2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

Sampling-based Motion Planning With Dynamic Intermediate State Objectives: Application to Throwing

Yajia Zhang Jingru Luo Kris Hauser
School of Informatics and Computing, Indiana University Bloomington
{zhangyaj,luojing,hauserk}@indiana.edu

***Abstract*— Dynamic manipulations require attaining high velocities at specified configurations, all the while obeying geometric and dynamic constraints. This paper presents a motion planner that constructs a trajectory that passes at an intermediate state through a dynamic objective region, which is comprised of a certain lower dimensional submanifold in the configuration/velocity state space, and then returns to rest. Planning speed and reliability are greatly improved by finding good intermediate states first, because the choice of intermediate state couples the ramp-up and ramp-down subproblems, and moreover very few (often less than 1%) intermediate states yield feasible solution trajectories. Simulation experiments demonstrate that our method quickly generates trajectories for a 6-DOF industrial manipulator throwing a small object.**

I. Introduction

Highly dynamic manipulations such as throwing, catching, batting, and hammering can be used by a robot to apply large forces and to project actions across long distances. Such tasks require precise coordination in order to accelerate the robot to high speed to arrive at a particular configuration or set of configurations, while also remaining safe by avoiding collisions and joint limits. Safety is again a concern when decelerating to a stop. This paper considers applying sampling-based planners to dynamic manipulation planning by introducing a formulation of *dynamic intermediate state objectives* (DISO) that must be met at a single point in time in the *middle* of the trajectory. To our knowledge this form of goal has not yet been considered in the sample-based motion planning literature.

DISOs naturally decouple the problem into ramp-up and ramp-down phases, which are joined at an intermediate subgoal state in the DISO region. But naive application of a sampling-based kinodynamic planner to both pieces would be extremely inefficient for two reasons. First, DISOs are lower-dimensional submanifolds in the configuration/velocity state space, which means that control-based sampling will reach them with probability zero. Second, the fraction of the DISO region that is safely reachable and recoverable to a stop is extremely small because the robot must carefully choose its control inputs over a long period in order to obey acceleration, torque and collision constraints. This is particularly challenging because most trajectories considered by a standard kinodynamic planner will lead to inevitable collision.

In order to plan quickly, we designed our planner to exploit knowledge of these problem characteristics. We use numerical root-finding techniques to sample a subgoal state on the DISO subspace. To find a larger fraction of states within the small reachable subset of a DISO, we apply a fast filter based on acceleration upper bounds to reject obviously unreachable states. Then, the resulting states are used as endpoints for a bidirectional sampling-based motion planner that considers ramp-up and ramp-down planning independently. To optimize this planner we use a local planner that constructs dynamically-feasible interpolants between states, and employ a lazy strategy [4] that delays expensive collision and torque limit tests.

We apply the planner to enable a 6-DOF robot arm (Staubli TX90L) to throw small objects into a receptacle (e.g., a basket or trash can) in a known 3D environment. Our tests show that fewer than 2% of instantaneously feasible DISOs at a 2 m distance admit sufficient room to safely accelerate to the desired velocity and then decelerate to a stop. This fraction drops below 1% at 4 m distance. We demonstrate in simulation that our planner can reliably generate throwing motions at distances up to 8 m and with a variety of obstacles.

It has been observed that learning is critical to account for uncertainties in dynamic manipulations. Because our planner is versatile enough to handle a variety of environments and robot characteristics, it is a useful method to generate starting points to be fine-tuned via learning. We demonstrate in simulation how our planner can be used in this manner, using a classical iterative approach to correct for errors between the simplified grasp dynamics model in the planner and the more sophisticated model used in the rigid body simulator (a proxy for "real world" modeling errors).

II. Related Work

Dynamic manipulations involve highly nonlinear dynamics, uncertain mechanics of contact, and underactuated phases of object motion (e.g., flight), which pose difficulties for planning. Mason and Lynch clearly articulated the problem of dynamic manipulation and analyzed the dynamics of throwing a club with a one joint robot [11]. Furukawa et al. developed a high-speed multifingered hand with a high-speed vision system to perform dynamic regrasping by throwing a cylinder and catching it [7]. The errors from the model or the throwing process were compensated by the high-speed visual feedback. Katsumata et al and Mettin et al proposed control and optimization techniques to pitch a

