Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling

# ITOMP: Incremental Trajectory Optimization for Real-Time Replanning in Dynamic Environments

**Chonhyon Park** and **Jia Pan** and **Dinesh Manocha**
University of North Carolina at Chapel Hill
http://gamma.cs.unc.edu/ITOMP/

**Abstract**

We present a novel optimization-based algorithm for motion planning in dynamic environments. Our approach uses a stochastic trajectory optimization framework to avoid collisions and satisfy smoothness and dynamics constraints. Our algorithm does not require a priori knowledge about global motion or trajectories of dynamic obstacles. Rather, we compute a conservative local bound on the position or trajectory of each obstacle over a short time and use the bound to compute a collision-free trajectory for the robot in an incremental manner. Moreover, we interleave planning and execution of the robot in an adaptive manner to balance between the planning horizon and responsiveness to obstacle. We highlight the performance of our planner in a simulated dynamic environment with the 7-DOF PR2 robot arm and dynamic obstacles.

## 1 Introduction

Motion planning is an important problem in many robotics applications, including autonomous navigation and task planning. Most of the earlier work on practical motion planning algorithms is based on randomized algorithm (Kavraki et al. 1996; Kuffner and LaValle 2000; LaValle 2006) and is mostly limited to static environments. However, robots must work reliably in dynamic environments with humans and other moving objects, e.g., when performing household jobs like cleaning a room. Therefore, the robot needs to acquire the ability to safely navigate in the environment and perform tasks in the presence of moving obstacles. In order to achieve this goal, many planning algorithms for dynamic environments have been proposed (van den Berg and Overmars 2005; Phillips and Likhachev 2011b; Hauser 2012). However, most of these methods assume that the future trajectories of dynamic obstacles are known a priori during the planning computation. This assumption may not hold in many real world applications. As a matter of fact, most moving objects' motions are not precisely predictable or can only be approximated over a small or local time interval. Such uncertainty about moving objects makes it hard to plan a safe trajectory for the robot over a long horizon.

Another challenge in terms of planning in dynamic environments is that the planning algorithm must be responsive to unpredictable situations, which requires real-time planning capability in terms of computing or updating the trajectory. There is some recent work on accelerating high-DOF planing algorithms, such as GPU parallelism (Pan, Lauterbach, and Manocha 2010) or distributed systems (Devaurs, Simeon, and Cortes 2011). A real-time planner can improve the responsiveness, but it may not provide an adequate solution for all situations. The reason is that there are difficult scenarios, e.g., narrow passages (LaValle 2006), which are hard for any planner in terms of real-time computation. In these cases, planning before task execution can lead to delays in the robot's movement and decrease its safety. One possible solution for handling such scenarios is by interleaving planning with execution; the overall algorithm lands up computing partial or sub-optimal plans to avoid delays in its handling of moving obstacles (Petti and Fraichard 2005; Bekris and Kavraki 2007; Hauser 2012).

In order to overcome these challenges, we present an efficient replanning framework based on optimization-based approaches. Our work is based on recent developments in optimization-based planning that can also handle dynamic constraints efficiently (Kalakrishnan et al. 2011; Ratliff et al. 2009; Dragan, Ratliff, and Srinivasa 2011). In order to handle dynamic obstacles and perform realtime planning, our approach uses an incremental approach (ITOMP). First, we estimate the trajectory of the moving obstacles over a short time horizon using simple estimation techniques. Next, we compute a conservative bound on the position of the moving obstacles based on the predicted motion. We then calculate a trajectory connecting robot's initial and goal configurations by solving an optimization problem that avoids collisions with the obstacles and satisfies smoothness and torque constraints. In order to make the robot respond quickly to the dynamic environments, we interleave planning with task execution: that is, instead of solving the optimization problem completely, we assign a time budget for planning and interrupt the optimization solver when the time runs out. The computed trajectory may be sub-optimal, which means that 1) its objective cost may not be minimized; 2) the collision-free constraints or other additional constraints may not be completely satisfied. The robot then executes over the short time interval based on this sub-optimal path computation. We repeat these steps until the robot reaches the goal position. During each iterative step,

we update the conservative bound on the object's position and also account for any new objects that may have entered the robot's workspace. The updated environment information is incorporated into the optimization formulation, which uses the sub-optimal result from the last step as the initial solution and tries to improve it incrementally within the given timing budget. We demonstrate the performance of our replanning algorithm in the ROS simulation environment where the PR2 robot tries to perform manipulation task with its 7-DOF robot arm.

The rest of the paper is organized as follows. We survey related work on planning for dynamic environments and replanning in Section 2. Section 3 introduces the notation used in the paper and gives an overview of our approach. We present our optimization-based replanning algorithm (ITOMP) in Section 4. We highlight its performance on simulated dynamic environments in Section 5.

## 2 Related Work

In this section, we give a brief overview of prior work on motion planning in dynamic environments, realtime replanning and optimization-based planning.

### Planning in Dynamic Environments

Most of the approaches for motion planning in dynamic environments assume that the trajectories of moving objects are known a priori. Some of them model dynamic obstacles as static obstacles with a short horizon and set a high cost around the obstacles. (Likhachev and Ferguson 2009). Another common approach is to use velocity obstacles, which are used to compute appropriate velocities to avoid collisions with dynamic obstacles (Fiorini and Shiller 1998; Wilkie, van den Berg, and Manocha 2009). However, these methods cannot give any guarantees on the optimality of the resulting trajectory.

