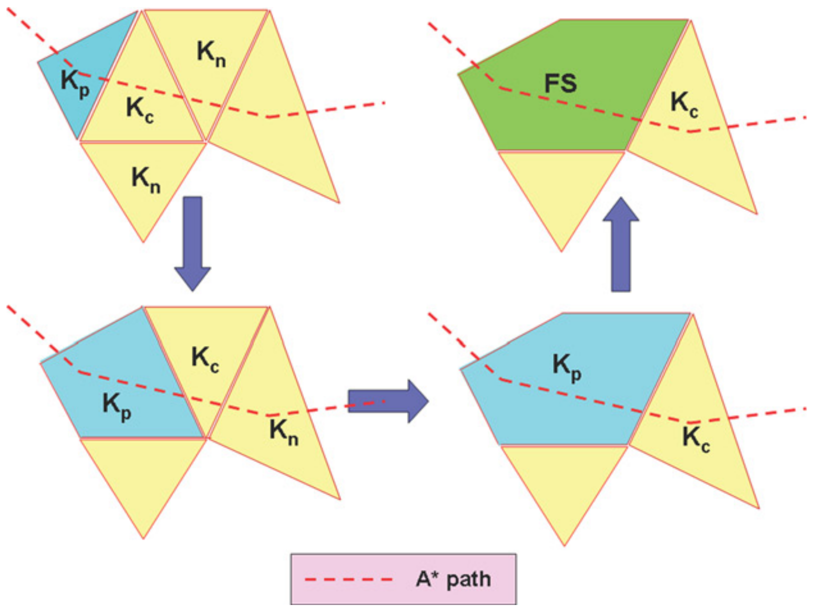


Fig. 7 Incremental construction of FS.

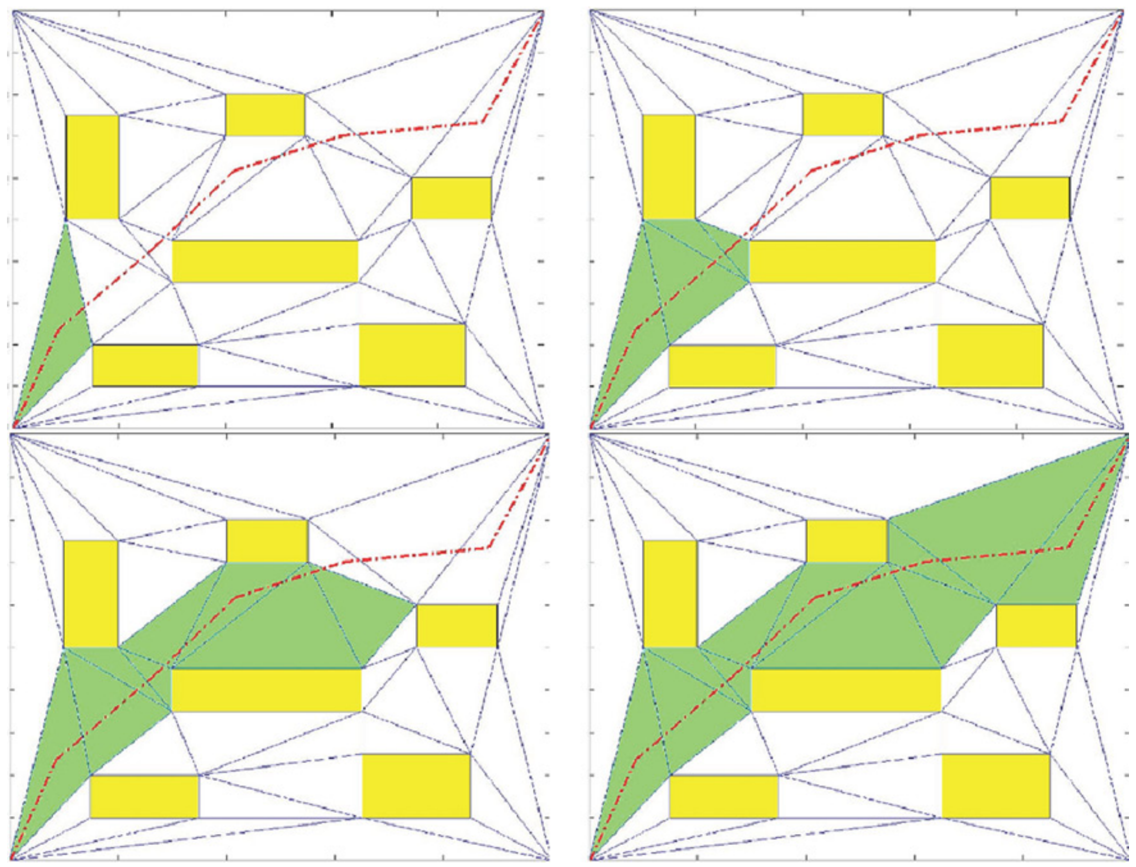
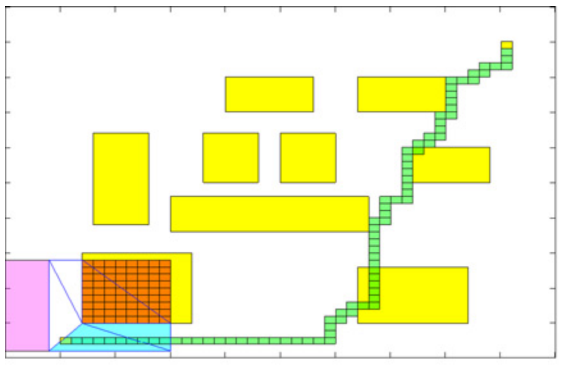