978-1-4673-1405-3/12/$31.00 ©2012 IEEE

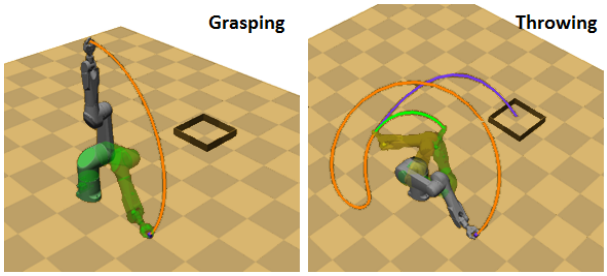


Fig. 1. Illustration of a throwing task: 1) Grasping. 2) Throwing: a ramp-up (orange curve) from the initial state (gray) to release state (yellow). The object's flight trajectory is drawn in purple. After release, a ramp-down (green curve) from the release state to a static configuration (green).

ball using an underactuated two-link robot arm, both in simulation and on a physical robot [10], [12]. These analytical approaches rely heavily on analyzing the equations of motion and constraints on the system, which makes it difficult to handle new scenarios, multiple objects, and obstacles in the environment, particularly with the high dimensionality of configuration/velocity state spaces of robot/object systems. Our work takes the approach of sampling-based motion planning (see Chapter 7 of [6]) which are adept at generating collision-free motions in high-dimensional spaces, but have not yet been widely applied to achieving dynamic manipulations.

This paper divides the stages of dynamic manipulations into discrete subproblems according to discrete contact states. The idea of dividing a complex motion planning problem into simpler subproblems has been studied in prior work in several areas including quasistatic manipulation [14], legged locomotion in rough terrain [5], and integration of symbolic tasks into motion planning [13]. These problems are instances of the multi-modal planning formulation that was introduced by [9] and was applied to simultaneous navigation and manipulation. Our current work involves simpler subproblem scheduling — there are two subproblems that can only be executed in fixed order — but in which each subproblem involves more complex motion planning queries that include dynamic constraints and goals. Our strategy to handling DISOs is similar to the workspace goal regions formulation of [3], but our planner addresses 1) a dynamic state space with velocity and acceleration bounds, 2) solving two motion planning sub-queries instead of one, since the validity of a candidate DISO state is contingent on the existence of a feasible trajectory that decelerates to a stop.

Learning approaches have been used for dynamic manipulations because outcomes are often sensitive to calibration errors and noise. Aboaf et al applied classical learning techniques to the problem of throwing a ball [1] as well as juggling a ball in three dimensions [2]. Using several practice iterations, the learning procedures adapt the task model and control policy to compensate for systematic inaccuracies. Such techniques require an initial policy that is sufficiently "close" to optimal in order to converge. We show that our planner can be used to generate good initial trajectories for learning, and that it can help retain trajectory feasibility during the adaptation process.

III. Planning with Dynamic Intermediate State Objectives

A. Problem Statement

We consider a robot R with configuration space $\mathcal{C}$ in an environment E. A *kinematically-feasible trajectory* $q(t) : [0,T] \mapsto \mathcal{C}$ is one in which $q(t)$ lies in $\mathcal{F} \subset \mathcal{C}$, the subset of collision-free configurations, and where $q(t)$ satisfies joint limits $|q(t)| \leq q_{max}$. Here $|\cdot|$ and $\leq$ denote element-wise absolute value and inequality, respectively. A *dynamically-feasible trajectory* satisfies certain differential constraints $f(q(t), \dot{q}(t), \ddot{q}(t)) \leq 0$ for all t. These constraints may include velocity bounds, $|\dot{q}(t)| \leq \dot{q}_{max}$, acceleration bounds $|\ddot{q}(t)| \leq \ddot{q}_{max}$, and torque bounds $|\tau(q(t), \dot{q}(t), \ddot{q}(t))| \leq \tau_{max}$.

The *dynamic intermediate state objective* (DISO) $\mathcal{D}$ is a subspace of the state space $\mathcal{C} \times \dot{\mathcal{C}}$ that is defined by requirements of the manipulation task. Typically $\mathcal{D}$ will require a point (or multiple points) on the robot attaining some range of positions and velocities (e.g., to launch a projectile to a desired target). The goal of the planner is to find a kinematically- and dynamically-feasible trajectory that passes from the start state x_{init} through $\mathcal{D}$ and ends in a goal state x_{end}. Both x_{init} and x_{end} are assumed to be at rest, and the configuration of x_{end} may or may not be given (in the latter case the planner may pick this arbitrarily).

