Reactive Integrated Motion Planning and Execution

Andreas Hofmann, Enrique Fernandez, Justin Helbert, Scott Smith, Brian Williams
Massachusetts Institute of Technology
77 Massachusetts Ave.
Cambridge, MA 02140

Abstract

Current motion planners, such as the ones available in ROS MoveIt, can solve difficult motion planning problems. However, these planners are not practical in unstructured, rapidly-changing environments. First, they assume that the environment is well-known, and static during planning and execution. Second, they do not support temporal constraints, which are often important for synchronization between a robot and other actors. Third, because many popular planners generate completely new trajectories for each planning problem, they do not allow for representing persistent control policy information associated with a trajectory across planning problems.

We present Chekhov, a reactive, integrated motion planning and execution system that addresses these problems. Chekhov uses a *Tube-based Roadmap* in which the edges of the roadmap graph are families of trajectories called flow tubes, rather than the single trajectories commonly used in roadmap systems. Flow tubes contain control policy information about how to move through the tube, and also represent the dynamic limits of the system, which imply temporal constraints. This, combined with an incremental APSP algorithm for quickly finding paths in the roadmap graph, allows Chekhov to operate in rapidly changing environments. Testing in simulation, and with a robot testbed has shown improvement in planning speed and motion predictability over current motion planners.

1 Introduction

The problem of moving a robot safely and effectively in uncertain environments is a challenging one. First, there is significant complexity due to the geometry of the robot and the environment, and these geometric relations can change in unexpected ways. Second, there are actuation limits and dynamics associated with a robot that limit speed and acceleration, and hence, may limit its ability to satisfy temporal constraints on tasks. Third, the actual task requirements may have significant complexity, and also, flexibility, which should be considered. Current motion planning and execution systems do not adequately address these requirements: they assume the environment is static, or at least, predictable; they do not support temporal constraints; and they do not adequately represent the true goals and constraints of a task.

In this paper, we describe Chekhov, a reactive motion execution system that addresses these requirements. Key innovations of Chekhov include: 1) representation of actual task requirements, including temporal and state constraints that capture available flexibility; 2) a representation of the control policy based on a roadmap graph [Boor *et al.*, 1999] where the edges of the graph are families of trajectories called flow tubes; and 3) an incremental All Pairs Shortest Path planning capability that can quickly provide path solutions from the graph, even when the environment changes. Chekhov observes temporal constraints and actuation limits, but utilizes task flexibility to achieve goals optimally, despite these limits. Chekhov supports the association of sophisticated control policies with trajectories to achieve this capability.

We endeavor to achieve a fast, reactive capability, and therefore use an approach inspired by grid-based incremental techniques [Cohen *et al.*, 2010]. In particular, we use a Tube-based Roadmap (TRM) representation that implements a two-layer control policy. The upper layer is implemented by a roadmap graph, combined with an incremental APSP algorithm that rapidly computes shortest paths through the graph. The lower level is implemented through Flow Tubes corresponding to each edge of the graph. Flow tubes contain control policy information about how to move through the tube, and also represent the dynamic limits of the system, which imply temporal constraints. Both upper and lower level control policies are continuously updated using incremental algorithms, as obstacles move through the environment.

The approach used in [Cohen *et al.*, 2010] uses a graph with regularly spaced nodes, edges representing single trajectories, and the D* lite algorithm for planning through the graph. In contrast, Chekhov uses a roadmap graph with irregular placement of nodes, edges representing Flow Tubes, and an incremental APSP based planner, developed previously in the operations research community [Demetrescu and Italiano, 2004], allowing for much faster replanning in many circumstances. Additionally, we consider temporal constraints, building on previous work in flexible discrete task execution systems [Morris *et al.*, 2001; Conrad and Williams, 2011] and hybrid task execution systems that compute feasible trajectory sets (Flow Tubes) to provide robust, high-performance control of dynamic systems [Hofmann and Williams, 2006; Hofmann, 2006; Tedrake, 2009; Hofmann and Williams, 2015].