Some of the planning methods handle the continuous state space directly, e.g., RRT variants have been proposed for planning in dynamic environments (Petti and Fraichard 2005). For discrete state spaces, efficient planning algorithms for dynamic environment include variants of A* algorithm, which are based on classic heuristic searches (Phillips and Likhachev 2011b; 2011a) and roadmap-based algorithms (van den Berg and Overmars 2005).

Most planning algorithms for dynamic environments (van den Berg and Overmars 2005; Phillips and Likhachev 2011b) assume that the inertial constraints, such as acceleration and torque limit, are negligible for the robot. Such an assumption implies that the robot can stop and accelerate instantaneously, which may not be the case for a physical robot.

### Real-time Replanning

Since path planning can be computationally expensive, planning before execution can lead to long delays during a robot's movement. To handle such scenarios, real-time replanning interleaves planning with execution so that the robot may decide to compute only partial or sub-optimal plans in order to avoid delays in the movement. Real-time replanning methods differ in many aspects; one key difference is the underlying planner used. Sample-based motion planning algorithms such as RRT have been applied to real-time replanning for dynamic continuous systems (Hsu et al. 2002; Hauser 2012; Petti and Fraichard 2005). These methods can handle high-dimensional configuration spaces but usually cannot generate optimal solutions. A* variants such as D* (Koenig, Tovey, and Smirnov 2003) and anytime A* (Likhachev et al. 2005) can efficiently perform replanning on discrete state spaces and provide optimal guarantees, but are mostly limited to low dimensional spaces. Most replanning algorithms that interleave planning and execution use fixed time steps (Petti and Fraichard 2005), although some recent work (Hauser 2012) computes the interleaving timing step in an adaptive manner so as to maintain a balance between the safety, responsiveness, and completeness of the overall system.

### Optimization-based Planning Algorithms

The most widely-used method of path optimization is the so-called 'shortcut' heuristic, which picks pairs of configurations along a collision-free path and invokes a local planner to attempt to replace the intervening sub-path with a shorter one (Chen and Hwang 1998; Pan, Zhang, and Manocha 2011). Another approach is based on Voronoi diagrams to compute collision-free paths (Garber and Lin 2004). Other approaches are based on elastic bands or elastic strips, which use a combination of mass-spring systems and gradient-based methods to compute minimum-energy paths (Brock and Khatib 2002; Quinlan and Khatib 1993). All these methods require a collision-free path as an initial value to the optimization algorithm. Some recent approaches, such as (Ratliff et al. 2009; Kalakrishnan et al. 2011; Dragan, Ratliff, and Srinivasa 2011) directly encode the collision-free constraints and use an optimization-based solver to transform a naive initial guess into a trajectory suitable for robot execution.

## 3 Overview

In this section, we introduce the notation used in the rest of the paper and give an overview of our approach.

We use the symbol $\mathcal{C}$ to represent the configuration space of a robot, including several $\mathcal{C}$-obstacles and the free space $\mathcal{C}_{free}$. Let the dimension of $\mathcal{C}$ be $D$. Each element in the configuration space, i.e., a configuration, is represented as a dim-$D$ vector $\mathbf{q}$.

For a single planning step, suppose there are $N_s$ static obstacles and $N_d$ dynamic obstacles in the environment. The number of dynamic obstacles is changed between the steps as the sensor introduces new obstacles and removes out of range obstacles and the information is kept for a planning interval. We assume that these obstacles are all rigid bodies. For static obstacles, we denote them as $\mathcal{O}_j^s$, $j = 1, ..., N_s$. For dynamic obstacles, as their positions vary with time, we denote them as $\mathcal{O}_j^d(t)$, $j = 1, ..., N_d$. $\mathcal{O}_j^s$ and $\mathcal{O}_j^d(t)$ correspond to the objects in the workspace, and we denote their corresponding $\mathcal{C}$-obstacles in the configuration space as $\mathcal{CO}_j^s$ and $\mathcal{CO}_j^d(t)$, respectively.

In the ideal case, we assume that we have complete knowledge about the motion and trajectory of dynamic obstacles, i.e., we know the functions $\mathcal{O}_j^d(t)$ and $\mathcal{CO}_j^d(t)$ exactly. However, in real-world applications, we may only have local estimates of the future movement of the dynamic obstacles. Moreover, the recent position and velocity of obstacles computed from the sensors may not be accurate due to the sensing error. In order to guarantee safety of the planning trajectory, we compute a conservative local bound on the trajectories of dynamic obstacles during planning. Given the time instance $t_{cur}$, the conservative bound for the moving object $\mathcal{O}_j^d$ at time $t > t_{cur}$ bounds the shape corresponding to $\mathcal{O}_j^d(t)$, and is computed as:

$$\overline{O}_j^d(t) = c(1 + e_s \cdot t)O_j^d(t) \tag{1}$$

where $e_s$ is the maximum allowed sensing error. As the sensing error increases the conservative bound becomes larger. When an obstacle has a constant velocity, it is guaranteed that the conservative bound includes the obstacle during the time period corresponding to $t > t_{cur}$ with $c = 1$. However, if an obstacle changes its velocity, we have to use a larger value of c in our conservative bound, and it would be valid for a shorter time interval. We can define the conservative bound for a moving object $\mathcal{O}_j^d$ during a given time interval $I = [t_0, t_1]$ as follows:

$$\overline{\mathcal{O}}_j^d(I) = \bigcup_{t \in I} \overline{\mathcal{O}}_j^d(t), \forall t \in I, t > t_{cur}. \tag{2}$$

Similarly, we can define conservative bounds in the configuration space, which are denoted as $\overline{\mathcal{CO}}_j^d(t)$ and $\overline{\mathcal{CO}}_j^d(I)$, respectively.