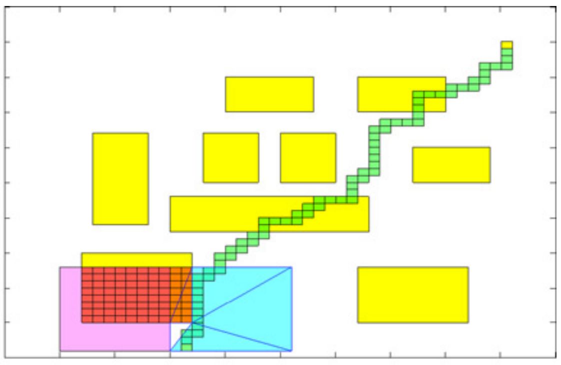
Fig. 8 Recursive construction of FSs in known environments.

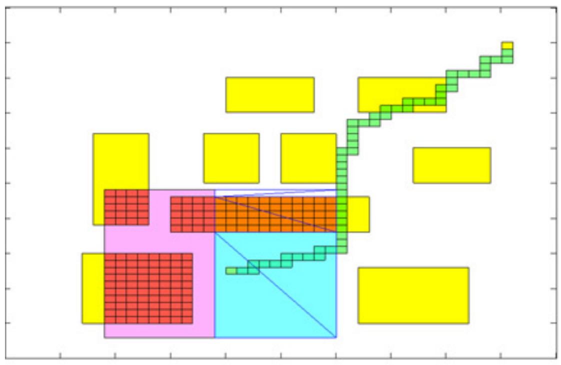
Fig. 8. Motion planning in unknown environment requires sensing and mapping of the surroundings as the vehicle maneuvers, therefore FSs are generated every time new information is gathered by the vehicle. The mechanism of FSs generation is similar to that for known environments, with a few differences. First, the use of A^* algorithm is not required. LPA^* provides the projected shortest path. The region inside the sensor horizon is triangulated and the set $\mathcal{K}$ is determined from the triangulated region and the projected shortest path. Then, Algorithm A takes $\mathcal{K}$ and creates FS. FS is generated repeatedly until the goal is reached. This is illustrated in Fig. 9.

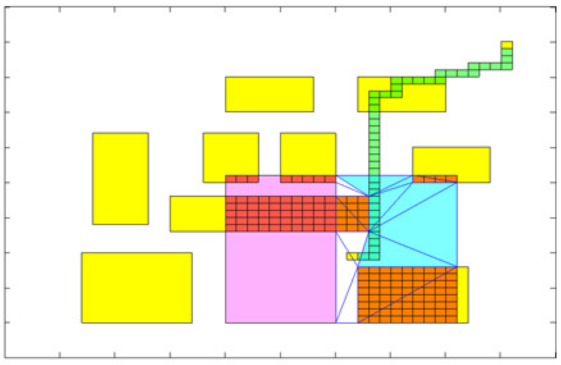
Algorithm A (Feasible Set Generation)