It is often useful to instantiate a *task phase* upon reaching $\mathcal{D}$ in which the state evolves according to some deterministic procedure specialized to the task. For example, manipulation tasks that involve an impact, such as striking a rigid object, can be modeled by allowing the velocity $\dot{q}(t)$ to change instantaneously upon arriving at $\mathcal{D}$. Strategies for coping with uncertainty can also be implemented here. In our throwing implementation, we translate the end effector for a short duration along a parabolic arc at constant velocity to handle uncertainty in the gripper release speed.

B. Illustration: Throwing with a One-Joint Robot

Suppose a point projectile is launched by a one-joint robot whose joint lies at height $h = 2\,\text{m}$ from an arm of length $L = 1\,\text{m}$. The projectile is launched instantaneously when released and flies at a parabolic arc with vertical deceleration $9.8\,\text{m/s}^2$. The goal is to launch the projectile to intercept a point on the $h = 0$ plane at horizontal distance x from the joint. If the projectile is launched at state θ, $\dot{\theta}$, then its start position will be $(\cos\theta, \sin\theta + 2)$ and its velocity will be $(-\dot{\theta}\sin\theta, \dot{\theta}\cos\theta)$. Hence, the time t at which it will hit the $h = 0$ plane is given by the positive solution to the quadratic equation:

$$\sin\theta + 2 + t\dot{\theta}\cos\theta - 4.9t^2 = 0$$

and the horizontal distance traveled will be $\cos\theta - t\dot{\theta}\sin\theta$. With joint limits $[-\pi, \pi]$ and acceleration limit $|\ddot{\theta}| \leq 2\pi$, Fig. 2 illustrates the DISOs corresponding to $x = 2\,\text{m}$, $3\,\text{m}$, and $4\,\text{m}$. Here it is useful to distinguish between the *instantaneously-feasible* subset $\mathcal{D}_{IF}$, which is the subset of

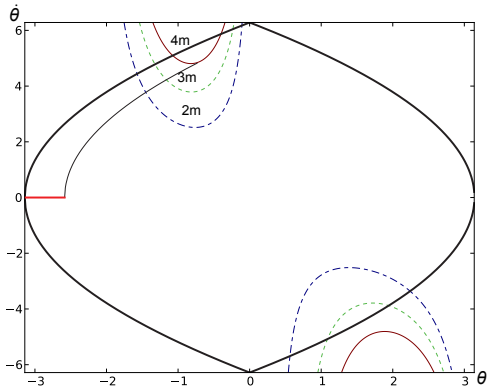


Fig. 2. The $(\theta, \dot{\theta})$ state space of a a one-joint robot that throws a projectile. DISOs according to 2 m, 3 m, and 4 m distance are illustrated. Dark parabolas bound the reachable set, due to acceleration bounds. To reach 4 m, the arm must accelerate at nearly maximum acceleration from a starting point near its joint limit $-\pi$ (highlighted region on θ axis).

states $(q, \dot{q}) \in \mathcal{D}$ satisfying $q \in \mathcal{F}$ and $|\dot{q}| \leq \dot{q}_{max}$, and the *dynamically-reachable* subset $\mathcal{D}_{DR}$, which is the subset of $\mathcal{D}_{IF}$ that is accessible from x_{init} *and* can recover to a state at rest. In Fig.2, $\mathcal{D}_{IF}$ is the entire DISO in the bounds of the plot, while $\mathcal{D}_{DR}$ is the portion within both acceleration bounds. Hence, in order to throw the projectile 4 m, the robot must initiate an underhand throw at near maximum acceleration starting near the joint limit $-\pi$, and then decelerate quickly before traveling more than $\approx 2\,rad$.

C. Sampling-Based Planning

A solution trajectory can be naturally divided into the following three sub-trajectories:

1) A ramp-up trajectory from x_{init} to a DISO state $x_{DO} \in \mathcal{D}$,
2) If applicable, a *task-phase* path from x_{DO} to the end task state x_{ET}.
3) A ramp-down trajectory from x_{ET} to a rest state x_{end}.