2 Problem statement

The problem solved by Chekhov is to plan and execute robot motions that accomplish a task specified by a set of temporal and spatial constraints. The resulting motions must be optimal, or near-optimal, according to a specified objective function. The objective function can be used to optimize energy efficiency, robustness, speed, smoothness, and a variety of other objectives. The system should react, effectively, instantaneously to disturbances; it should act as if it always, "instantly" knows what to do, for any combination of goals and circumstances. This fast reaction is key to providing robots the capability to operate effectively in unstructured, uncertain, fast-changing environments. The capabilities of humans and other animals in this regard provide a good reference for the desired behavior we hope to achieve. This is in contrast to current systems that spend a lengthy period planning a motion, and then trying to execute it, assuming that nothing will change.

The inputs to Chekhov are: 1) an environment containing obstacles; 2) a plant model representing the actuation limits of the robot; 3) the current state of the robot; 4) a set of spatial and temporal constraints that represent goals to be achieved; and 5) an objective function for optimization. The outputs of Chekhov are control commands to the robot such that all constraints are observed, including achievement of goal regions and avoidance of obstacles, and such that its behavior is optimal according to the objective function. This can be expressed as

$$\begin{gathered} minimize\ c\left(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{u}, \dot{\mathbf{u}}\right) \\ such\ that\ \neg\left(\mathbf{q} \cap \mathbf{C}(\mathbf{E})\right) \\ \mathbf{q}_{min} < \mathbf{q} < \mathbf{q}_{max},\ \dot{\mathbf{q}}_{min} < \dot{\mathbf{q}} < \dot{\mathbf{q}}_{max} \\ \mathbf{f}_{min}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}\right) < \ddot{\mathbf{q}} < \mathbf{f}_{max}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}\right) \\ \mathbf{o}_{min}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}\right) < \left[\mathbf{q}, \dot{\mathbf{q}}\right]^T < \mathbf{o}_{max}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}\right) \\ \mathbf{g}_{min}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}\right) < \left[\mathbf{q}, \dot{\mathbf{q}}\right]^T < \mathbf{g}_{max}\left(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{t}\right) \\ d_{min} < d < d_{max} \end{gathered} \tag{1}$$

where c is the cost function, $\mathbf{q}$ is the robot joint pose vector, $\mathbf{u}$ is the robot actuation vector, $\mathbf{C}(\mathbf{E})$ is the collision space of the environment, $\mathbf{q}_{min}$ and $\mathbf{q}_{max}$ are the robot joint pose limits, $\dot{\mathbf{q}}_{min}$ and $\dot{\mathbf{q}}_{max}$ are the velocity limits, f_{min} and f_{max} are functions that express acceleration limits, $\mathbf{o}_{min}$ and $\mathbf{o}_{max}$ are functions that express operating constraints on robot state, $\mathbf{g}_{min}$ and $\mathbf{g}_{max}$ are functions that express goal constraints on robot state, d is the task duration, and d_{min} and d_{max} are bounds on this duration.

The inputs to Chekhov can change quickly and unexpectedly with time while the motion is being executed. For practical applications, changes fall into three categories: 1) the current state of the robot changes; 2) the goals to be achieved change; and 3) an environment obstacle moves in a way that affects the robot. Thus, we define a *disturbance* as such an unexpected change to task goals, environment, or robot state. Note that such a disturbance may be due to an actual physical change, or a change in the estimated state of the environment or robot (possibly because sensor data improves as the robot moves). The system's outputs must adapt to these changes in real time, so that the goals are still achieved, if possible.

We make a number of key assumptions. Although these assumptions may seem restrictive, we believe that they are consistent with a large class of practical robotic manipulation problems. First, we assume that the manipulation workspace is characterized by a limited set of pre-grasp poses. Second, we assume that the pre-grasp to grasp motion is short, and is best handled by visual and force servoing loops, rather than open-loop planners. Third, we assume that the collision environments are not overly complex. We are not trying to solve "piano mover" problems like reaching into tunnels or through a maze of obstacles. Instead, we assume that there is a small set of potential obstacles, such as a workpiece, a table, another robot, or a human, but that some of these may move (see Figure 1). The emphasis here is on achieving fast performance in typical, practical situations.