We treat motion planning in dynamic environments as an optimization problem in the configuration space, i.e., we search for a smooth trajectory that minimizes the cost corresponding to collisions with moving objects and some additional constraints, such as joint limit or acceleration limit. Specifically, we consider trajectories corresponding to a fixed time duration $T$, discretized into $N$ waypoints equally spaced in time. In other words, the discretized trajectory is composed of $N$ configurations $\mathbf{q}_1, ..., \mathbf{q}_N$, where $\mathbf{q}_i$ is a trajectory waypoint at time $\frac{i-1}{N-1}T$. We can also represent the trajectory as a vector $\mathbf{Q} \in \mathcal{R}^{D \cdot N}$:

$$\mathbf{Q} = [\mathbf{q}_1^T, \mathbf{q}_2^T, ..., \mathbf{q}_N^T]^T. \tag{3}$$

We assume that the start and goal configurations of the trajectory, i.e., $\mathbf{q}_s$ and $\mathbf{q}_g$, are given, and are fixed during optimization. Figure 1 illustrates the symbols used by our optimization-based planner.

Similarly to previous work (Ratliff et al. 2009; Kalakrishnan et al. 2011), our optimization problem is formalized as:

$$\min_{\mathbf{q}_1, ..., \mathbf{q}_N} \sum_{i=1}^{N} c(\mathbf{q}_i) + \frac{1}{2}\|\mathbf{AQ}\|^2, \tag{4}$$

where $c(\cdot)$ is an arbitrary state-dependent cost function, which can include obstacle costs for static and dynamic objects, and additional constraints such as joint limit and torque limit. That is, the cost function can be divided into three parts:

$$c(\mathbf{q}) = c_s(\mathbf{q}) + c_d(\mathbf{q}) + c_o(\mathbf{q}), \tag{5}$$

where $c_s(\cdot)$ is the obstacle cost for static objects, $c_d(\cdot)$ is the obstacle cost for moving objects and $c_o(\cdot)$ is the cost for additional constraints. As $c_d(\cdot)$ changes along with time due to movement of dynamic obstacles, we sometimes denote it as $c_d^t(\cdot)$ to show the dependency on time explicitly. $\mathbf{A}$ is a matrix that is used to represent the smoothness costs. We choose $\mathbf{A}$ such that $\|\mathbf{AQ}\|^2$ represents the sum of squared accelerations along the trajectory. Specifically, $\mathbf{A}$ is of the form

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 1 & -2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{pmatrix} \otimes \mathbf{I}_{D \times D}, \tag{6}$$

where $\otimes$ denotes the Kronecker tensor product and $\mathbf{I}_{D \times D}$ is a square matrix of size $D$. It follows that $\ddot{\mathbf{Q}} = \mathbf{AQ}$, where $\ddot{\mathbf{Q}}$ represents the second order derivative of the trajectory $\mathbf{Q}$.

The solution to the optimization problem in Equation (4) corresponds to the optimal trajectory for the robot:

$$\mathbf{Q}^* = \{\mathbf{q}_1^* = \mathbf{q}_s, \mathbf{q}_2^*, ..., \mathbf{q}_{N-1}^*, \mathbf{q}_N^* = \mathbf{q}_g\}. \tag{7}$$

However, notice that $\mathbf{Q}^*$ is guaranteed to be collision-free with dynamic obstacles only during a short time horizon. Because we only have a rough estimation based on the extrapolation of the motion of the moving objects, rather than an exact model of the moving objects' motion, the cost function $c_d^t(\cdot)$ is only valid within a short time interval. In order to associate a period of validity with the result of our optimization algorithm, we use $\mathbf{Q}_I^*$ to represent the planning result that is valid during the interval $I = [t_0, t_1] \subseteq [0, T]$.

In order to improve robot's responsiveness and safety, we interleave planning and execution threads, in which the robot executes a partial or suboptimal trajectory (based on a high-rate feedback controller) that is intermittently updated by the replanning thread (at a lower rate) without interrupting the execution. We assign a time budget $\Delta_k$ to the $k$-th step of replanning, which is also the maximum allowed time for execution of the planning result from last step. We use a constant timing budget $\Delta_t = \Delta$, but our approach can be easily extended to use a dynamic timing budget that is adaptive to replanning performance (Hauser 2012). The interleaving strategy is subject to the constraint that the current trajectory being executed cannot be modified. Therefore, if the replanning result is sent to the robot for execution at time $t$, it is allowed to run for time $\Delta$, and no portion of the computed trajectory before $t + \Delta$ may be modified. In other words, the planner should start planning from $t + \Delta$. Due to limited time budget, the planner may not be able to compute an optimal solution of the optimization function shown in Equation (4) and the resulting trajectory may be, and usually is, sub-optimal. Its cost may be greater than or equal to

