Preprocessing-based Methods for Robotic Manipulation

Yash Oza

CMU-RI-TR-22-29

July 15, 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Dr. Maxim Likhachev, *chair*
Dr. Oliver Kroemer
Dhruv Mauria Saxena

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To my family.

Abstract

Robotic manipulation is a key problem for several applications such as welding, pick-and-place, and automated assembly. However, motion planning for manipulation can be computationally expensive as it requires planning in the high-dimensional configuration space of the manipulator. Additionally, task-specific constraints such as strict time limits or constraints on end-effector motion further increase the complexity of this problem. In order to solve the planning problem tractably, several methods heavily rely on preprocessing, in which relevant information about the planning problem is gathered beforehand in order to enhance the performance of the planner. In this thesis, we propose algorithmic and system-level contributions to preprocessing-based methods for robotic manipulation.

In the first part of the thesis, we demonstrate how planning for constrained manipulation can be sped up by adding additional actions to the action set of a search-based planner. Macro-actions are ordered combinations of primitive actions and can help make faster progress towards the goal and reduce planning time. However, this introduces a trade-off as additional actions will increase the branching factor of the search. We leverage preprocessing to compute a set of macro-actions that provide probably approximately correct (PAC) bounds on the performance of the planner. We demonstrate the benefits of our approach on a container-opening task with a PR2 robot arm.

In the second part of the thesis, we present a planning and perception framework for intercepting projectiles using robotic manipulators. We focus on the problem of intercepting a projectile moving towards a robot equipped with a manipulator holding a shield. To successfully perform this task, the robot needs to (i) detect the incoming projectile, (ii) compute a trajectory that can intercept the projectile, and (iii) execute the found trajectory. These three steps need to be executed as fast as possible (<= 1 second in our setting) in order to maximize the number of episodes where the projectile is successfully intercepted. To this end, we propose a planning framework that constructs a trajectory database, wherein the individual trajectories can be used in order to intercept the projectile. Additionally, we also present a perception pipeline that continually provides state estimates of a fast-moving projectile within a short time window. We evaluate our approach both in simulation and on the physical PR2 robot with RGB-D cameras.

Acknowledgments

First and foremost, I want to thank my thesis advisor, Dr. Maxim Likhachev. Right from selecting me for a staff position at CMU to advising me during my Master's, you've played an instrumental role in my professional development at each and every stage. You gave me the opportunity to work on some really interesting projects and collaborate with lots of folks from the lab. Your method of approaching new problems, working with various stakeholders, and building simple yet efficient algorithms has been extremely inspiring. My time at CMU and the work that I did here would not have been possible without your guidance. I would also like to thank my thesis committee - Dr. Oliver Kroemer and Dhruv Mauria Saxena for providing me with their valued guidance that helped me accomplish the work discussed in this thesis.

I would also like to thank all members of the Search-Based Planning Laboratory for all the collaborations and friendships that we've built over the years. I'd also like to thank my friends from the MSR cohort that I've had the chance to interact with and learn from.

Most of all, I'd like to thank my family for the love and support that they've provided me with over the years. I will be forever grateful to all of you.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

Robots are increasingly being used to perform a wide variety of tasks around us. This includes using ground robots for performing delivery tasks [27], robotic manipulators for automated part assembly [24], and quadcopters for autonomous inspection and monitoring [32]. Different tasks that the robot has to perform present different constraints that the robot must satisfy in order to successfully complete the task. In this thesis, we focus on developing efficient motion planning algorithms for robotic manipulators.

In our first work, we demonstrate how planning for constrained manipulation can be sped up by adding additional actions to the action set of a search-based planner. Heuristic search-based planning grows a graph of candidate solutions for any planning problem by using a set of actions provided to it. Any solution found by these algorithms is composed of a sequence of these elementary 'motion primitives' joining the start and goal configurations. *Macro-actions* are longer-horizon actions composed of a sequence of the elementary motion primitives. Since they are longer-horizon actions, they can help the search make quick progress towards the goal, and potentially avoid local minima of the heuristic illustrated in Figure 1.1. At the same time, they increase the branching factor of the search as they add an additional successor state for every state expanded.

Prior work on using macro-actions with heuristic search has primarily concerned itself with finding one (or a few) good macro-actions after solving one (or a few) planning problems. There is no straightforward way to extend these approaches, which we list in Section 2.1, to compute a good *set* of macro-actions for use in planning. Adding elements to this set can increase planning times by increasing the branching factor, and macro-actions