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


(a) Iteration 1


(b) Iteration 2


(c) Iteration 3


(d) Iteration 4

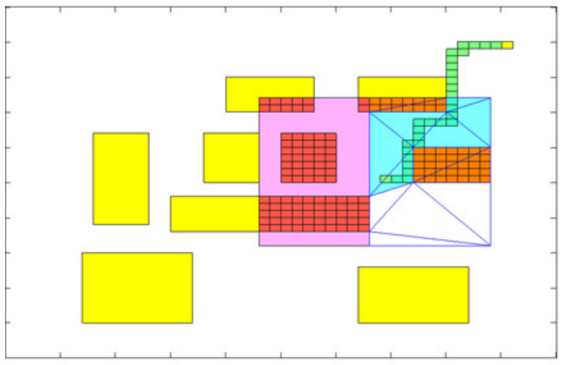

(e) Iteration 3

Fig. 9 Recursive construction of FSs in unknown environments.

2. *B-Splines and Trajectory Parameterization*

Once the waypoints and the corresponding FSs have been constructed, trajectories that satisfy the dynamics of the vehicle need to be generated. We generate these trajectories using OCP formulation. For this purpose we parameterize trajectories using B-splines [20], which is defined as follows

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

where Φ_i are the n control points and $B_{i,r}$ denotes B-spline basis functions with a degree r. The set t represents a nondecreasing 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 $s \leqslant r$, 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

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

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

The choice of the parameters N, r, and s, also define the degree of freedom in the optimization problem, which is given by the expression $N(r - s) + s$. Care should be taken to provide more degrees of freedom than the number of constraints in the problem. There is no systematic way of selecting these parameters. There are few rules of thumb, which are governed by the desired degree of continuity of the trajectory and their derivatives, along with the desired redundancy in the degrees of freedom.

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 that simplifies path constraints, is that a B-spline curve remains inside the convex hull of its control points. We can translate this relationship mathematically as

$$A_c S(t) \leqslant b_c \tag{2}$$

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 eq. (2)

$$A_f \rho \leqslant b_f \tag{3}$$

where subscript f denotes the relation with FS and ρ represents control points (α, β). This linear constraint greatly simplifies the OCP formulation, as obstacle avoidance usually requires a large number of nonlinear path constraints. Figure 10 shows a graphical representation of the concept.

3. *Trajectory Generation Using OCP Formulation*

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

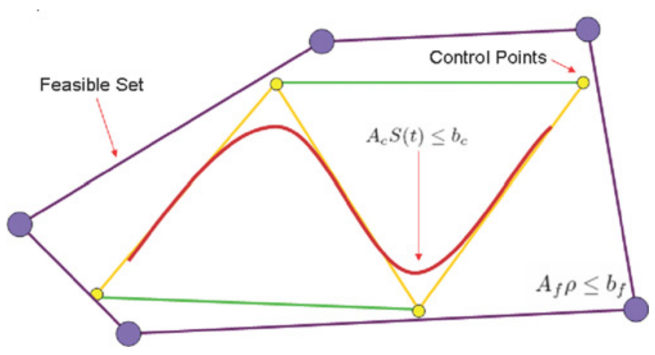


Fig. 10 B-splines lie in the convex hull of control points.

Minimize

$$J = \phi_0(x_0, u_0, t_0) + \int_{t_0}^{t_f} L(x(t), u(t), t)\,\mathrm{d}t + \phi_f(x_f, u_f, t_f) \tag{4}$$

subject to dynamics

$$\dot{x} = f(x(t), u(t)) \tag{5}$$

The initial, final, and trajectory constraints are

$$\begin{aligned} lb_0 &\leqslant \psi_0(x_0, y_0, t_0) \leqslant ub_0 \\ lb_f &\leqslant \psi_f(x_f, y_f, t_f) \leqslant ub_f \\ lb(t) &\leqslant \psi_t(u(t)) \leqslant ub(t) \end{aligned}$$