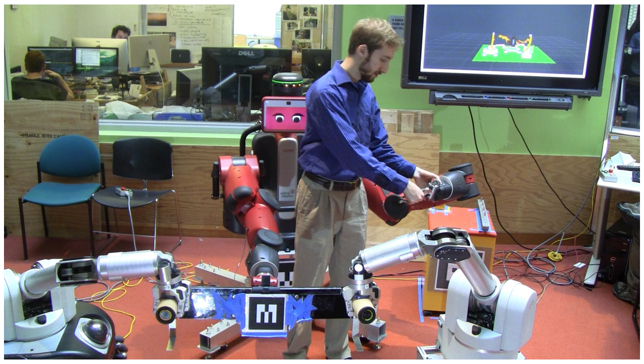


Figure 1: The collision space in this testbed is relatively simple, involving other robots, humans, and workpieces.

3 Implementation

Chekhov addresses the challenges of planning complexity in high-dimensional state spaces by using an integrated planning and execution architecture in which, the motion planner and executive cooperate closely, passing information back and forth. This blurs the sharp distinction between planning and control used in traditional systems. The planner component uses an incremental APSP algorithm to quickly find solution paths in a roadmap graph. Because the roadmap graph edges are flow tubes, rather than single trajectories, the Chekhov motion planner does not generate single reference trajectories, as standard motion planners do, but rather, generates a Qualitative Control Plan (QCP) [Hofmann and Williams, 2006; 2015]. A QCP represents families of feasible trajectories and associated control policies. In effect, this is a sophisticated control policy that observes the actuation limits of the robot specified in the plant model, and observes the specified temporal and spatial goal constraints, but also takes advantage of any flexibility implied by these constraints. Chekhov uses the control policy to respond to some, bounded disturbances, eliminating the need to call the motion planner in these cases.

This represents a first line of defense in reactive capability. Because such control policies are computationally expensive to compute, they are computed offline and then saved and re-used. This is a key distinguishing feature of the Chekhov system.

Figure 2 shows the main components of the Chekhov architecture. Chekhov appears, from the outside, as a black box that provides a comprehensive control policy; it outputs commands to the robot based on three inputs: a set of goals and constraints (see 1), a plant model representing the dynamics and actuation limits of the robot, and sensor data from the robot that is used to estimate robot and environment state. The control policy attempts to satisfy the goals and constraints, even when there are state and temporal disturbances.

A key component is a compiler that generates intermediate data structures to support fast, reactive behavior. These data structures are used by both the Motion Planner and the Motion Executive. The Executive contains State Estimator and Execution Monitor components, a Scheduler that decides the temporal duration of movement segments, and a Controller that implements the control policy.

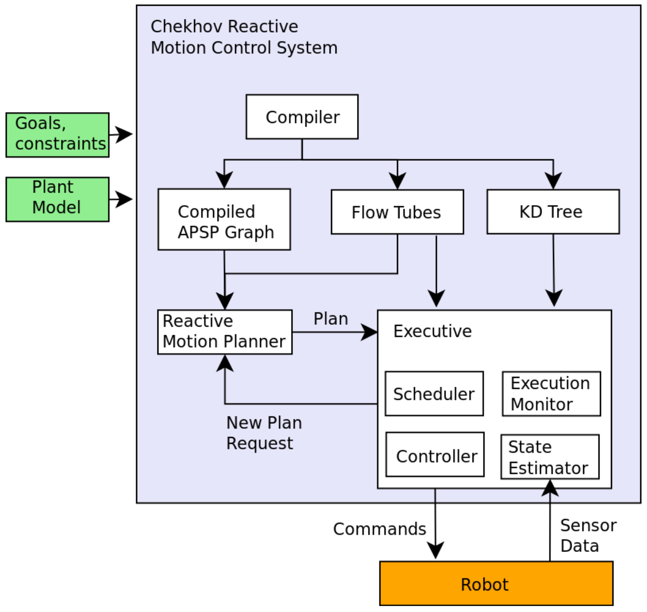


Figure 2: Architecture for Chekhov reactive motion control system.

3.1 Reactive Motion Planner

A key component of the Reactive Motion Planner is an incremental *All-Pairs Shortest Path* (APSP) algorithm [Demetrescu and Italiano, 2004], previously developed in the Operations Research community. An important feature of this algorithm, as applied to robot motion planning, is its ability to quickly replan when the robot goal state, current state, or the environment changes. A key disadvantage of the algorithm is its reliance on a discretization of the state space, resulting in computational tractability problems due to the curse of dimensionality. We address this dimensionality problem utilizing our first assumption: that the manipulation workspace is characterized by a limited set of pre-grasp poses. This assumption allows us to use search graphs with a relatively coarse discretization of poses (nodes). Some nodes in the graph represent the pre-graph poses, and other nodes represent random poses interspersed about the workspace to provide flexibility in avoiding obstacles. The edges in the graph represent families of trajectories (flow tubes) between the poses. The result is a comprehensive control policy for the entire manipulator workspace of interest.