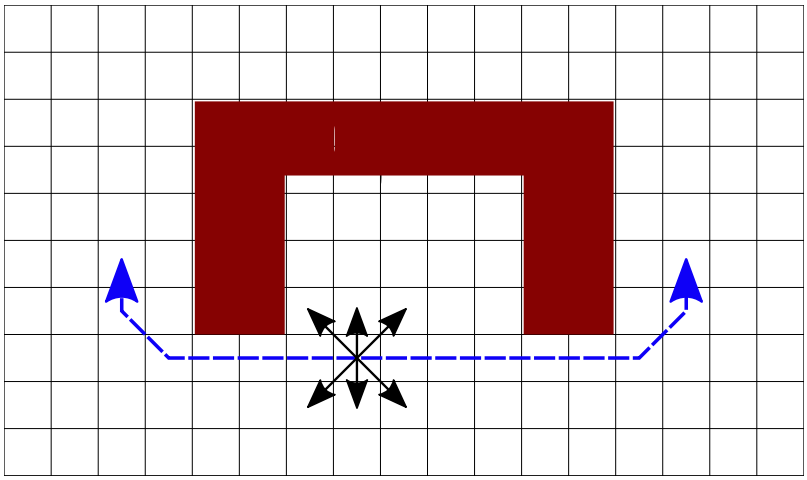


Figure 1.1: In this toy example, the macro-actions in the action space (blue) can help avoid the local minima represented by the cul-de-sac.

that perform well on some test cases might not render the same benefit in other cases. To the best of our knowledge, our work is the first to provide PAC style bounds for a set of macro-actions.

In our second work, we present a planning and perception framework for intercepting projectiles using robotic manipulators. Robots superior ability to perform (repetitive) tasks has made them ubiquitous in various industries. And with time we have also seen a transition in the kinds of environments these machines operate in; from extremely controlled settings to unstructured and unpredictable environments (like factory floors [26], space robotics [25], and domestic settings [16]). However, the unpredictability of the environment potentially poses safety hazards for the robots - for example, an object/machine part could unexpectedly fall onto the robot while it is operating on a factory floor or a stray space debris could collide with a robot manipulator mounted on a satellite. And it is important that these expensive machines are capable of protecting themselves during such unexpected accidents.

In this work, we focus on tackling a version of this problem, where a robot is tasked with protecting itself from incoming projectiles using a small shield attached to its manipulator arm (Fig. 1.2). Although repetitive, the time-critical nature of the task calls for an intelligent framework that can maximize the robots chances of intercepting projectiles. Based on the range of the vision system and average velocities of the incoming projectiles in our setting, the robot typically has about 1 second from when the projectile is first detected until it hits the body of the robot. Within this time frame, our framework should be capable of