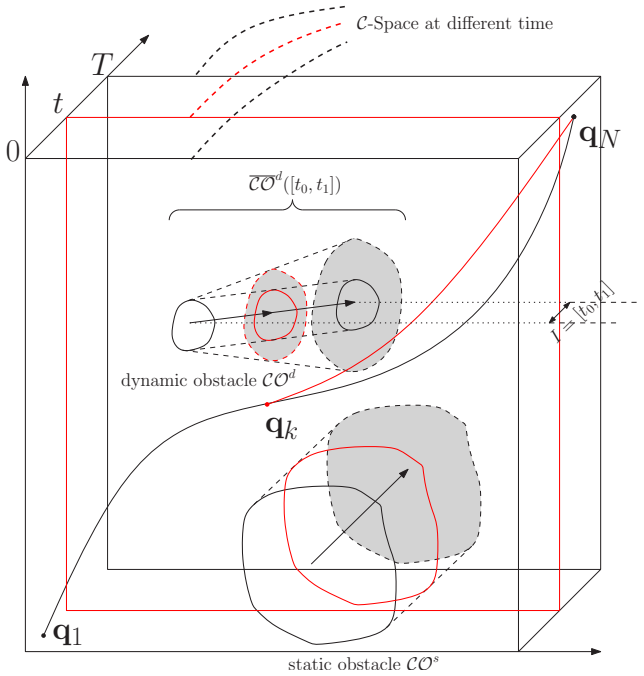



Figure 1: Optimization-based motion planning for dynamic environments. We show how the configuration space changes over time: each plane slice represents the configuration space at time $t$. In the environment, there are two $\mathcal{C}$-obstacles: the static obstacle $\mathcal{CO}^s$ and the dynamic obstacle $\mathcal{CO}^d$. We need to plan a trajectory to avoid these obstacles. The trajectory starts at time 0, stops at time $T$, and is represented by a set of way points $\mathbf{q}_1$, ..., $\mathbf{q}_k$, ..., $\mathbf{q}_N$. Supposing that the trajectory is to be executed by the robot during time interval $I = [t_0, t_1]$, we only need to consider the conservative bound $\overline{\mathcal{CO}}^d([t_0, t_1])$ for the dynamic obstacle during the time interval. The $\mathcal{C}$-obstacles shown in the red color correspond to the obstacles at time $t \in I$.

the cost of the optimal trajectory $\mathbf{Q}^*$. i.e., $f(\mathbf{Q}_I^*) \leq f(\overline{\mathbf{Q}}_I^*)$, where we denote the resulting trajectory from the planner as $\overline{\mathbf{Q}}_I^*$, and $f(\mathbf{Q}) = \sum_{i=1}^{N} c(\mathbf{q}_i) + \frac{1}{2}\|\mathbf{AQ}\|^2$.

## 4 ITOMP Algorithm

In this section, we present our ITOMP algorithm for planning in dynamic environments, i.e., how to solve the optimization problem corresponding to Equation (4). We first introduce the cost metric for static obstacles and dynamic obstacles. Next, we present our incremental optimization algorithm.

### Obstacle Costs

Similarly to prior work (Kalakrishnan et al. 2011; Ratliff et al. 2009), we model the cost of static obstacles using signed Euclidean Distance Transform (EDT). We start with a boolean voxel representation of the static environment, obtained either from a laser scanner or from a triangle mesh model. Next, the signed EDT $d(\mathbf{x})$ for a 3D point $\mathbf{x}$ is computed throughout the voxel map. This provides information about the distance from $\mathbf{x}$ to the boundary of the closest static obstacle, which is negative, zero or positive when $\mathbf{x}$ is inside, on the boundary or outside the obstacles, respectively. One advantage of EDT is that it can encode the discretized information about penetration depth, contact and proximity in a uniform manner and can make the optimization algorithm more robust. After the signed EDT is computed, the planning algorithm can efficiently check for collisions by table lookup in the voxel map. In order to compute the obstacle cost, we approximate the robot shape $\mathcal{B}$ by a set of overlapping spheres $b \in \mathcal{B}$. The static obstacle cost is as follows:

$$c_s(\mathbf{q}_i) = \sum_{b \in \mathcal{B}} \max(\epsilon + r_b - d(\mathbf{x}_b), 0)\|\dot{\mathbf{x}}_b\|, \qquad (8)$$

where $r_b$ is the radius of one sphere $b$, $\mathbf{x}_b$ is the 3D point of sphere $b$ computed from the kinematic model of the robot in configuration $\mathbf{q}_i$, and $\epsilon$ is a small safety margin between robot and the obstacles. The speed of sphere $b$, $\|\dot{\mathbf{x}}_b\|$, is multiplied to penalize the robot when it tries to traverse a high-cost region quickly. The static obstacle cost is zero when all the sphere are at least $\epsilon$ distance away from the closest obstacle.

EDT computation is efficient for static obstacles but cannot be applied to dynamic obstacles, though a GPU-based parallel EDT computation algorithm could be used (Sud, Otaduy, and Manocha 2004). The reason is that the movement of dynamic obstacles implies that EDT needs to be recomputed during each time step and it is hard to perform such computation in real-time on current CPUs. Instead, we perform geometric collision detection between the robot and moving obstacles and use the collision result to formalize the dynamic obstacle cost. Given a configuration $\mathbf{q}_i$ on the trajectory and the geometric representation of moving obstacles $\mathcal{O}_j^s$ at the corresponding time (i.e., $\frac{i-1}{N-1}T$), the obstacle cost corresponding to configuration $\mathbf{q}_i$ is given as:

$$c_d(\mathbf{q}_i) = \sum_j \texttt{is\_collide}(\overline{\mathcal{O}}_j^s(\frac{i-1}{N-1}T), \mathcal{B}), \qquad (9)$$

where $\texttt{is\_collide}(\cdot,\cdot)$ returns one when there is a collision and zero otherwise. The `is_collide` function can be performed efficiently using object-space collision detection algorithms, such as (Gottschalk, Lin, and Manocha 1996). This obstacle cost function is only used during a short or local time interval, i.e. from replanning's start time $t$ to its end time $t + \Delta$, since the predicted positions of dynamic obstacles can have high uncertainty during a long time horizon.