FSs are enforced as trajectory constraints. Therefore, it is expressed as

$$A_i \rho \leqslant b_i$$

As the trajectory migrates through the sequence of N FSs, 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 $\rho_i = [\alpha_i \ \beta_i]^{\mathrm{T}}$. After OCP is formulated, we parameterize states x and y into B-spline curves. Now OCP has become an optimization problem over coefficients α_i and β_i at every time step

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

where Nc is the total number of collocation points where all constraints are enforced and the cost function is evaluated. The OCP formulation is then transcribed into a NLP. To implement the SNOPT [21] solver, the formulation was altered as follows

$$\min\ F(z)$$

subject to constraints

$$\mathrm{LB} \leqslant \left\{ \begin{matrix} z \\ Az \\ C(z) \end{matrix} \right\} \leqslant \mathrm{UB}$$

The resulting α and β from the NLP generate state and control output. This NLP problem is sparse in z, and SNOPT is the solver of choice for such sparse problems.

4. Receding Horizon Control

In the RHC framework, a finite time OCP (see Eq. (4)) is solved repeatedly. The optimal control obtained from a given solution, is applied to the system over a very short interval. The updated states are then used to initialize the next OCP, thus achieving state feedback. With reference to Fig. 11, the OCP is solved with initial condition at **A**, over a horizon length of T. The optimal state trajectories are shown in dotted lines. The optimal control obtained is applied to the system over $t \in [t_0, t_0 + \Delta T]$. The system is expected to evolve differently from the model used to determine the control trajectory, and is shown in solid. At the end of $t_0 + \Delta t$, the system reaches point **B**. The OCP is resolved with initial condition at **B**. This is performed repeatedly. For a detailed description of RHC framework, the reader is directed to reference [22,23].

Since it is desired to reach a given waypoint, provided by the global-level planner, we formulate the cost function to be minimized as the distance from the waypoint at the end of final time T, i.e., minimize

$$J = (x(T) - x_{\mathrm{wp}})^2 + (y(T) - y_{\mathrm{wp}})^2.$$

To ensure smoothness of the state and control trajectories, we impose appropriate boundary conditions every time the OCP is solved. The waypoints are changed to the next one in sequence, once the cost-to-go has reached below a specified threshold value. In this manner, the RHC-based trajectory generation algorithm is able to connect the sequence of waypoints, provided by the global-level planner, with trajectories are that are dynamically feasible and piecewise optimal. OPTRAGEN [24] was used to obtain optimal trajectories at each iteration of RHC. OPTRAGEN has a built-in transcription method to NLP, so that cost and constraints, types of collocation and trajectory representation is completely transparent to the user.

C. Extension to Three Dimension

The algorithms presented in the previous section can be applied to motion planning problems two-and three-dimensional environments. The complexity in two-dimensional space will be lower than that in three-dimensional space. The algorithms presented are general and do not depend on the dimensionality of the environment. The growth in complexity due to increased dimensionality will occur both at the global-level planner and at the mid-level planner.

In the high-level planner, complexity will increase due to triangulation of a three-dimensional space. The complexity of Delaunay triangulation algorithms is $\mathcal{O}(N^2)$ and under mild assumptions can be reduced to $\mathcal{O}(N\log(N))$ [25],

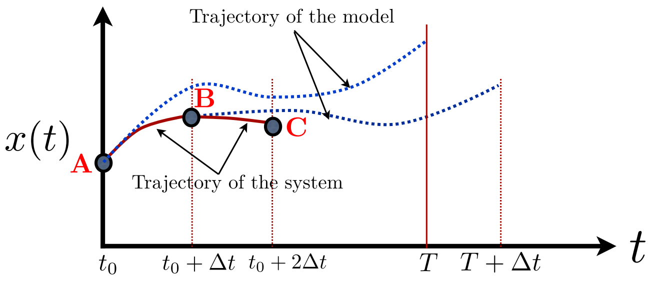


Fig. 11 Mechanism of RHC framework.

where N is the number of vertices in the problem. For completely known environments, N can be large, but the discretization can be performed completely offline and thus growth in complexity is not of great concern. For unknown environments, the robot is only able to sense the environment locally. It is expected that the number of vertices in the triangulation problem in this case is not too large and hence can be computed in real time.