This approach can be thought of as a compromise between industrial robot systems that simply play back a single trajectory, and algorithms like RRT* [Karaman and Frazzoli, 2011] that can, theoretically, cover all poses in a workspace. Chekhov covers only a small percentage of poses in a workspace, but it covers all the poses in the workspace of interest, given our assumptions.

Graph Representation of Search Space

The discretized states form the vertices of an *APSP graph*, defined as follows.

Definition 1 (APSP graph): A graph $G\left(V, E, w\right)$, where V is the set of vertices representing states, E is the set of edges representing movement of the arm from one state to a neighboring state, and w is a set of non-negative real edge weights. If an edge is blocked by an obstacle, then this edge weight is infinity. The discretization of states in the graph does not have to be regular; randomized arrangements can be used. We denote by w_{uv} the weight of edge (u, v), by π_{xy} a path from vertex x to y, and by $w\left(\pi_{xy}\right)$, the sum of the weights of edges in π_{xy}.

We augment the edges with flow tube information [Hofmann and Williams, 2006; 2015], which represents the relationship between the dynamic limits of the robot, and the feasible range of temporal durations for transitioning along the edge. The flow tubes associated with edges represent families of feasible trajectories and associated control policies. This explicit representation of multiple feasible paths provides a flexibility that is essential for motion execution in uncertain environments [Hofmann and Williams, 2006; 2015].

A description of the detailed implementation of flow tubes is beyond the scope of this paper; the reader is referred to [Hofmann and Williams, 2006], [Hofmann, 2006], and [Hofmann and Williams, 2015] for these details. Conceptually, a flow tube is defined as follows.

Definition 2 (Flow Tube): A flow tube, $F\left(R_i, R_g, D\right)$ is a set of trajectories $x(t)$ such that $\forall x(t) \in F\left(R_i, R_g, D\right)$, $x(0) \in R_i$, $x(t_f) \in R_g$, and $t_f \in D$.

Thus, R_i is a set of feasible initial states, R_g is the goal state set, and D is a set of controllable durations. The flow tube represents the fact that a trajectory exists that will go from anywhere in R_i to somewhere in R_g. Furthermore, the duration of the trajectory is controllable within the set D. The planner uses this information to ensure that plans that are generated satisfy all dynamic state constraints, obstacle constraints, and temporal constraints (that all constraints in 1 are satisfied). The executive uses the flow tube information to implement a comprehensive control policy that is capable of compensating for limited disturbances without requiring re-planning. In particular, the executive may adjust the scheduled duration of edge segments to compensate for temporal disturbances such as delays, so that the temporal constraints on the overall motion are satisfied.

A *valid motion plan*, output by the planner, is defined as follows.

Definition 3 (Valid Motion Plan): A valid motion plan from vertex x to y is a tuple $\langle \pi_{xy}, F \rangle$, where π_{xy} is a path as defined above, where $w\left(\pi_{xy}\right)$ is the minimum possible while satisfying the constraints specified in 1, and where each edge $E_j = (u, v)$ in π_{xy} has an associated flow tube F_j. Each F_j has an associated duration set, D_j. The duration sets must be consistent with the temporal constraints specified in 1.

In order to simplify planning, we require that the goal regions of incoming edges of a vertex be subsets of the initial regions of outgoing edges. Let $E_{in}(V)$ and $E_{out}(V)$ be the incoming and outgoing edges of vertex V. Then

$$\forall V \forall E_i = E_{in}(V) \forall E_o = E_{out}(V) \mid R_g(E_i) \subset R_i(E_o) \tag{2}$$

This ensures that in any plan generated by the planner, the goal region of a flow tube is always a subset of (fits inside) the initial region of its successor flow tube. If the planner is unable to find a plan that satisfies all the constraints, it returns failure.