### Dynamic Environment and Replanning

ITOMP makes no assumption about the global motion or trajectory of each moving obstacle. Instead, we predict the future position and the velocity of moving obstacles based on their recent positions, which are generated from noisy sensors. This prediction and maximum error bound are used to compute a conservative bound on the moving obstacles during the local time interval. Therefore, the planning result is guaranteed to be safe only during a short time period. In order to offer quick response during unpredictable cases (e.g., the trajectory prediction about some of the objects is not correct or new moving obstacles enter the robot's workspace), the robot must sense the environment frequently and the planner needs to be interrupted to update the description of the environment.

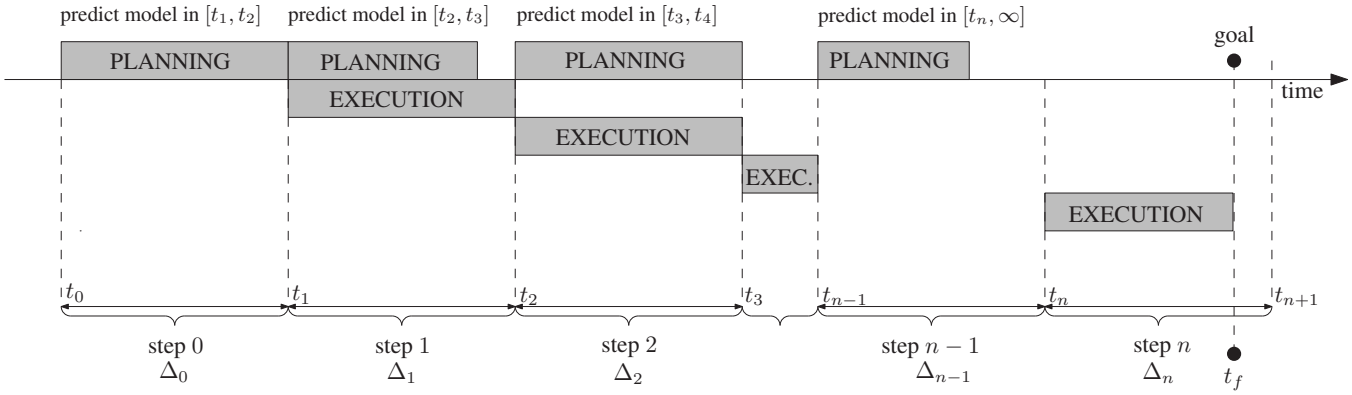



Figure 3: Interleaving of planning and execution. The planner starts at time $t_0$. During the first planning time budget $[t_0, t_1]$, it plans a safe trajectory for the first execution interval $[t_1, t_2]$, which is also the next planning interval. In order to compute the safe trajectory, the planner needs to compute a conservative bound for each moving obstacle during $[t_1, t_2]$. The planner is interrupted at time $t_1$ and the ITOMP scheduling module notifies the controller to start execution. Meanwhile, the planner starts the planning computation for the next interval $[t_2, t_3]$, after updating the bounds on the trajectory of the moving obstacles. Such interleaving of planning and execution is repeated until the robot reaches the goal position. In this example, $n$ interleaving steps are used, and the time budget allocated to each step is $\Delta_i$, which can be fixed or changed adaptively. Notice that if the robot is currently is an open space, the planner may compute an optimal solution before the time budget runs out (e.g., during $[t_2, t_3]$).

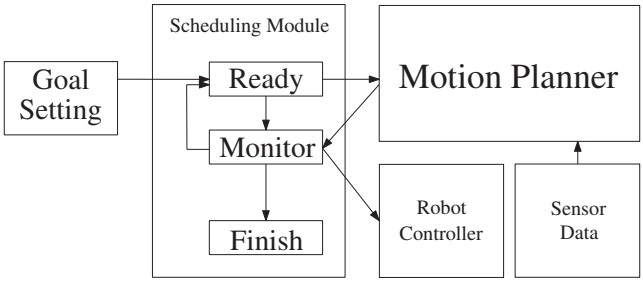



Figure 2: The overall pipeline of ITOMP: the scheduling module runs the main algorithm. It gets input from the user and interleaves the planning and execution threads. The Motion Planner module computes the trajectory for the robot and the Robot Controller module is used to execute the trajectory. The planner also receives updated environment information frequently from sensors.

In order to handle uncertainty from moving obstacles and provide high responsiveness, ITOMP interleaves planning and execution of the robot. As illustrated in Figure 2, ITOMP consists of several parts: the scheduling module, the motion planner, the robot controller and the data-collecting sensor. The scheduling module gets the goal information as input and controls the other modules. When a new goal position is set, the scheduling module sends a new trajectory computation request to the motion planner. When the motion planner computes a new trajectory that is safe within a short horizon, the scheduling module notifies the robot controller to execute the trajectory. Meanwhile, it also sends a new request to the motion planner to compute a safe trajectory for the next execution interval. The planner also needs to incorporate the updated environment description from the sensor data. Since the motion planner runs in a separate thread, the scheduling module does not need to wait for the planner to terminate. Instead, it checks whether the robot reaches the goal, updates the dynamic environment description, and checks whether the planner has computed a new trajectory.

The details about the interleaved planning and execution method are shown in Figure 3. The $i$-th time step of short-horizon planning has a time budget $\Delta_i = t_{i+1} - t_i$, which is also the time budget for the current step of execution. During the $i$-th time step, the planner tries to generate a trajectory by solving the optimization problem in Equation 4. The trajectory should be valid during the next step of execution, i.e., during the time interval $[t_{i+1}, t_{i+2}]$.