In the mid-level planner, higher-dimensional environment implies increase in the state space of the dynamics of the vehicle. For typical robotic vehicles this will add possibly two more states, corresponding to position and velocity of the vehicle in the third dimension, which is turn will increase the complexity of the OCP. However, in this problem, the main element in the OCP that affects the computational time OCP is the complexity of the constraints that guarantee obstacle avoidance. For three-dimensional space, this will be defined by a convex polytope instead of a convex polygon. Since the constraint set is still convex, the growth in complexity in the mid-level planner is not significant.

IV. Experimental Results

For the purpose of verification and validation of the proposed motion planning algorithm, 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. There are three components that govern the low-level robot control. Figure 12 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 the motion planning algorithm. After all the trajectories and controls are generated, the trajectory generation module sends commands to the robot control software, written in Java, to give appropriate vehicle level control commands. Through wireless connections, the robot receives command inputs, then executes them. The vision system detects the robot and the feedback loop is complete.

A. Hardware Description of Vehicle

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 is the Qwerk embedded computer. It features a 200 MHz ARM9 RISC processor with 32 MB of SDRAM and 8 MB flash memory. It is equipped with four 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

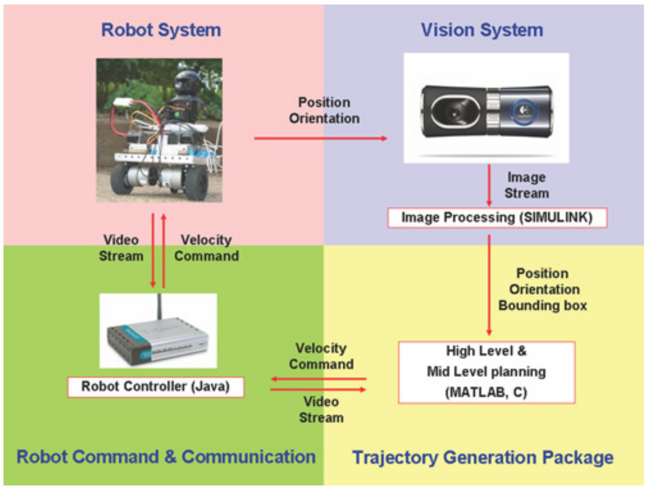


Fig. 12 Experiment setup.

commands and transmitting information. The Logitech QuickCam STX mounted on the robot can provide real-time video stream to the computer.

B. 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 320×240 with the frame rate of 30 fps. Its wide angle lens allows for larger coverage area of the experimental environment compared with conventional webcams. The localization algorithm was built using SIMULINK video processing blocksets and default blocksets.

C. Model for the Vehicle

In this research work we have used the TeRK as the primary vehicle in the motion planning set up. The dynamics of the vehicle is described as follows

$$\dot{x} = \frac{R}{2}(u_l + u_r)\cos\theta \tag{6}$$

$$\dot{y} = \frac{R}{2}(u_l + u_r)\sin\theta \tag{7}$$

$$\dot{\theta} = \frac{R}{L}(u_r - u_l) \tag{8}$$

where $R = 1$ in. is the radius of the wheel, $L = 5.5$ in. is the distance between wheels, and u_l and u_r are the left and right motor speeds, constrained by $0 \leqslant u_l, u_r \leqslant 6$ in./s. This system is differentially flat [26], therefore instead of directly parameterizing x, y, u_l, u_r, we described the system in terms of flat outputs

$$\begin{aligned} z_1 &= x, \quad z_1^{(1)} = \dot{x}, \quad z_1^{(2)} = \ddot{x} \\ z_2 &= y, \quad z_2^{(1)} = \dot{y}, \quad z_2^{(2)} = \ddot{y} \end{aligned} \tag{9}$$