The first challenge that must be addressed is that $\mathcal{D}$ has small volume in the state space, and in fact has zero volume if $\mathcal{D}$ lies on a lower-dimensional submanifold. So, direct sampling-based planning has low (or zero) probability of solving Step 1. To address this we sample x_{DO} explicitly from $\mathcal{D}$, and then call a bidirectional sampling-based planner to connect x_{init} and x_{DO}. Step 3 is similarly solved to connect x_{ET} and x_{end}.

The second challenge is that due to acceleration, torque, and collision constraints, few states in $\mathcal{D}_{IF}$ are also in $\mathcal{D}_{DR}$. In the throwing example, this fraction shrinks below 1% as the target moves farther from the robot. But membership in $\mathcal{D}_{DR}$ is hard to test, so the planner will call many infeasible motion planning queries. To avoid this problem, we perform a quick reject test for x_{DO} and x_{ET} that checks only dynamic reachability from x_{init} and recovery to x_{end} without checking collision. This strategy is reminiscent of lazy evaluation [4], and can lead to an order of magnitude speedup. Our proposed algorithm is as follows.

For iteration $i = 1, 2, \ldots$, repeat:

1) Sample a candidate DISO state $x_{DO} \in \mathcal{D}_{IF}$.
2) Evaluate the endpoint of the task phase x_{ET}.
3) Check whether x_{DO} and x_{ET} are obviously dynamically unreachable.
4) If not, perform N_i iterations of planning a ramp-up trajectory to x_{DO}, and N_i iterations of planning a ramp-down trajectory from x_{ET}.
5) If both plans are successful, return the composite trajectory.

This algorithm depends on the parameter N_i that establishes the number of planning iterations for the i'th sampled release state. If it is too high, we run the risk of wasting time on an infeasible planning problem defined by a poorly chosen candidate release state. If it is too low, we may fail to find feasible paths, especially in cluttered environments. Our tests suggest the planner is relatively insensitive to the parameter setting, although we suggest incrementing N_i at every iteration because this may yield a probabilistically complete planner.

The subroutines used in Steps 3 and 4 will be described in more detail below.

D. Filtering Dynamically Unreachable States

To quickly reject candidate DISO states x_{DO} that are not dynamically reachable (Step 3) from a rest state we use a fast algebraic test based on a parabolic acceleration bound. Consider a single joint j and an upper bound a_{max} on its acceleration, where the bound is taken over all configurations and joint velocities. Consider accelerating or decelerating the joint state in the opposite direction of its velocity until the velocity is zero. If the resulting joint angle lies outside of joint j's limits, then we can be sure that x_{DO} is not reachable. A similar test is used to determine whether x_{ET} can recover to a rest state. Together, both tests quickly compute membership of x_{DO} in a set that forms an outer bound of $\mathcal{D}_{DR}$.

E. State-To-State Motion Planning

To implement the ramp-up motion planner of Step 4 we use a heavily modified SBL planner [15] that uses a bidirectional tree search in the configuration/velocity state space. For ramp-down motion planning we use a similar technique, except because the object is in flight we must use the configuration/velocity/time state space and consider the object as a moving obstacle. The planner is lazy in that in the first stage, the planner searches for a sequence of dynamically-feasible trajectory segments connecting feasible states. In the second stage, the more expensive collision and torque limit constraints are tested only along the trajectory.

First, we seed each forward tree with the maximum braking trajectory and each backward tree with the maximum reverse-braking trajectory. Then each tree is extended repeatedly by sampling a state x in the tree and a random candidate zero-velocity state x_{rand} in $\mathcal{F}$. We use the technique of [8] to construct a trajectory between x and x_{rand} that is dynamically-feasible with respect to the velocity and acceleration bounds $\dot{q}_{max}$ and $\ddot{q}_{max}$. If successful, we add

the midpoint state on the trajectory x_{mid} as a child of x and the endpoint x_{rand} as a child of x_{mid}. When the trees can be connected with a dynamically-feasible trajectory, we then check for collisions along the path. If satisfied, the shortcutting technique of [8] is used to smooth the path (typically reducing execution time by 30%). Finally, we find a time-parameterization of the path that satisfies torque limits. If any step fails, the infeasible edge is deleted from the tree and the process repeats.

IV. Application to Throwing

This section describes the application of the planner to a throwing task, which requires a task-specific subroutine for sampling DISO states $x_{DO} \in \mathcal{D}$ and a task-phase subroutine for generating x_{ET} from x_{DO}. We also implement an additional acceleration constraint based on the maximum gripper force, because high accelerations encountered during throwing may cause an object to slip out of the robot's grip.