Due to the limited time budget, the planner may only be able to compute a sub-optimal solution before it is interrupted. The sub-optimal solution may not be collision-free or may violate some other constraints during the next execution interval $[t_{i+1}, t_{i+2}]$. To handle such cases, we use two techniques. First, we assign higher weights to the obstacle costs related to the trajectory waypoints during the interval $[t_{i+1}, t_{i+2}]$, which biases the optimization solver to reduce the cost during the execution interval. If the optimization result is not valid during the execution interval, ITOMP's scheduling module chooses not to execute during the following execution interval. This approach keeps the planner from violating hard constraints(e.g. torque, end effector orientation, etc.) and allocates more time to the planner to improve the result. If the optimization result is valid but not optimal, i.e., the cost is not minimized during time interval $[t_{i+2}, \infty]$, the planner can also improve it incrementally during following time intervals. The time budget for each step of short-horizon planning can be changed adaptively according to the quality of the resulting trajectory, which tries to balance the robot's responsiveness and safety (Hauser 2012).

Notice that usually the optimization can converge to local optima quickly because during the $i$-th step planning we use the result of $(i-1)$-th step as the initial value. On the other hand, the optimization algorithm tends to compute a sub-optimal solution when the robot is near a region with multiple minima in the configuration space or a narrow passage.

## 5 Results

In this section, we highlight the performance of our planning algorithm in dynamic environments. We have implemented our algorithm in a simulator that uses the geometric and kinematic model of Willow Garage's PR2 robot in the ROS environment. All the experiments are performed on a PC equipped with an Intel i7-2600 8-core CPU 3.4GHz with 8GB of memory. Our experiments are based on the accuracy of the PR2 robot's LIDAR sensor (i.e. 30mm), and the planning routines obtain information about dynamic obstacles (positions and velocities) every 200 ms.

The first experiment is designed to to evaluate the performance of our planner with various levels of sensor error. We use a simulation environment with moving obstacles as shown in Figure 4. There are two static (green) and two moving (red) obstacles. We plan a trajectory for the right arm of the PR2, which has 7 degrees of freedom, from a start configuration to a goal configuration. The obstacles move along a pre-determined trajectory, which is unknown to the planner. The planner uses replanning to compute a collision-free trajectory in this environment. During each planning step, the planner computes the conservative bound for each moving obstacle using Equation 2 and uses that bound to compute a trajectory. If the planner computes a collision-free trajectory for a given time step, ITOMP allows the robot to execute the planned motion during the next time step. This replanning step is repeated until the robot reaches the goal configuration. If the robot collides with an obstacle during the execution, we count it as a failure. We measure the success rate of planning and the cost of trajectory with different values of sensor noise. We repeat the test 10 times for each value of the sensor noise, and result is shown in Table 1. The costs of trajectories are the average costs corresponding to successful plans. It is shown that as the maximum sensor error increases, the success rate of the planner decreases and the cost (as shown in Equation 5) of the computed trajectory increases. For a successful planning instance with no collisions with the obstacles, this cost corresponds to the smoothness cost ($\frac{1}{2}\|\mathbf{AQ}\|^2$ in Equation 4): i.e. trajectories associated with higher cost values are less smooth than trajectories with lower costs.

For a succeeded planning result which has no collision, the cost mostly corresponds to the smoothness cost, i.e., trajectories with high costs are less smooth than other trajectories which have low costs. A large error value results in large conservative bounds for the moving obstacles, which reduces the search space for the planner to explore, and thereby it is harder to compute a feasible or optimal solution. However, we observe that at the maximum error of 30mm corresponding to PR2 robot sensors, our planner demonstrates good performance. We use this error value (30mm) in the following experiments.

In the second experiment, we test the responsiveness of our parallel replanning algorithm in dynamic environments with a high number of moving obstacles (Figure 5). In this environment, there are several moving obstacles which have the same speed and direction and some of them may collide with the arm of PR2 robot if the arm remains in the initial position. As in the first experiment, the planner uses the replanning approach to reach the goal position and avoid collisions with the moving obstacles. Figure 6 shows a planned trajectory and conservative bounds of moving obstacles. In this environment, we vary the speed of the obstacles, and measure the resulting success rates of the planning computations, as well as the cost functions corresponding to each computed trajectory. The performance data for each scenario (run for 10 trials per scenario) is laid out in Table 2. In this experiment, the planner successfully compute collision-free trajectory when the obstacles are moving at a slow speed. However, if the speed of obstacles is too high for the planner to avoid, the planner frequently fails to find a collision-free path. In each planning step, the planner finds a trajectory which avoids collision with the conservative bounds of the obstacles for the next time step (Equation 2). As the obstacle speed increases, the distance that an obstacle moves during a given time step is larger, and the resulting conservative bound for the rapidly-moving object covers a large area of the configuration space.

We also measure the performance of ITOMP with sets of different number of moving obstacles (Figure 7). We fix the size and speed of obstacles, varying only change the number of obstacles in each scenario. In this environment, a higher