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

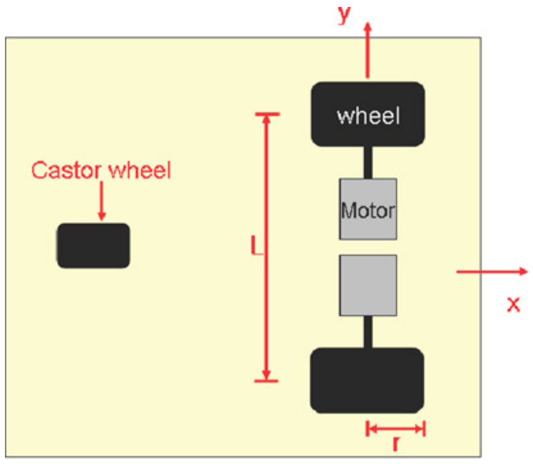
$$\begin{aligned} \theta &= \tan^{-1}\left(\frac{\dot{y}}{\dot{x}}\right) \\ &= f_1\left(z_1, z_2, z_1^{(1)}, z_2^{(1)}\right) && (10) \\ u_l &= \dot{x}(1 + \dot{y}^2/\dot{x}^2)^{1/2} - 1.5(\dot{x}\ddot{y} + \dot{y}\ddot{x})/(\dot{x}^2 + \dot{y}^2) \\ &= f_2(z_1, z_2, z_1^{(1)}, z_2^{(1)}, z_1^{(2)}, z_2^{(2)}) && (11) \\ u_r &= \dot{x}(1 + \dot{y}^2/\dot{x}^2)^{1/2} + 1.5(\dot{x}\ddot{y} + \dot{y}\ddot{x})/(\dot{x}^2 + \dot{y}^2) \\ &= f_3(z_1, z_2, z_1^{(1)}, z_2^{(1)}, z_1^{(2)}, z_2^{(2)}) && (12) \end{aligned}$$

The schematic and picture of TeRK (Fig. 13) shows the setup of the wheels and motors underneath the robotic platform.

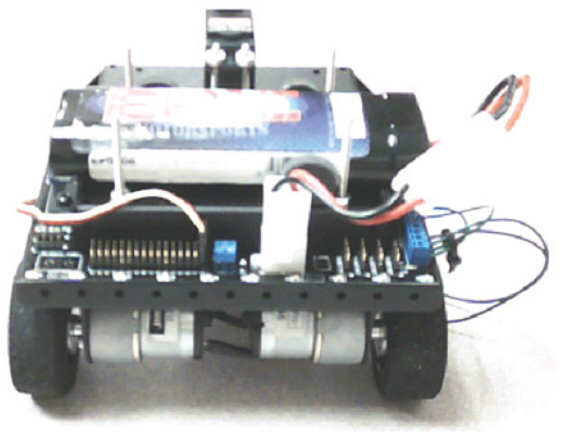
D. Results

1. Known Environment

All obstacles were assumed to be stationary. 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 map. Originally, the map was designed to provide pixel by pixel value for the vision system, therefore its size is 320×240. Considering the real experimental field size of 12 ft $\times$ 8 ft, the speed was scaled, so that the maximum



(a) Schematic for bottom view of TeRK



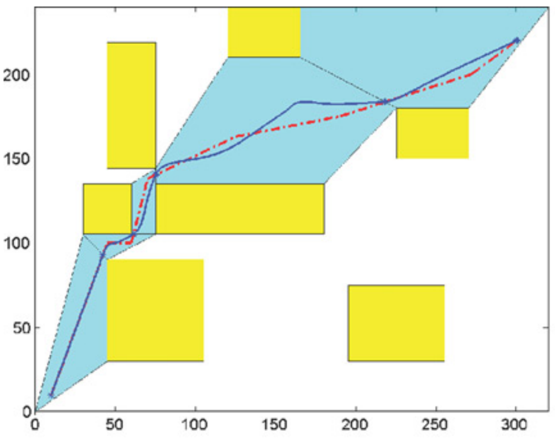
(b) TeRK: Motion planning testbed.

Fig. 13 Bottom view and the picture of the TeRK.

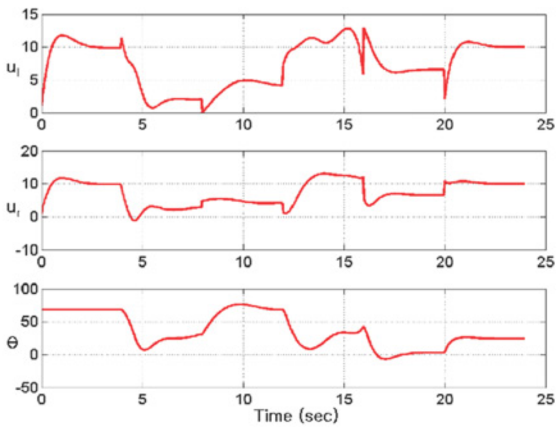
speed of the robot at 6 in./s would be expressed as 14 pixels/s. We can see that wheel speeds u_l and u_r on the TeRK do not exceed the 14 units, thereby satisfying the constraints imposed. In a narrow environment such as in Fig. 14, TeRK was also able to avoid obstacles without violating any control constraints. The control is not globally continuous in its derivative as no constraint was imposed to ensure it.