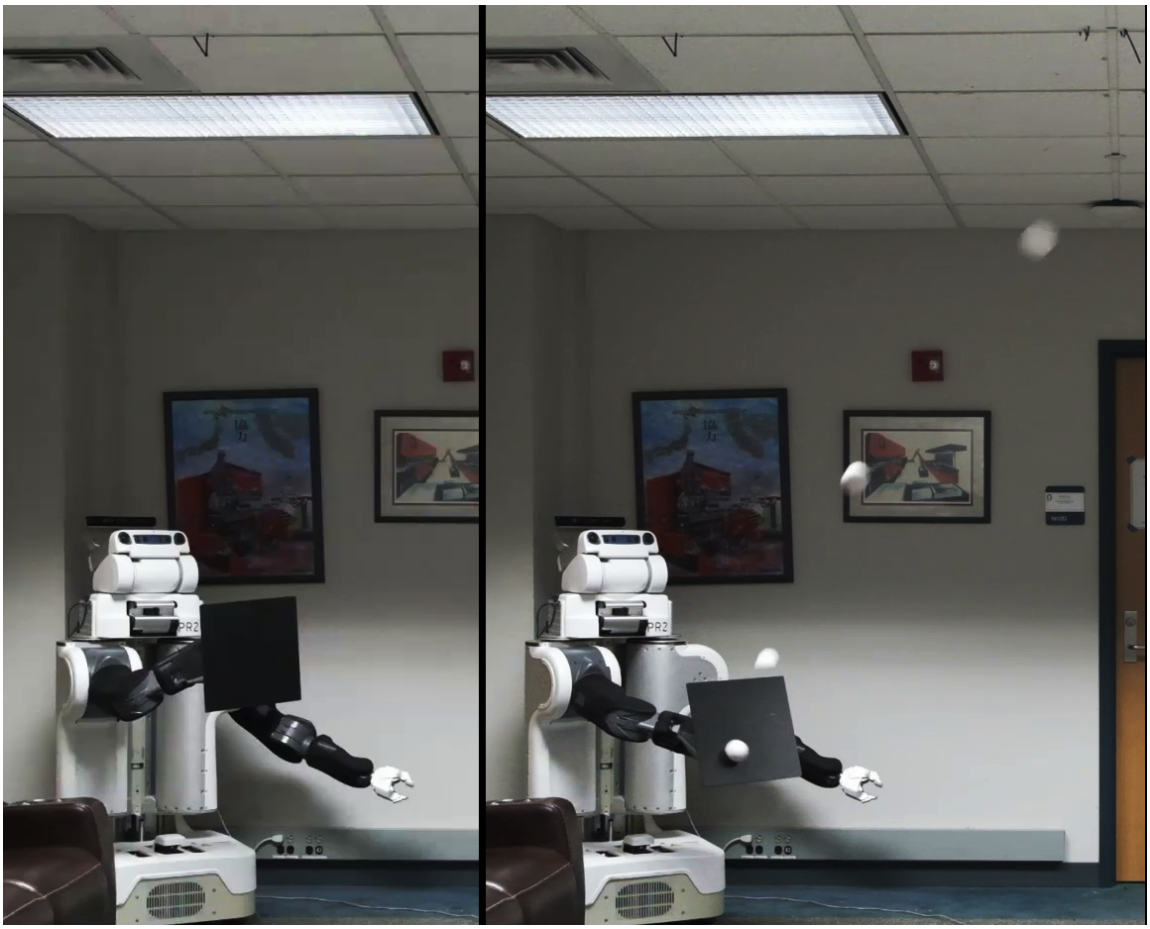


Figure 1.2: PR2 robot with a shield attached to the end-effector of its right arm (left). The robot executes a trajectory that successfully intercepts the incoming projectile (right).

performing 3 major tasks: (i) detecting and accurately predicting the motion of the incoming projectile, (ii) querying a motion planner for a manipulator trajectory that would enable the robot to safely intercept the projectile, and (iii) executing the trajectory returned by the motion planner in the real world. The fast nature of the task combined with the physical limitations of the robot (typical executions of blocking maneuvers consume about 700 of the 1000 milliseconds available to us), calls for a quick, yet accurate framework.

In this manuscript, we present our Bliss (BLink and you'll mISS) framework that comprises of (i) an off-the-shelf RGB-D sensor based perception module that provides continuous estimates of the incoming projectile and of its predicted trajectory, and (ii) a preprocessing-based planning pipeline based on the Constant-Time Motion Planning

(CTMP) paradigm [12] that guarantees to return a blocking trajectory if it exists (based on the perception estimates) within a significantly small time budget (1ms).

The perception module utilizes a simple point cloud filtering technique to detect the projectiles state (position and velocity) and predicts its future trajectory while accounting for air drag. These predictions continually improve through the course of the projectiles flight owing to the improvement in performance of the RGB-D sensor in closer proximity (as opposed to when the projectile is farther away from the vision sensor) and the ability to better predict the future trajectory of the projectile with more samples of its past states.

The extremely limited time budget available for planning forced us to resort to a preprocessing based planning module, wherein a database of manipulator trajectories starting from a fixed start configuration and leading to various interception configurations is created and queried online. Given the infinite space of interception configurations, an intelligent *dome-based* discretization technique was employed to limit the preprocessing time significantly while guaranteeing the existence of an interception configuration in this discretized set (if one exists at all) for all possible projectile trajectories.

Due to the time consuming nature of the execution phase ($\sim$700ms), it is initiated as quickly as possible based on the initial estimates of the projectile. However, when updated estimates are available from the perception system, the planning module dynamically reroutes the manipulator to a new interception configuration if required. These replanning/rerouting maneuvers are also created as part of the preprocessed database.

The performance of our framework was evaluated in simulation and in the real world using the Willow Garage PR2 robot with projectiles moving at speeds of up to 10m/s. An extensive ablation study and comparison with baselines has been presented to elicit the impact of our contributions.

Concisely put, our contributions include

- The formulation of the problem of robot protection against incoming projectiles and the development of the BLISS framework.
- A point cloud filtering based perception module that returns continuous estimates of the predicted trajectory of the projectile.
- A theoretically sound preprocessing-based motion planning module that guaranteedly returns blocking trajectories within an extremely small time window (including rerouting maneuvers).

- A *dome-based* discretization technique that makes preprocessing tractable while still providing strong guarantees.
- Demonstration of the effectiveness of our pipeline in the real world, by deploying it on a PR2 robot and an RGB-D camera setup.

Chapter 2

Learning Macro Actions

Heuristic search-based planning algorithms systematically build a graph using a specified set of primitive actions in order to find a least-cost path. This action set can be augmented with macro-actions which are ordered combinations of the primitive actions, and let the search evaluate successors which would otherwise have required multiple steps to be generated. Macro-actions can help make faster progress towards the goal and potentially reducing planning times, at the expense of an increase in the branching factor of the search and an increase in planning time. The trade-off affects the determination of a beneficial set of macro-actions for a planning domain, not just specific instances of the problem. In this work, we use results from probably approximately correct (PAC) learning to select a close-to-optimal set of macro-actions. We consider the domain of constrained manipulation with a robot arm, where robot motions are restricted to lie on a known lower-dimensional manifold. As the number of candidate macro-actions grows exponentially with the size of the primitive action set, we formulate the learning problem using multi-armed bandits, and devote computation power to macro-actions that seem promising for our domain. Our evaluation shows that the set of macro-actions selected by our approach outperforms baseline approaches for finding macro-actions.