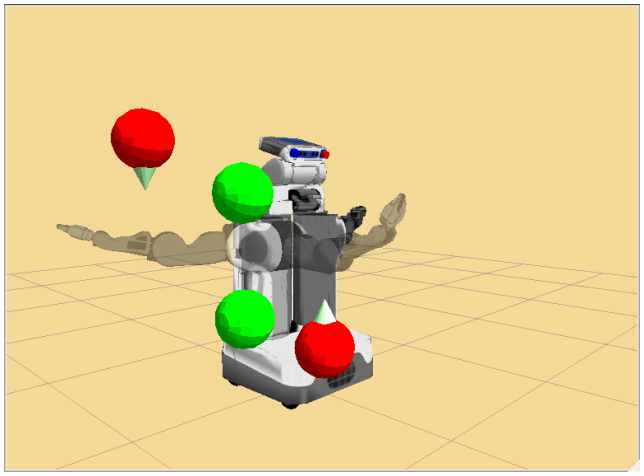
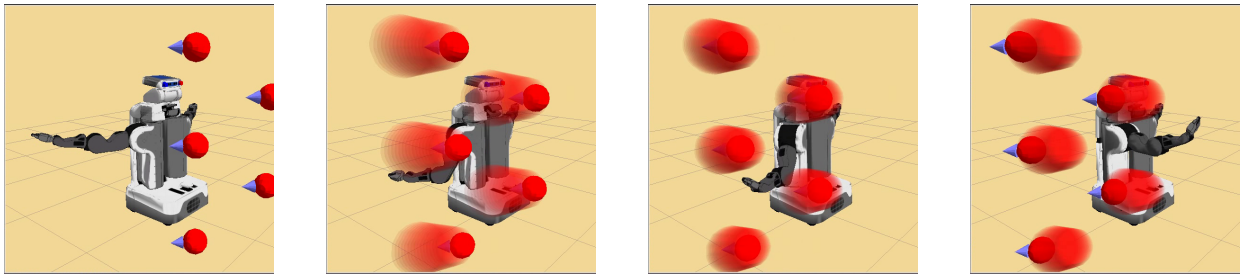


Figure 4: The planning environment used in experiments related to sensor noise. The planner computes a trajectory for the right arm of PR2 robot, moving it from the start configuration to the goal configuration while avoiding both static and dynamic obstacles. In the figure, green spheres correspond to static obstacles and the red spheres are dynamic obstacles.

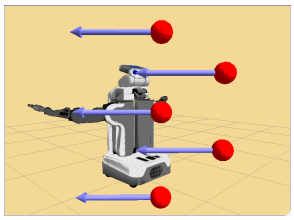
| sensor noise (mm) | # of successful plans | trajectory cost |
|---|---|---|
| 0 | 10/10 | 1.373 |
| 30 | 10/10 | 1.400 |
| 60 | 10/10 | 1.417 |
| 120 | 10/10 | 1.480 |
| 180 | 4/10 | 1.541 |

Table 1: Results obtained from sensor noise experiments. Success rate of planning and trajectory cost are measured with different sensor noise values. As the noise increases, the trajectory cost increases.

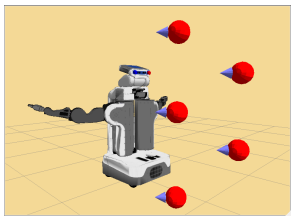


(a) (b) (c) (d)

Figure 6: A collision-free trajectory and conservative bounds of moving obstacles. (a) There are five moving obstacles. The arrows shows the direction of obstacles. (b)(c) During each step, the planner computes conservative local bounds on obstacle trajectory for the given time step. (b)(c)(d) The robot moves to the goal position while avoiding collisions with the obstacle local trajectory computed using the bounds. (d) The robot reaches the goal position.



(a) Planning environment with fast-moving obstacles

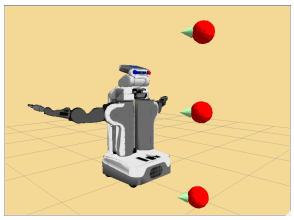


(b) Planning environment with slow-moving obstacles

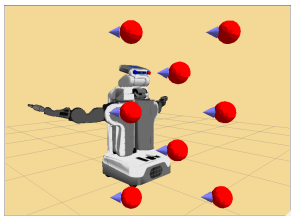
Figure 5: Planning environments used to evaluate the performance of our planner with moving obstacles with varying speeds. The planner uses the latest obstacle position and velocity to estimate the local trajectory. (a)(b) The obstacles (corresponding to red spheres) in the environment have varying (high or low) speeds. The size of each arrow corresponds to the magnitude of each's speed.

| obstacle speed(m/s) | # of successful plans | trajectory cost |
|---|---|---|
| 1 | 10/10 | 0.694 |
| 2 | 10/10 | 0.748 |
| 3 | 8/10 | 0.714 |
| 4 | 3/10 | 0.816 |

Table 2: Results obtained from experiments corresponding to varying obstacle speeds. The higher speed of obstacles lowers the success rate of planning and increases the trajectory cost.



(a) Planning environment with 3 moving obstacles



(b) Planning environment with 8 moving obstacles

Figure 7: Planning environments used to evaluate the performance of our planner with different numbers of moving obstacles.

| # of obstacles | # of successful plans | trajectory cost |
|---|---|---|
| 3 | 10/10 | 1.382 |
| 5 | 9/10 | 1.404 |
| 8 | 6/10 | 2.876 |

Table 3: Results obtained from the experiments with different number of moving obstacles. Success rate of planning and trajectory cost are measured. The success rate of the planner decreases when there are more obstacles in the environment.

number of obstacles result in reducing the size of collision-free space. The results are shown in Table 3. We observe that that a higher number of obstacles result in lower planning success rates and higher-cost trajectories.

## 6 Conclusion and Future Work

We present ITOMP, an optimization-based algorithm for motion planning in dynamic environments. ITOMP does not require a priori knowledge about global movement of moving obstacles and tries to compute a trajectory that is collision-free and also satisfies smoothness and dynamics constraints. In order to respond to unpredicted cases in dynamic scenes, ITOMP interleaves planning optimization and task execution. This strategy can improve the responsiveness and safety of the robot. We highlight the performance of the planning algorithm at guiding a model 7-DOF PR2 robot arm through various environments containing dynamic obstacles. We measured the algorithm's changing performance at differing levels of sensing error; in environments with dynamic obstacles moving at varying speeds; and in environments with varying numbers of dynamic obstacles.