2. *Unknown Environments*

To demonstrate the planning in partially known environment, the LPA^* algorithm was used, with permissible motion along eight directions. The motion planning was done on a 50×50 map. This is a smaller map than the one for the known environment, consequently the maximum TeRK wheel velocity is scaled down to nearly 3.5 pixels/s. In both the scenarios, sensor horizon was set at 10 grids. In the plots shown, motion planning stopped when the goal (shown as a yellow square) falls within the range of the sensors. Once again, the control trajectories satisfy the constraints. Like before, they do not satisfy derivative continuity everywhere as the OCP did not impose such constraints. Figure 15 show the results obtained from the experiments.

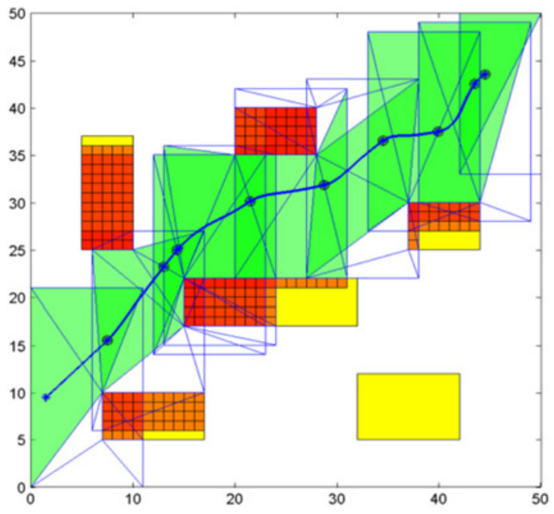


(a) Trajectory in (x, y) plane

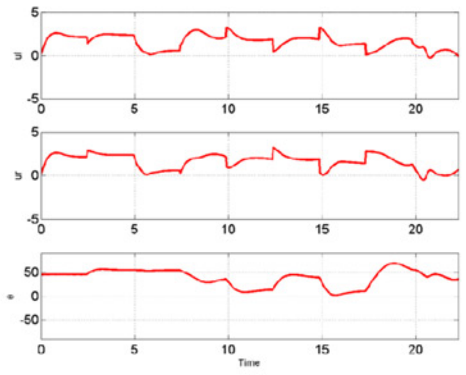


(b) State and control trajectories. u_l, u_r are in pixel / s; θ in degrees.

Fig. 14 Known environment, $T = 4$ s.



(a) Trajectory in (x, y) plane



(b) State and control trajectories. u_l, u_r are in in / s; θ in degrees.

Fig. 15 Unknown environment map 1, $T = 5$ s.

V. Summary and Conclusions

In this paper we presented a hierarchical framework for motion planning in obstacle rich environments. The proposed hierarchy achieves a separation of concern by addressing global-level planning using coarse representation of the environment and mid-level planning using fine representation of the vehicle dynamics and the environment. Motion planning at the global level is achieved by using graph search algorithms such as A^* and D^* which issues waypoints and local convex regions that are obstacle free. The dynamics of the vehicle is ignored at this level of planning. Motion planning at the mid-level is done using optimal control formulation where dynamics, actuator constraints and path constraints can be explicitly imposed. The waypoints and the FSs from the global planner are used to formulate an appropriate OCP and trajectories are generated in the receding horizon framework. The framework is shown to work successfully on a robotic platform.

References

[1] Lozano-Perez, T., and Wesley, M. A., "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, Vol. 22, No. 10, 1979, pp. 560–570.
doi: 10.1145/359156.359164

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

[3] Latombe, J. C., *Robot Motion Planning*, Kluwer, Dordrecht, 1991.