A. Robot, Gripper, and Object Model

A fixed-base, fully-actuated robot arm is asked to throw an object at a desired target $T \in \mathbb{R}^3$ under known gravity and a known environment. Our model is that of the Staubli TX90L industrial robot arm with an attached gripper, taken from a model of Willow Garage's PR2 robot. We calculated that the hypothetical maximum object velocity reachable with our robot's given joint, velocity, and acceleration limits is approximately 12.7m/s, and can throw an object at most 16.5 m distance from the robot's base (using a combined "sidearm" and "overhead" throwing motion). It is unlikely to achieve such distances in practice because collision constraints prevent the robot from using the entire configuration space $\mathcal{C}$.

The object's geometry, position $P \in \mathbb{R}^3$, and mass characteristics are assumed known (4 cm cube of mass 0.5 kg). The gripper is assumed to consist of two parallel jaws which are placed on opposite sides of the object; this assumption will simplify our work in checking contact forces, and will also ensure that the object will separate cleanly (without significant rolling) upon release. With the gripper the robot has a reach of approximately 1.2 m, and a maximum jaw width of about 10 cm.

Before initiating the throw planner, we first direct the robot to grasp the object by sampling a candidate grasp and then querying the motion planner to plan a collision-free trajectory. If the query fails, a new grasp is picked and the process repeats. The initial state x_{init} of the throwing trajectory is taken at the end of the grasping trajectory[1].

B. Release State Sampling

As described below, to sample DISO states we first sample object release points and velocities at random that send the object on a ballistic arc to the target T. This point and velocity are then used as objectives for an IK solver. Finally, the ballistic arc is tested for collision.

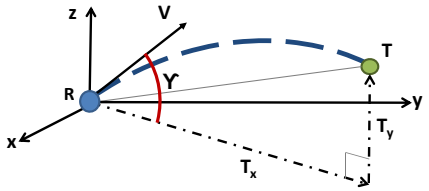


Fig. 3. After sampling a release point R, the object velocity V that causes the object to intercept the target T can be determined as an elementary function of a sampled release angle γ.

1) Sampling a Ballistic Object Trajectory: To sample an object releasing point R and velocity V that would cause the object's flight trajectory to intersect T, we first sample a hypothetically reachable release point R from the upper hemisphere with radius L centered at the robot's base, where L is the maximum extent of the robot arm. Then we find a velocity as follows.

Assuming negligible air resistance, the trajectory of a flying object follows a parabola. We wish to choose from the family of curves, parameterized by release angle, that connect the releasing point R to target point T. In an appropriate 2D reference frame F we simply must find a parabola intercepting the target's horizontal and vertical coordinates (T_x, T_y) relative to R (Fig. 3).

For release angle γ and speed $v = ||V||$, the parabolic trajectory $(x(t), y(t))$ satisfies the equations $x(t) = tv\cos\gamma$ and $y(t) = tv\sin\gamma - \frac{1}{2}gt^2$, where $g = 9.8$ m/s is the downward gravitational acceleration. Since the horizontal position T_x is achieved at time $t = T_x/(v\cos\gamma)$, some straightforward algebra shows that

$$v^2 = \frac{gT_x^2}{2\cos^2\gamma(T_x\tan\gamma - T_y)}. \quad (1)$$

Since v^2 must be nonnegative, we derive the constraint $\arctan(T_y/T_x) \leq \gamma \leq \frac{\pi}{2}$. So, we sample γ within this interval and then determine v from (1), and finally compute the release velocity vector V by projecting from the frame F back into world space.

2) Solving for a Release State Using Inverse Kinematics: To get the releasing configuration q_{DO}, we first sample configurations at random and use a Newton-Raphson IK solver to ensure the center of mass of the object lies directly on R. We also include a gripper orientation constraint to ensure that the opening of the gripper is parallel to the object velocity. We reject the sample if the inverse kinematics solver fails. Next, we determine the joint velocities $\dot{q}_{DO}$ that achieve V using the relation $\dot{q}_{release} = J^\dagger V$ where $J^\dagger$ is the pseudoinverse of the Jacobian matrix J evaluated at q_{DO}. Finally, we reject infeasible release states by checking whether q_{DO} is collision-free and $\dot{q}_{DO}$ is within velocity limits.