2.1 Related Work

The idea of adding macro-actions to the action set for improving planning performance has been investigated in different domains. Macro-actions have been used in search-based

planning to jump across regions of the search space that might be expensive for the planner to evaluate without macro-actions [17]. Some existing approaches to find macro-actions look at information about planned solutions, such as the heuristic function profile along the path [6], or the evaluation function [11], and generate macro-actions based on some criteria. Another approach looks at the solution path itself and selects macro-actions based on some filtering criterion (possibly as simple as frequency counts of action sequences) [2]. Methods that generate macro-actions using genetic algorithms[23] have also been shown to work well in practice. The key difference between these algorithms and ours is bounded optimality of the solution and the explicit use of sets of macro-actions for heuristic search-based planning, rather than an arbitrary number of unrelated macro-actions.

2.2 Preliminaries

Let $\mathcal{X}$ denote the d-dimensional configuration space, $\mathcal{X}_{\text{free}} \subset \mathcal{X}$ the free space, and $\mathcal{X}_{\text{obs}} = \mathcal{X} \backslash \mathcal{X}_{\text{free}}$ the obstacle space. Let $\mathcal{A}$ be the basic action set available to a motion planner. $f : \mathcal{X}_{\text{free}} \times \mathcal{A} \rightarrow \mathcal{X}$ is the transition function such that $x' = f(x, a)$, where $x \in \mathcal{X}_{\text{free}}, a \in \mathcal{A}$ and action a is valid in state x. The successors of a state x, denoted by $\boldsymbol{x}$, are a set of configurations reachable from x given an action space $\mathcal{A}$. Formally, $\boldsymbol{x} = \textsc{GetSuccs}(x, \mathcal{A}) = \{x' : f(x, a_i) \, \forall \, a_i \in \mathcal{A} \text{ valid from } x\}$.

2.2.1 Motion Primitives and Macro-actions

Motion primitives are feasible atomic actions $a_i \in \mathcal{A} \, \forall \, i \leq |\mathcal{A}|$[1]. A heuristic search-based planning algorithm for robots systematically calls the GetSuccs function to build a lattice of states in $\mathcal{X}$ [30]. Each motion primitive a_i has an associated cost c_i. This means the cost of every feasible transition, $c(x, x') = c_i$ if $x' = f(x, a_i)$.

Definition 1. A *macro-action* $\boldsymbol{m}$ is any ordered sequence of motion primitives $\{a_1, \cdots, a_n\}$, such that $x' = f(x, \boldsymbol{m}) \in \mathcal{X}$ for $x \in \mathcal{X}_{\text{free}}, a_i \in \mathcal{A} \, \forall \, a_i \in \boldsymbol{m}$.

The transition function f that takes a macro-action as an argument is applied successively as we describe here. Consider a macro-action $\boldsymbol{m} = \{a_1, \cdots, a_n\}$. Let $x'_j = f(x'_{j-1}, a_j), 1 \leq j \leq n, a_j \in \boldsymbol{m}, x'_0 = x$. If action a_j is valid in state $x'_{j-1} \, \forall \, 1 \leq j \leq n$,

[1]For a four-connected grid, these are unit steps in the four cardinal directions. In our domain of robot arm manipulation, these correspond to individual movements of each joint.

$x' = f(x, \boldsymbol{m}) = x'_n$. Since a macro-action is made up of motion primitives, the cost of the transition $c(x, f(x, \boldsymbol{m})) = \sum_{i=1}^{n} c_i$. We denote a set of macro-actions as $\mathcal{M}_\mathcal{A}$ to make the dependency on action set $\mathcal{A}$ explicit. Our focus in this work is to determine the elements of $\mathcal{M}_\mathcal{A}$ such that we reduce planning times of heuristic search-based planning algorithms that use an augmented action space $\mathcal{A}' = \mathcal{A} \cup \mathcal{M}_\mathcal{A}$. Note that $\mathcal{M}_\mathcal{A} \subset \wp(\mathcal{A})$ where $\wp(\cdot)$ is the power set operator.

Planning Times with Macro-actions

Consider a planning problem defined as the tuple $\phi = (\xi, \mathcal{A})$ where the *environment* $\xi = (\mathcal{X}, x_s, x_g)$. $x_s \in \mathcal{X}_{\text{free}}$ and $x_g \in \mathcal{X}_{\text{free}}$ are the given start configuration and goal respectively. Let $t(\phi)$ denote the *planning time* for solving ϕ, i.e. the time taken by an algorithm to find a feasible path from x_s to x_g using action set $\mathcal{A}$.