Compilation

In order to support fast reactive operation, Chekhov performs two important compilations. These compilations are based on the APSP graph, and thus, are independent of particular goals or plans. They do take into account plant model information, including actuation limits of the robot.

First, Chekhov generates flow tubes corresponding to each edge of the APSP graph, as described above. Second, Chekhov generates a *dynamic* version of the APSP graph, which augments the graph with supporting information to allow for fast, incremental update when edge weights change (due to obstacles). Detailed explanation of the first compilation step is beyond the scope of this paper; we will focus on the second.

The key to the incremental APSP algorithm is that it maintains, for each pair of vertices in the graph, a priority queue of *potentially uniform paths* [Demetrescu and Italiano, 2004]. A path, π_{xy}, is uniform if every proper subpath is a shortest path. Note that this includes the case where π_{xy} is, itself, a shortest path. A path is potentially uniform if every proper subpath is a shortest path, or, if it was previously a shortest path, and none of its edges have been updated. Each potentially uniform path in the priority queue, $P_{xy} = \{\pi_{xy} : \pi_{xy} is potentially uniform in G\}$, has priority $w\left(\pi_{xy}\right)$. Thus, the first element in this queue is always the shortest path between x and y. This allows for extremely fast path queries.

During compilation, the priority queue is set up for each vertex pair, based on an environment that is free of moving obstacles. Initially, the priority queues just contain uniform paths. After compilation, as changes are made to edge weights due to the introduction or removal of obstacles, the priority queues are updated and potentially uniform paths may be introduced.

Fast Re-planning and Update of Dynamic APSP Graph

If a disturbance is severe enough that it cannot be handled by the Executive alone, then re-planning is required. If the disturbance is a change in goal state or current robot state, then re-planning is a simple lookup to retrieve the shortest path plan from the dynamic APSP data structure. When the environment changes due to obstacle introduction, removal, or movement, the dynamic APSP graph must be updated. Note that the flow tubes do not change.

Update of the dynamic APSP graph involves updating the priority queue of potentially uniform paths, for any vertex pair whose edge weight has changed due to an obstacle. Update for an edge (u, v) with a new weight w proceeds in two steps: cleanup and fixup [Demetrescu and Italiano, 2004]. In the cleanup step, all paths containing (u, v) are removed from the priority queue. In the fixup step, edge (u, v), with its new weight, is added to the priority queue P_{uv}. A new priority queue, containing only the minimum weight paths from the priority queues for each vertex pair is then generated. The algorithm then iteratively processes this queue, computing new potentially uniform paths taking into account the new edge. The overall update process is $O(n^2)$, and when it is finished, new path queries can be made.

3.2 Motion Executive

The Motion Executive executes the motion plan produced by the planner. Besides the motion plan, it accepts as inputs sensor data, and the plant model. The sensor data, combined with the model, is used by the State Estimator component of the Executive (see Figure 2) to generate optimal estimates of the robot and environment state. The motion plan, combined with the robot (plant) model are used to implement a control policy; the Executive applies the control policy to the state estimate to generate actuation commands for the robot.

Two components of the Executive implement the control policy. First, the scheduler decides on target execution times for each segment, j of the motion plan. This results in a schedule, S, consisting of a set of target execution times, $t_f(j)$ such that $t_f(j) \in D_j$ (see Definition 3). Thus, the scheduler generates a schedule that is consistent with the execution duration windows specified in the plan, and therefore, is consistent with all the temporal constraints. Second, the controller component attempts to execute each segment such that the motion segment completes at the target execution time. Thus, for a motion plan segment, j, the controller uses the flow tube, F_j, combined with the plant model, and taking into account state estimates, to generate a sequence of commands such that the trajectory reaches $R_g\left(F_j\right)$ at time $t_f(j)$. Under conditions of no disturbances, this is guaranteed to succeed, since the flow tube guarantees controllability for any time in D_j, and $t_f(j) \in D_j$.

The state estimate is also used by the Execution Monitor component of the Motion Executive to determine the current and also, the likely future status of plan execution. Thus, the Execution Monitor checks not only whether the current status of plan execution is satisfactory, but it also runs a prediction using the plant model to determine the likely future trajectory and status. Thus, the Execution Monitor continually checks for disturbances that jeapordize plan execution success. Note that disturbances include disturbances to the robot state, changes to the goal state, and changes to the environment state. Note, further, that a disturbance to the robot state can be an actual physical disturbance to the state (due to an unexpected collision, for example), or it can be a "virtual" disturbance due to an unexpected shift in the robot state estimate.