C. Object Release Task Phase

Since grippers cannot open instantaneously, the object will be released slightly after the robot initiates gripper

[1]For small objects under force closure, decoupling of the grasping and throwing stages does not jeopardize the reliability of the planner because feasibility is essentially invariant to the chosen grasp. In a more general setting, a planner may be more reliable if it considered multiple grasps.

opening. In order to compensate for this delay we employ a task phase that moves the arm at constant velocity slightly before and after it passes through the desired releasing state. We construct an adjusted prerelease configuration $q_{PR} = q_{DO} - \delta t \dot{q}_{DO}$ and move the robot arm with the constant joint velocity $\dot{q}_{DO}$ for $2\delta t$ while it opens its grippers. We choose $\delta t = 0.005\,s$ to approximate the time needed for the gripper to release contact with the object. The ending state of the task phase is then $x_{ET} = (q_{DO} + \delta t \dot{q}_{DO}, \dot{q}_{DO})$. The straight line path between q_{PR} and $q_{DO} + \delta t \dot{q}_{DO}$ is also checked for collision.

V. Simulation Results

Several experiments were performed on the throwing task in order to understand the characteristics of the DISO and its effect on planner behavior. All timing evaluations are conducted on an Intel i7 2.7 GHz PC using a single thread. Simulated executions of several of the examples in this paper can be viewed at `http://www.iu.edu/~motion/throwing/`.

A. DISO Reachability

First we aim to better understand the shape and volume of the instantaneously-feasible DISO subset $\mathcal{D}_{IF}$ and the dynamically-reachable subset $\mathcal{D}_{DR}$. For fixed releasing points R in a polar grid, we computed the fraction of release configurations, out of 20 samples, that yield instantaneously-feasible release velocities $\dot{q}_{DO}$. The left column of Fig. 4 plots the resulting success probabilities at 2 m and 4 m target positions. Next, we tested the fraction of those states that passed the quick rejection test of Sec. III-D. The middle column shows that 50% of states are rejected at 2 m distance while 83% are rejected at 4 m. This indicates that the use of the quick reject step improves planning speed by approximately an order of magnitude for larger distances.

Finally, the right column plots the overall fraction of instantaneously-feasible states that yield feasible ramp-up and ramp-down trajectories. At 2 m, this fraction is 1.2% overall, and decreases to 0.6% at 4 m. These indicate that the volume of $\mathcal{D}_{DR}$ relative to $\mathcal{D}_{IF}$ becomes miniscule as the distance to the target grows.

B. Variability with Respect to Target Distance

Next we examine planner running time with respect to increasing target distance. For each distance, 20 trials were run with different random seeds and initial configurations. A trial is considered a failure if no solution is found after 5 minutes. We found that at approximately 9 m distance, the running time increases substantially such that only 15% of the trials succeed (Tab. I). Interestingly, this steep increase occurs primarily because of collision constraints, because a significant fraction (about 5%) of DISO states are dynamically reachable in the absence of obstacles even at distance 8 m. One potential avenue for improving planning speed is to learn sampling biases toward the portion of $\mathcal{D}$ that is reachable when common obstacles are present (e.g., self collisions and collisions with a floor).

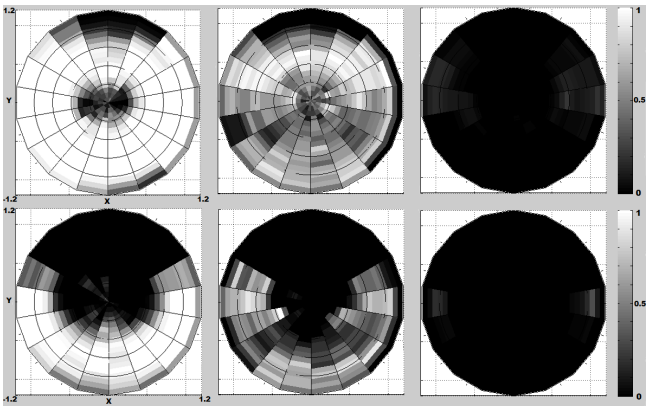


Fig. 4. As further constraints are tested on candidate DISO states (left to right), the fraction of feasible and reachable states drop dramatically. First we test for instantaneously feasibility $\mathcal{D}_{IF}$ (left); then filter dynamically unreachable states using the technique in Sec. III-D (middle); then perform a full test for dynamic reachability $\mathcal{D}_{DR}$ (right). Each plot denotes the fraction of states passing all prior tests for target positions y =2 m (top row) and y =4 m (bottom row), and for release points in a polar grid on an x-y plane fixed at height 1.0 m.