For search-based planning algorithms, the branching factor of the search graph is $|\mathcal{A}|$. In general, $t(\phi)$ increases with the branching factor [6, 23]. However, macro-actions provide extended reachability in $\mathcal{X}$, which might reduce state expansions in the search, thereby decreasing $t(\phi)$. $t(\phi)$ is also affected by ξ as a change in the obstacle positions changes $\mathcal{X}_{obs}$, and different start and goals require traversal through different regions of $\mathcal{X}$.

Given a distribution of environments $\mathbb{P}(\Xi)$, we associate an expected planning time with $\mathcal{A}$,

$$t_\mathcal{A} = \mathbb{E}_{\xi \sim \mathbb{P}(\Xi)} t(\phi), \quad \phi = (\xi, \mathcal{A}) \tag{2.1}$$

For every environment ξ, there exists an optimal set of macro-actions $\mathcal{M}^*_\mathcal{A}(\xi)$ which trivially contains a singleton, the optimal path from x_s to x_g. However, such sets suffer from incredibly poor generalisation across $\mathbb{P}(\Xi)$, leading to large values of $t_{\mathcal{A}'}$ for $\mathcal{A}' = \mathcal{A} \cup \mathcal{M}^*_\mathcal{A}(\xi)$. Our goal in this work is to find a set of macro-actions that generalises well over $\mathbb{P}(\Xi)$. We do this by restricting ourselves to macro-actions of length at most n, and denote a set of such macro-actions as $\mathcal{M}_\mathcal{A}(n)$. We aim to solve the following optimisation problem for the optimal set $\mathcal{M}^*_\mathcal{A}(n)$,

$$\mathcal{M}^*_\mathcal{A}(n) = \underset{\mathcal{M}_\mathcal{A}(n)}{\arg\min}\, t_{\mathcal{A}'}, \quad \mathcal{A}' = \mathcal{A} \cup \mathcal{M}_\mathcal{A}(n) \tag{2.2}$$

Let $\wp_n(\mathcal{A})$ be a subset of $\wp(\mathcal{A})$ with all elements of length at most n, i.e. $\wp_n(\mathcal{A}) = \{\boldsymbol{m} : \boldsymbol{m} \in \wp(\mathcal{A}) \wedge |\boldsymbol{m}| \leq n\}$. We note that $\mathcal{M}^*_\mathcal{A}(n) \subset \wp(\wp_n(\mathcal{A}))$.

2.3 Algorithmic Framework

2.3.1 Problem Formulation

Since equation 2.2 is doubly exponential in $|\mathcal{A}|$, it is intractable for any reasonably sized $\mathcal{A}$. We relax equation 2.2 in two ways. Given $\mathbb{P}(\Xi)$, and $\mathcal{A}$ we,

1. Compute a set of macro-actions $\mathcal{M}_{\mathcal{A}}(n)$ from $\mathcal{A}$.
2. Find a probably approximately correct (PAC) set of macro-actions $\widehat{\mathcal{M}}^*_{\mathcal{A}}(n) \subset \wp(\mathcal{M}_{\mathcal{A}}(n))$ that is (ϵ, δ)-optimal.

Let $\widehat{\mathcal{A}}' = \mathcal{A} \cup \widehat{\mathcal{M}}^*_{\mathcal{A}}(n)$ and $\mathcal{A}' = \mathcal{A} \cup \mathcal{M}^*_{\mathcal{A}}(n)$. $\widehat{\mathcal{M}}^*_{\mathcal{A}}(n)$ is (ϵ, δ)-optimal if,

$$\mathbb{P}(t_{\widehat{\mathcal{A}}'} \leq t_{\mathcal{A}'} + \epsilon) \geq 1 - \delta \tag{2.3}$$

To determine an (ϵ, δ)-optimal set $\widehat{\mathcal{M}}^*_{\mathcal{A}}(n) \subset \wp(\mathcal{M}_{\mathcal{A}}(n))$ under the PAC framework, we require *iid* samples of planning times, i.e. compute $t(\phi)$ for $\phi = (\xi, \mathcal{A}')$ where $\xi \sim \mathbb{P}(\Xi)$ is a randomly drawn environment, and the action set $\mathcal{A}' = \mathcal{A} \cup \mathcal{M}(n)$ for any $\mathcal{M}(n) \in \wp(\mathcal{M}_{\mathcal{A}}(n))$. As discussed in Section 2.4.1, a lower bound on the number of such samples required (*sample complexity*) to obtain an (ϵ, δ)-optimal solution $\widehat{\mathcal{M}}^*_{\mathcal{A}}(n)$ is given by $\Omega(\frac{\left|\wp(\mathcal{M}_{\mathcal{A}}(n))\right|}{\epsilon^2} \ln(\frac{1}{\delta}))$. Since $\left|\wp(\mathcal{M}_{\mathcal{A}}(n))\right| = 2^{|\mathcal{M}_{\mathcal{A}}(n)|}$ grows exponentially with $|\mathcal{M}_{\mathcal{A}}(n)|$, computing an initial set of macro-actions $\mathcal{M}_{\mathcal{A}}(n)$ efficiently is crucial. Moreover, it is import that $\mathcal{M}_{\mathcal{A}}(n)$ contain good candidate macro-actions to keep the sampled values of $t(\phi)$ low, which should reduce the sample complexity for computing $\widehat{\mathcal{M}}^*_{\mathcal{A}}(n)$.