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

[5] Hart, P. E., Nilson, N. J., and Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions of Systems Science and Cybernetics*, Vol. 4, No. 2, 1968, pp. 100–107.
doi: 10.1109/TSSC.1968.300136

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

[7] Frazzoli, E., Dahleh, M. A., and Feron, E., "Real-Time Motion Planning for Agile Autonomous Vehicles," *Proceedings of American Control Conference*, Arlington, VA, Vol. 1, No. 1, 2001, pp. 43–49.

[8] Hsu, D. et al., "Randomized Kinodynamic Planning with Moving Obstacles," *The International Journal of Robotics Research*, Vol. 21, No. 3, 2002, pp. 233–255.
doi: 10.1177/027836402320556421

[9] LaValle, S., and Kuffner, J., "Randomized Kinodynamic Planning," *Proceedings of the 1999 IEEE Conference on Robotics and Automation*, Detroit, MI, Vol. 2, 2000, pp. 473–479.
[10] Milam, M. B., Franz, R., and Murray, R. M., "Real-Time Constrained Trajectory Generation Applied to a Flight Control Experiment," *IFAC*, Barcelona, Spain, 2002.
[11] Singh, B., and Bhattacharya, R., "Near Time Optimal Waypoint Tracking of a 3-DOF Model Helicopter," *AIAA Guidance, Navigation and Control Conference and Exhibit*, Hilton Head, SC, AIAA-2007-6738, 2007.
[12] Betts, J. T., "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 2, 1998, pp. 193–207.
doi: 10.2514/2.4231
[13] Petit, N., Milam, M., and Murray, R., "A New Computational Method for Optimal Control of a Class of Constrained Systems Governed by Partial Differential Equations," *Proceedings of the 15th IFAC World Congress*, Barcelona, Spain, 2002.
[14] Kuwata, Y., Bellingham, J., and How, J., "Stable Receding Horizon Trajectory Control for Complex Environments," *Proceedings of American Control Conference*, Boston, MA, Vol. 1, No. 1, 2004, pp. 902–907.
[15] Flores, M. E., and Milam, M. B., "Trajectory Generation for Differentially Flat Systems via Nurbs Basis Functions with Obstacle Avoidance," *American Control Conference*, Minneapolis, MN, 2006.
[16] Dullerud, G. E., and Paganini, F., *A Course in Robust Control Theory—A Convex Approach*, Texts in Applied Mathematics, edited by Dullerud, G. E., and Paganini, F., Vol. 36, Springer, 2000, reprint, 2004, 444 p.
[17] O'Rourke, J., *Computational Geometry in C*, Cambridge Univ. Press, Cambridge, UK, 1998.
[18] Lindemann, S. R., and LaValle, S. M., "A Multiresolution Approach for Motion Planning Under Differential Constraints," *IEEE International Conference on Robotics and Automation*, Orlando, FL, 2006.
[19] Koenig, S., Likhachev, M., and Furcy, D., "Lifelong Planning A*," *Artificial Intelligence Journal*, Vol. 155, No. 1–2, 2004, pp. 93–146.
doi: 10.1016/j.artint.2003.12.001
[20] De Boor, C., *A Practical Guide to Splines*, Springer-Verlag, Berlin, 1978.
[21] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Journal on Optimization*, Vol. 12, No. 4, 2002, pp. 979–1006.
doi: 10.1137/S1052623499350013
[22] Kwon, W. H., Han, S., and Han, S. H., *Receding Horizon Control: Model Predictive Control for State Models*, Springer-Verlag, Berlin, 2005.
[23] Allgwer, F., and Zheng, A., *Nonlinear Model Predictive Control*, Birkhäuser, Boston, MA, 2000.
[24] Bhattacharya, R., "OPTRAGEN: A Matlab Toolbox for Optimal Trajectory Generation," *IEEE Conference on Decision and Control*, San Diego, CA, 2006.
[25] Attali, D., Boissonnat, J.-D., and Lieutier, A., "Complexity of the Delaunay Triangulation of Points on Surfaces the Smooth Case," *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, San Diego, CA, 2003, pp. 201–210.
[26] Fliess, M., Lévine, J., Martin, P., and Rouchon, P., "Flatness and Defect of Nonlinear Systems: Introductory Theory and Examples," *International Journal of Control*, Vol. 61, No. 6, 1995, pp. 1327–1361.

Ella Atkins
Associate Editor