If a disturbance occurs, the Executive will first attempt to deal with it locally. First, the Executive determines whether the disturbance can be handled without changing the schedule. It applies the control policy expressed by the motion plan flow tubes, combined with the plant model to a forward simulation to predict whether the trajectory can still meet the goal at the scheduled time. If this is not the case, then the Executive attempts to find a new schedule, consistent with the allowed durations in the motion plan. If this fails as well, then the Executive requests a new plan [Hofmann and Williams, 2015].

4 Results

In order to test the Chekhov algorithms, we have integrated Chekhov into the OpenRAVE environment, and have imported a model of the Barrett WAM manipulator. Testing was performed in a simulated environment, in order to simplify the state estimation aspect of the problem. We also integrated Chekhov with the Open Motion Planning Library (OMPL), and through this, with ROS MoveIt!. The Chekhov motion planning component was implemented as a new planner OMPL, allowing for use in MoveIt!. Unfortunately, MoveIt!'s standard integration with OMPL assumes that planners start from scratch to solve each plan request, and therefore creates a new planner for each planning problem. This is a problem for Chekhov, as it would delete the compiled data structures that Chekhov relies upon. For this reason, we modified MoveIt! sourcecode so that planner instances are kept alive between planning requests.

Figure 3 shows execution of an example motion plan generated by the Motion Planner in the OpenRAVE environment. The arm moves from its initial pose to the final pose without hitting the table, or the block on the table. Generation of this initial trajectory was a simple lookup of the dynamic APSP structure, and so was, effectively, instantaneous. Fig. 4 shows the addition of an obstacle, and the plan generated by incremental replanning being executed.

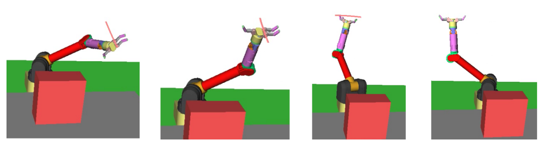

(a) (b) (c) (d)

Figure 3: Execution of initial motion plan.

In order to test the performance of the incremental update of the APSP structure (and therefore, of incremental re-planning), we repeated the obstacle test, 300 times, with cube-shaped boxes, ranging from 0.2 to 0.6 meters (edge length), placed at random points in the workspace. Table 1 shows results in terms of the number of edges affected by introduction of the obstacle, and the runtime for the update operation. As expected, the number of edges affected, and the runtime for update increase, on average, as box size increases.

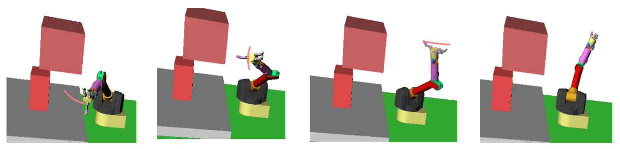
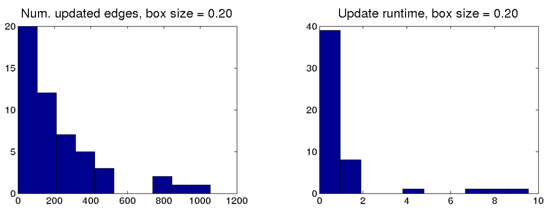

(a) (b) (c) (d)

Figure 4: Execution of adjusted motion plan, side view.

box sizes	0.2	0.3	0.4	0.5	0.6
avg. num updated edges	217	372	450	498	742
min. num updated edges	0	5	81	198	448
max. num updated edges	1056	1040	931	1031	947
avg. runtime	0.977	1.37	1.976	1.497	2.57
min. runtime	0.000	0.020	0.170	0.340	1.200
max. runtime	9.54	7.56	8.06	9.3	5.21

Table 1: Number of updated edges, and runtime for update, for randomly placed boxes (cubes) of edge length 0.2, 0.3, 0.4, 0.5, and 0.6 meters.