We model the problem of determining the (ϵ, δ)-optimal set of macro-actions $\widehat{\mathcal{M}}^*_{\mathcal{A}}(n) \subset \wp(\mathcal{M}_{\mathcal{A}}(n))$ as a best arm identification problem in stochastic multi-armed bandits (MABs). MABs form a powerful framework for problems in decision theory. Analogous to the MAB problem, we have a set of alternatives to choose from. In our case, each alternative (or arm) is a set of macro-actions $\mathcal{M}(n)$. Each of these alternatives has a stochastic reward that is drawn from a fixed unknown distribution and is independent of each other. In our case, the stochastic reward is inversely proportional to $t(\phi)$.

2.3.2 Compute a Set of Macro-Actions

Let $\gamma_\phi = \{x_s, x_2, \cdots, x_g\}$ be the ordered set of states that make up the solution path for a planning problem ϕ found by any heuristic search-based planner such as A* or Weighted A* [31]. Let τ_i be the timestamp at which $x_i \in \gamma_\phi$ was expanded during the search. We use $\Delta\tau_i = \tau_i - \tau_{i-1}$ as the expansion delay for state x_i. Intuitively, $\Delta\tau_i$ is the time spent by the search algorithm between expansions of consecutive states on a solution path. Large values of $\Delta\tau_i$ imply that the search did not make desirable progress towards x_g between expansions of x_{i-1} and x_i. This represents the amount of search effort required between expansions of states x_{i-1} and x_i. More generally, for $i < j, \Delta\tau_i^j = \sum_{k=i+1}^{j} \Delta\tau_k$ gives us the corresponding estimate for states x_i and x_j.

Given a threshold on search effort $\Delta\tau_{\text{thres}}$, we can construct a macro-action $\boldsymbol{m}_i^j$ for any $\Delta\tau_i^j \geq \Delta\tau_{\text{thres}}$. $\boldsymbol{m}_i^j = \{a_{i+1}, \cdots, a_j\}$ where $x'_u = f(x_{u-1}, a_u) = x_u$ for $(x_{u-1}, x_u) \in \gamma_\phi$ as before. To construct our candidate set of macro-actions, $\mathcal{M}_{\mathcal{A}}(n)$, Algorithm 2 solves a few randomly drawn planning problems $\phi = (\xi, \mathcal{A}), \xi \sim \mathbb{P}(\Xi)$, and appends macro-actions $\boldsymbol{m}_i^j$ to the set $\mathcal{M}_{\mathcal{A}}(n)$ until $|\mathcal{M}_{\mathcal{A}}(n)| = K$ for a pre-specified size K of $\mathcal{M}_{\mathcal{A}}(n)$. Algorithm 2 calls a sub-routine HeuristicPlanner which returns the solution γ_ϕ and the corresponding set of timestamps $\boldsymbol{\tau} = \{0, \tau_2, \cdots, t(\phi)\}$[2]. $\mathcal{M}_{\text{all}}$ is a max priority queue sorted in order of $\Delta\tau_{\text{total}}$ values which are associated with each macro-action and computed in ChainTopMPrims.

Our assumption in this approach is that macro-actions $\boldsymbol{m}_i^j$ computed in this way might help the planner avoid spending large amounts of search effort in the future for previously unseen planning problems drawn from $\mathbb{P}(\Xi)$ in the same way. However, a macro-action computed for one planning problem may not perform well in other planning problems from the same distribution, and so we still need to compute an (ϵ, δ)-optimal set of macro-actions $\widehat{\mathcal{M}}^*_{\mathcal{A}}(n) \subset \wp(\mathcal{M}_{\mathcal{A}}(n))$.

2.4 Find A Near-Optimal Set of Macro-actions

To determine an (ϵ, δ)-optimal set of macro-actions, we look into the algorithms that solve the best arm identification problems in MABs. Additionally, we are interested in PAC like

[2]With a slight abuse of notation, $\tau_{x_s} = 0$ since the start state is expanded at the outset of planning, and $\tau_{x_g} = t(\phi)$ since planning terminates when the goal state is expanded.

confidence bounds on the solution to guarantee the optimality of our selection. We also argue that it is more practical in terms of sample complexity to determine an (ϵ, δ)-optimal set of macro-actions, rather than the optimal set. Thus, we only discuss (ϵ, δ)-PAC class of algorithms, which are expected to return an ϵ-optimal solution with probability at least $(1 - \delta)$. We compare and contrast three such algorithms, namely Naïve (NV), Successive Elimination (SE), and Median Elimination (ME) [8]. This discussion informs our decision to not test the performace of ME for calculating $\widehat{\mathcal{M}}^*_{\mathcal{A}}(n)$. The SE algorithm provided in [8] is a $(0, \delta)$-PAC algorithm, but we convert it to a (ϵ, δ)-PAC version and present it in Algorithm 3.