TABLE I

Performance vs. target distance over 20 runs with a 300 s cutoff. Standard deviations in parentheses.

Distance	Success rate	Time (s)
5 m	1.0	3.75(3.34)
6 m	1.0	9.91(5.67)
7 m	1.0	37.1(28.1)
8 m	0.9	119(88.6)
9 m	0.15	134(69.4)

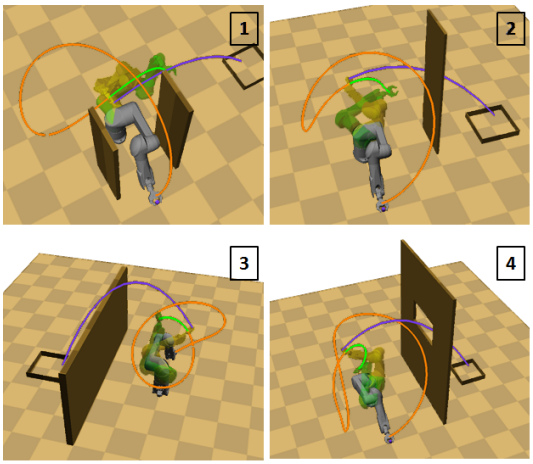
C. Robustness in Cluttered Environments

Next we tested our planner on each of 4 cluttered scenarios of varying difficulty, illustrated in Fig. 5. Scenario 1 adds an obstruction to the robot's movement. Scenarios 2 to 4 add a variety of obstacles to the object's movement. Over all 20 trials on each scenario with different random seeds and initial configurations, the planner was able to find a solution in under 10 seconds.

D. Correcting Execution Errors Using Iterative Learning

To execute the planned trajectories we use the Open Dynamics Engine rigid body simulator using a joint-controlled robot with feedforward torques. Particularly at high speeds, the landing position is sensitive to minor unmodeled gripper-object interactions and errors in the feedback controller. To compensate for these errors we use observed feedback from past trials to adjust the trajectory in an iterative fashion. Our unique contribution is that we are able to ensure that the generated trajectories always remain feasible.

The process iterates as follows. Set the initial target location $T_0 = T$, and let the planner produce an initial path $q_0(t)$, which is executed and the actual landing location L_0 is observed. On iteration $i = 1, 2, \ldots$, the target is adjusted to $T_i = T_{i-1} + k(T - L_{i-1})$ where k is a gain constant set to 0.5 in our implementation. The planner then "warps" $q_{i-1}(t)$ to a new trajectory $q_i(t)$ that throws the object to T_i.



#	1	2	3	4
Time (s)	6.57	0.66	0.58	0.69
Std. dev (s)	7.27	0.64	0.30	0.31

Fig. 5. (Top) Four scenarios with obstacles. (Bottom) Average and standard deviation of planning time over 20 runs.

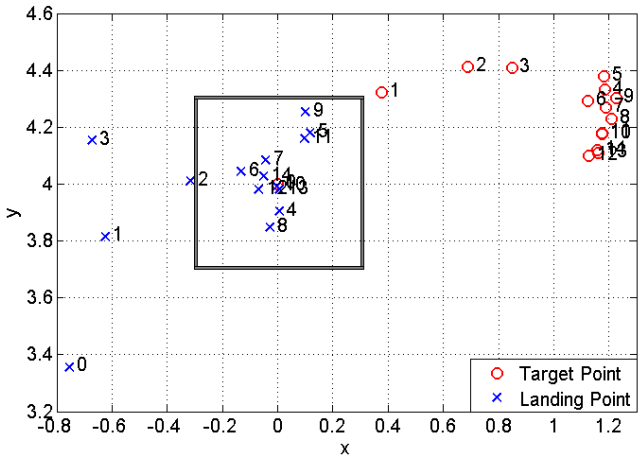


Fig. 6. Iterative adaptation of simulated target points to observed landing points. Each point is labeled by its iteration number.