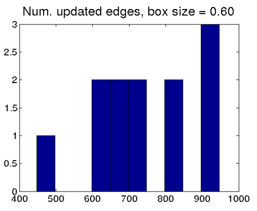
Figure 5 shows histograms of number of affected edges, and update runtimes for box size 0.2. Figures 6 shows corresponding histograms for box size 0.6. As expected, for the smaller box size, the histogram is weighted towards smaller numbers of edges, and shorter runtimes.



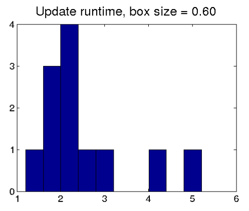
(a) (b)

Figure 5: Histogram of number of edges requiring update (a), and update runtime (b), in seconds, for box size 0.2.

These tests were performed using a Macbook Pro, with a



(a)



(b)

Figure 6: Histogram of number of edges requiring update, and update runtime, for box size 0.6.

2.3GHz Intel Core i7 (4 cores), running a Linux virtual machine.

In order to compare performance with other planning algorithms, we performed 200 tests using random start and goal poses, and a randomly generated obstacle. Table 2 shows timing performance of the Chekhov algorithm versus RRT [LaValle and Kuffner Jr, 2000], RRTconnect [Kuffner and LaValle, 2000], and Probabilistic Roadmap (PRM).

alg.	mean	std	min	max
Chekhov	0.0425	0.0213	0.0191	0.138
RRT	0.188	0.12	0.01	5
RRTconnect	0.042	0.04	0.014	0.45
PRM	0.12	0.08	0.07	0.2

Table 2: Timing performance of Chekhov vs. RRT and RRTconnect.

5 Discussion

Chekhov represents a domain-specific control policy, relevant to a particular type of workspace environment in the form of a Tube-based Roadmap (TRM). The roadmap graph includes nodes corresponding to pre-grasp poses required in the environment, as well as additional random poses to support obstacle avoidance. The edges of the graph correspond to Flow Tubes rather than the single trajectories common to standard roadmap approaches. The Flow Tubes contain control policy information, as well as dynamic information that relates state to temporal duration limits. Due to their complexity, the Flow Tubes must be generated offline. This persistency requirement rules out the use of many common planning algorithms that generate new trajectories from scratch for each new planning problem. However, use of a persistent roadmap graph requires updating when edges of the graph become occluded by obstacles. To address this, we incorporate an incremental APSP algorithm that provides fast path queries even when graph edge weights change.

Flow Tubes are especially useful for controlling underactuated systems, such as quadcopters or bipedal robots. However, if the goal is fast, safe, reactive motion, where actuation limits are pushed, even a manipulator becomes underactuated, requiring the use of sophisticated control policies, rather than simple trajectory following. Flow tubes also give information that relates the dynamic limits of a system to implied temporal constraints. Such constraints may come in conflict with user-imposed constraints.

As can be seen from Table 2, Chekhov significantly outperforms RRT and PRM in terms of average and worst-case times. The RRTconnect algorithm is comparable with Chekhov for average time, but is not as good for worst-case time. Besides these performance advantages, it is important to remember that unlike RRT and similar algorithms, Chekhov supports temporal constraints and association of saved control policies with trajectories. Because such control policies serve as a first line of defense against disturbances, they reduce the need for replanning when disturbances occur.

Upon further testing and investigation, we found that MoveIt! uses a goal pose initialization thread, that runs in parallel with planning, and blocks the start of planning until it is finished. Unfortunately, this thread uses a sampling approach that takes on the order of several 10s of milliseconds or more, which means that some of the results in Table 2 are likely dominated by this thread, rather than the actual planning. For example, further testing revealed that the actual APSP look-up to find the path was about 3 orders of magnitude faster than the results shown in the table. Further testing is needed to determine the actual planning times of the other planning algorithms, and to determine a better test for comparison. Ultimately, the initialization thread should be replaced with something more efficient.

As can be seen from the results in Table 1, performance is better when the number of affected edges is minimized. However, depending on where an obstacle occurs, the number of affected edges can become large. In particular, an obstacle placed near the manipulator base can greatly restrict motion of the manipulator's proximal link, thus eliminating a significant portion of nodes and edges, because the distal link poses for the restricted proximal link poses are all infeasible. It may be best to treat this situation as a special case, and switch to a different planning algorithm when this occurs.