Let $f : \mathbb{R}_{\geq 0} \rightarrow [0, 1]$ be some function that maps planning time $t(\phi)$ to rewards $r_{\mathcal{A}}$ associated with the action set used for planning, and let $\hat{r}_{\mathcal{A}} = f\left(\frac{1}{n}\sum_{j=1}^{n} t\left(\phi_j = (\xi_j \sim \mathbb{P}(\Xi), \mathcal{A})\right)\right)$ be the empirical reward. For our domain, rewards are inversely proportional to planning times, and an arm in the MAB framework refers to an element $\mathcal{M}(n) \subset \wp(\mathcal{M}_{\mathcal{A}}(n))$, so for simplicity of notation we denote r_i and $\hat{r}_i$ to be the properties of element i of $\wp(\mathcal{M}_{\mathcal{A}}(n))$.

2.4.1 On Practicality of Near Optimal Arm Identification

From [8] we know that the upper bound sample complexities for NV and ME are $O(\frac{n}{\epsilon^2}\log(\frac{n}{\delta}))$ and $O(\frac{n}{\epsilon^2}\log(\frac{1}{\delta}))$ respectively given n arms. The (ϵ, δ)-PAC SE algorithm also has a upper bound sample complexity of $O(\frac{n}{\epsilon^2}\log(\frac{n}{\delta}))$. The current lower bound on sample complexity for any algorithm that returns a single ϵ-optimal arm with probability at least $(1 - \delta)$ is $\Omega(\frac{n}{\epsilon^2}\log(1/\delta))$ [21]. While the average sample complexity of these algorithms have been well studied in the literature [5, 14, 21], we present some practical comments on choosing one among the three based on the problem at hand.

Algorithm 1 ComputeExpansionDelay(S, T)

Require: Path γ_ϕ and timestamps $\boldsymbol{\tau}$
Ensure: Set E of expansion delay tuple $(x_i, x_{i-1}\Delta\tau_i)$
1: $E \leftarrow \emptyset$
2: **for** $x_i \in \gamma_\phi \backslash x_s$ **do**
3: $\quad E \leftarrow E \cup (x_i, x_{i-1}\Delta\tau_i)$
4: **end for**

Algorithm 2 Create Macro Actions

Require: Set of Basic Motion-primitives $\mathcal{A}$. Distribution of environments $\mathbb{P}(\Xi)$. Threshold on expansion delay $\Delta\tau_{\text{thres}}$. Desired number of macro-action candidates K. Number of planning problems to consider L.

Ensure: Set of macro-actions $\mathcal{M}_{\mathcal{A}}(n)$

procedure ChainTopMPrims(E, $\Delta\tau_{\text{thres}}$, $\mathcal{A}$)
  $\mathcal{M} \leftarrow \emptyset$; $\boldsymbol{m} \leftarrow \emptyset$
  $\Delta\tau_{\text{total}} \leftarrow 0$; RecordMA $\leftarrow$ False
  for $(x_i, x_{i-1}, \Delta\tau_i) \in E$ **do**
    if $\Delta\tau_i \geq \Delta\tau_{\text{thres}}$ **then**
      RecordMA $\leftarrow$ True
      $\boldsymbol{m} \leftarrow \boldsymbol{m} \cup a_i$ ▷ $x_i = f(x_{i-1}, a_i)$
      $\Delta\tau_{\text{total}} \leftarrow \Delta\tau_{\text{total}} + \Delta\tau_i$
    else if $\Delta\tau_i < \Delta\tau_{\text{thres}}$ and RecordMA **then**
      $\mathcal{M} \leftarrow \mathcal{M} \cup (\Delta\tau_{\text{total}}, \boldsymbol{m})$
      $\boldsymbol{m} \leftarrow \emptyset$
      $\Delta\tau_{\text{total}} \leftarrow 0$; RecordMA $\leftarrow$ False
    end if
  end for
  return $\mathcal{M}$
end procedure
procedure CreateMActions($\mathcal{A}, \mathbb{P}(\Xi), \Delta\tau_{\text{thres}}, K, L$)
  $\mathcal{M}_{\text{all}} \leftarrow \emptyset$ ▷ Empty Priority Queue
  $i \leftarrow 1$
  for $i \leq L$ **do**
    $\phi \leftarrow (\xi \sim \mathbb{P}(\Xi), \mathcal{A})$
    $(\gamma_\phi, \boldsymbol{\tau}) \leftarrow$ HeuristicPlanner(ϕ)
    $E \leftarrow$ ComputeExansionDelay($\gamma_\phi, \boldsymbol{\tau}$)
    $\mathcal{M}_i \leftarrow$ ChainTopMPrims(E, $\Delta\tau_{\text{thres}}$)
    $\mathcal{M}_{\text{all}} \leftarrow \mathcal{M}_{\text{all}} \cup \mathcal{M}_i$
  end for
  return Top K macro-actions in $\mathcal{M}_{\text{all}}$
end procedure