The warping process is implemented by first using a Newton-Raphson root finding procedure to find a suitable new release point R and release state x_{DO}. These values are seeded with their prior values in q_{i-1} so that as L_{i-1} approaches T, the trajectory converges. Finally the local planner is used to connect the prior ramp-up and ramp-down trajectories to the DISO state. This is repeated for each of the milestones defining the trajectories. If these simple connections fail to be feasible, the ramp-up and ramp-down are planned from scratch using the state-to-state motion planner (Sec. III-E). If this finally fails after some elapsed time, the planner replans the trajectory *and* release state from scratch.

Fig. 6 shows a trial of the adaptation process for a 4 m throw after 14 iterations. The initial throw is off by over 1 m. After 4 iterations it consistently throws the target into the box, and after 12 the object lands consistently closer than 0.1 m from the target point.

This scheme is not perfect; precise long-distance throws are difficult to achieve. Although a feasible throw can be found according to the planner's internal model, at high speed the object's trajectory is quite sensitive to errors at the object-gripper contact. Future work might study methods to make releasing more robust to these errors.

VI. Conclusion

This paper presented the notion of dynamic intermediate state objectives (DISOs) and a strategy for addressing them using sampling-based motion planning. Significant speed gains were achieved by dividing the overall problem into subproblems that can be solved using two lazy bidirectional motion planning queries, and by employing a fast test for rejecting dynamically-unreachable states. The technique has solved several throwing scenarios for a 6DOF robot manipulator in which the target distance and complexity of obstacles were varied. It can also be used in an iterative learning framework to adjust for moderate amounts of execution error. In future work we hope to apply our ideas to different tasks like catching and batting, and to extend it to multi-stage tasks like dynamic locomotion.

References

[1] E. W. Aboaf, C. G. Atkeson, and D. J. Reinkensmeyer. Task-level robot learning: ball throwing. ai memo 1006. Technical report, Massachusetts Institute of Technology, 1987.

[2] E. W. Aboaf, S. M. Drucker, and C. G. Atkeson. Task-level robot learning: juggling a tennis ball more accurately. In *Proc. IEEE Intl. Conf. of Robotics and Automation*, pages 1290–1295, 1989.

[3] D. Berenson, S. Srinivasa, D. Ferguson, A. C. Romea, and J. Kuffner. Manipulation planning with workspace goal regions. In *IEEE International Conference on Robotics and Automation*, May 2009.

[4] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *IEEE Int. Conf. Rob. Aut.*, pages 521–528, San Francisco, CA, 2000.

[5] T. Bretl. Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem. *Int. J. Rob. Res.*, 25(4):317–342, 2006.

[6] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT press, 2005.

[7] N. Furukawa, A. Namiki, S. Taku, and M. Ishikawa. Dynamic regrasping using a high-speed multifingered hand and a high-speed vision system. In *Proc. IEEE Intl. Conf. of Robotics and Automation*, pages 181–187, 2006.

[8] K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *IEEE Intl. Conf. on Robotics and Automation*, May 2010.

[9] K. Hauser, V. Ng-Thow-Hing, and H. Gonzales-Baños. Multi-modal planning for a humanoid manipulation task. In *Intl. Symposium on Robotics Research*, Hiroshima, Japan, 2007.

[10] S. Katsumata, S. Ichinose, T. Shoji, S. Nakuara, and M. Sampei. Throwing motion control based on output zeroing utilizing 2-link underactuated arm. In *American Control Conference*, 2009.

[11] M. T. Mason and K. M. Lynch. Dynamic manipulation. In *Proc. IEEE Int. Conf. Intelligent Robots and Systems*, pages 152–159, 1993.

[12] U. Mettin, A. S. Shiriaev, L. B. Freidovich, and M. Sampei. Optimal ball pitching with an underactuated model of a human arm. In *Proc. IEEE Intl. Conf. of Robotics and Automation*, pages 5009–5014, 2010.

[13] E. Plaku, L. Kavraki, and M. Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Trans. Robotics*, 26(3):469–482, 2010.

[14] A. Sahbani, J. Cortés, and T. Siméon. A probabilistic algorithm for manipulation planning under continuous grasps and placements. In *IEEE/RSJ Int. Conf. Int. Rob. Sys.*, pages 1560–1565, Lausanne, Switzerland, 2002.

[15] G. Sánchez and J.-C. Latombe. On delaying collision checking in PRM planning: Application to multi-robot coordination. *Int. J. of Rob. Res.*, 21(1):5–26, 2002.