An additional consideration is the fact that in a real testbed, obstacles don't appear out of thin air as they did in our simulated test; they move across the workspace. It is worth investigating whether the number of edges affected by obstacle movement greatly reduces the number of edges that have to be updated, as long as the update can be performed frequently enough.

We are currently testing Chekhov in a testbed (Figure 1), using actual, rather than simulated robots. This includes testing of faster movements, where the manipulator's actuation limits become important, and Flow Tube control policies must be used. Preliminary results, in an application involving human-robot interaction for cooperative assembly, show a noticeable improvement in planning speed and predictability of robot motions over existing planners. Both types of improvement are extremely beneficial for this type of application.

An important challenge, when using our approach, is deciding the nodes in the roadmap graph, particularly the ones corresponding to pre-grasp poses. While we have developed tools to help automate this process, it still currently requires human intervention to validate pre-grasp poses. This is analogous to the type of work needed to program single repetitive paths for industrial robots. Further research is needed to better automate this aspect of installation. Our system does automatically generate the additional nodes needed to support planning around obstacles, using a sampling approach. Once the graph nodes are determined, computation of the Flow Tubes corresponding to the graph edges is fully automated.

A second challenge, when using our approach, is the fact that the roadmap typically provides coverage of only a small fraction of the configuration space. For practical cases, where our assumptions hold, this is not a problem. However, we are investigating integration of trajectory optimization techniques that support adjustment of goals from poses in the graph, to neighboring poses outside the graph. Recent advances in trajectory optimization [Schulman *et al.*, 2014] make this a promising avenue of research.

In future work, we will investigate anytime, sub-optimal versions of the incremental APSP algorithm. In particular, the Potentially Uniform Paths queue could be leveraged for such a capability, in that it contains valid, but sub-optimal paths. This may allow for a very fast, but sub-optimal update consisting only of the cleanup part of the algorithm, after which, a path query can be made. The fixup part of the algorithm could then be run in background.

We believe that ultimately, parallel architectures show great promise for dramatically improving the capabilities of robotic planning and execution systems. We are currently investigating deployment of Chekhov, including the incremental APSP algorithm on parallel architectures, in order to speed up obstacle detection and dynamic update of the APSP graph.

Acknowledgments

This work was supported, in part, by Boeing Corp., under contract MIT-BA-GTA-1.

References

[Boor *et al.*, 1999] Valérie Boor, Mark H Overmars, and A Frank van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Robotics and automation, 1999. proceedings. 1999 ieee international conference on*, volume 2, pages 1018–1023. IEEE, 1999.

[Cohen *et al.*, 2010] Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for manipulation with motion primitives. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2902–2908. IEEE, 2010.

[Conrad and Williams, 2011] Patrick R Conrad and Brian C Williams. Drake: An efficient executive for temporal plans with choice. *J. Artif. Intell. Res.(JAIR)*, 42:607–659, 2011.

[Demetrescu and Italiano, 2004] Camil Demetrescu and Giuseppe F Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)*, 51(6):968–992, 2004.

[Hofmann and Williams, 2006] A.G. Hofmann and B.C. Williams. Exploiting Spatial and Temporal Flexibility for Plan Execution of Hybrid, Under-actuated Systems. In *AAAI 2006*, 2006.

[Hofmann and Williams, 2015] Andreas Hofmann and Brian Williams. Temporally and spatially flexible plan execution for dynamic hybrid systems (to appear). *Artificial Intelligence Journal, special issue on AI and robotics*, 2015.

[Hofmann, 2006] Andreas Hofmann. *Robust execution of bipedal walking tasks from biomechanical principles*. PhD thesis, Massachusetts Institute of Technology, 2006.

[Karaman and Frazzoli, 2011] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[Kuffner and LaValle, 2000] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[LaValle and Kuffner Jr, 2000] Steven M LaValle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *Proceedings Workshop on the Algorithmic Foundations of Robotics*. Citeseer, 2000.

[Morris *et al.*, 2001] Paul Morris, Nicola Muscettola, Thierry Vidal, et al. Dynamic control of plans with temporal uncertainty. In *IJCAI*, volume 1, pages 494–502. Citeseer, 2001.

[Schulman *et al.*, 2014] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[Tedrake, 2009] Russ Tedrake. Lqr-trees: Feedback motion planning on sparse randomized trees. 2009.