Our approach has some limitations. Currently, we need to assume the overall time for trajectory computation and set the time for each waypoint on the trajectory before the optimization solver is used. In our implementation, we set the overall time to be $T$ and distribute the waypoints equally spaced in time. These two requirements may cause failure of the planner in some cases: for example, when a robot is at a crossroad and there is a moving object passing by in the perpendicular direction, the robot needs to wait for a certain time and then starts moving. In this case, it is difficult to know the time corresponding to the waypoint a priori and our method may not be able to compute a collision-free trajectory. One possible solution is to remove these as-

sumptions and solve the trajectory using dynamic programming similar to (Vernaza and Lee 2011). However, we need an incremental optimization solver to use interleaved planning and execution to obtain high responsiveness and safety. Recently, we have developed a parallel version of ITOMP for multi-core CPUs and many-core GPUs (Park, Pan, and Manocha 2012). In particular, we show that by exploiting commodity parallelism, we can considerably improve the responsiveness and performance of the planner.

## 7 Acknowledgments

This work was supported by ARO Contract W911NF-10-1-0506, NSF awards 1000579 and 1117127, and Willow Garage.

## References

Bekris, K., and Kavraki, L. 2007. Greedy but safe replanning under kinodynamic constraints. In *Robotics and Automation, 2007 IEEE International Conference on*, 704 –710.

Brock, O., and Khatib, O. 2002. Elastic strips: A framework for motion generation in human environments. *International Journal of Robotics Research* 21(12):1031–1052.

Chen, P., and Hwang, Y. 1998. Sandros: a dynamic graph search algorithm for motion planning. *IEEE Transactions on Robotics and Automation* 14(3):390–403.

Devaurs, D.; Simeon, T.; and Cortes, J. 2011. Parallelizing rrt on distributed-memory architectures. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2261 –2266.

Dragan, A.; Ratliff, N.; and Srinivasa, S. 2011. Manipulation planning with goal sets using constrained trajectory optimization. In *Proceedings of IEEE International Conference on Robotics and Automation*, 4582–4588.

Fiorini, P., and Shiller, Z. 1998. Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research* 17(7):760–772.

Garber, M., and Lin, M. C. 2004. Constraint-based motion planning using voronoi diagrams. In *Algorithmic Foundations of Robotics V*, volume 7 of *Springer Tracts in Advanced Robotics*. 541–558.

Gottschalk, S.; Lin, M. C.; and Manocha, D. 1996. Obbtree: a hierarchical structure for rapid interference detection. In *SIGGRAPH*, 171–180.

Hauser, K. 2012. On responsiveness, safety, and completeness in real-time motion planning. *Autonomous Robots* 32(1):35–48.

Hsu, D.; Kindel, R.; Latombe, J.-C.; and Rock, S. 2002. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research* 21(3):233–255.

Kalakrishnan, M.; Chitta, S.; Theodorou, E.; Pastor, P.; and Schaal, S. 2011. STOMP: Stochastic trajectory optimization for motion planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, 4569–4574.

Kavraki, L.; Svestka, P.; Latombe, J. C.; and Overmars, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580.

Koenig, S.; Tovey, C.; and Smirnov, Y. 2003. Performance bounds for planning in unknown terrain. *Artificial Intelligence* 147(1-2):253–279.

Kuffner, J., and LaValle, S. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, 995 – 1001.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

Likhachev, M., and Ferguson, D. 2009. Planning long dynamically feasible maneuvers for autonomous vehicles. *International Journal of Robotics Research* 28(8):933–945.

Likhachev, M.; Ferguson, D.; Gordon, G.; Stentz, A.; and Thrun, S. 2005. Anytime dynamic A*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Pan, J.; Lauterbach, C.; and Manocha, D. 2010. g-Planner: Real-time motion planning and global navigation using GPUs. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Pan, J.; Zhang, L.; and Manocha, D. 2011. Collision-free and curvature-continuous path smoothing in cluttered environments. In *Proceedings of Robotics: Science and Systems*.

Park, C.; Pan, J.; and Manocha, D. 2012. Real-time optimization-based planning in dynamic environments using GPUs. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill.

Petti, S., and Fraichard, T. 2005. Safe motion planning in dynamic environments. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2210–2215.

Phillips, M., and Likhachev, M. 2011a. Planning in domains with cost function dependent actions. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Phillips, M., and Likhachev, M. 2011b. SIPP: Safe interval path planning for dynamic environments. In *Proceedings of IEEE International Conference on Robotics and Automation*, 5628–5635.

Quinlan, S., and Khatib, O. 1993. Elastic bands: connecting path planning and control. In *Proceedings of IEEE International Conference on Robotics and Automation*, 802–807 vol.2.

Ratliff, N.; Zucker, M.; Bagnell, J. A. D.; and Srinivasa, S. 2009. CHOMP: Gradient optimization techniques for efficient motion planning. In *Proceedings of International Conference on Robotics and Automation*, 489–494.

Sud, A.; Otaduy, M. A.; and Manocha, D. 2004. DiFi: Fast 3d distance field computation using graphics hardware. *Computer Graphics Forum* 23(3):557–566.

van den Berg, J., and Overmars, M. 2005. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics* 21(5):885–897.

Vernaza, P., and Lee, D. D. 2011. Learning dimensional descent for optimal motion planning in high-dimensional spaces. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Wilkie, D.; van den Berg, J. P.; and Manocha, D. 2009. Generalized velocity obstacles. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5573–5578.