Median Elimination (ME)

s presented above, this algorithm has the lowest upper bound sample complexity, which also matches with the lower bound sample complexity. At every iteration of the algorithm, it eliminates half of the arms, but the number of samples it requires per non-eliminated action at any iteration i is given by $\frac{64}{(\epsilon\times(3/4)^{i-1})^2}\ln(\frac{3\times 2^i}{\delta})$. Note that the sample requirement increases exponentially with i, and the algorithm does not adapt to the observed empirical rewards. Given ϵ, δ and n, the algorithm will return after $\log_2(n)$ iterations. On the other hand, NV requires $\frac{4}{\epsilon^2}\ln(\frac{2n}{\delta})$ samples per arm. The number of samples required per arm by ME at the first iteration ($i = 1$) becomes equal to that of NV when $n = 6^{16}/(2 \cdot \delta^{15})$. Thus, in practice NV would require significantly fewer samples in most of the cases until n is large.

Successive Elimination (SE)

The upper bound sample complexity for both (ϵ, δ)-PAC SE and NV algorithms are same. However, the former eliminates arms based on the error margins of the empirical rewards as captured by $\max_{j\in n(n)} \hat{r}_i(t) - \hat{r}_j(t) > 2\epsilon_t$ where $n(t)$ are the arms remaining at iteration t, and $\hat{r}_i(t)$ is the best empirical reward observed until iteration t. On the contrary, NV samples all the arms equal number of times in each iteration. Elimination in SE depends on the true rewards, and there might be cases when (ϵ, δ)-PAC SE would not eliminate any arm if all the arms are ϵ-optimal. In such cases, (ϵ, δ)-PAC SE would sample more arms than NV. This has contributed to the fact that (ϵ, δ)-PAC SE being an iterative algorithm needs to maintain an *any-time confidence bound* [9]. However, in some cases (ϵ, δ)-PAC SE return the best arm with sample complexity lower than that of NV. Thus, one may decide to

Algorithm 3 (ϵ, δ)-PAC Successive Elimination (SE)

Require: Number of arms n. δ. ϵ

$t \leftarrow 1$; $\quad \chi_t = \{1, 2, \ldots, n\}$
Pull each arm in χ_t once and compute $\hat{r}_{i,t}$
while $|\chi_i| > 1$ and $\epsilon_t > \epsilon_{\text{thres}}$ **do**
    $\chi_{t+1} = \chi_t \backslash \{i \in \chi_t \mid \max_{j\in\chi_t} \hat{r}_{j,t} - \hat{r}_{i,t} > 2\epsilon_t\}$
    Pull each arm in χ_t once and compute $\hat{r}_{i,t}$
    $t \leftarrow t + 1$
end while
return $\arg\max_{j\in\chi_t} \hat{r}_{j,t}$

use (ϵ, δ)-PAC SE over NV given some (even approximate) domain knowledge on the true rewards.

2.5 Experimental Results

We demonstrate our approach on a challenging constrained manipulation task, similar to the one presented in [29]. Specifically, the task is for a robot with a 7 Degrees-of-freedom (Dof) arm to open the lid of a container. The end-effector of the arm is constrained to lie on the manifold that the lid follows.

The distribution of planning problems, $\mathcal{D}$, that we use for coming up with the macro-actions are generated from the above mentioned setting. The start state is a 7-Dof position of the arm in the joint space. The goal state is specified in terms of the position of the end-effector of the arm. The planner has access to fourteen basic motion-primitives, wherein the arm can move any of the 7 joints in the positive or the negative direction. All the planning problems are solved using the ARA* planner, with a timeout of 600 seconds. To generate the initial set of plans that we use in Algorithm 1, we sample planning problems from the aforementioned distribution.

For our experiments, we sample 2 planning problems and plot the expansion delay plots for the obtained solution. The top five macro-actions are extracted and are then used to come up with the ϵ-optimal set of macro-actions.

We have evaluated our algorithm using the naive algorithm and successive elimination algorithms. We have also compared our algorithm with the existing state-of-the-art macro-action selection algorithms MACRO-FF[2] and Marvin [6]. Baseline without any macro-actions is referred to as "No MA".

2.5.1 Performance of (ϵ, δ)-optimal algorithms

Table 2.1: Performance of (ϵ, δ)-PAC algorithms (seconds)

	No MA	1-set	2-set	3-set	4-set	5-set
NV	46.85	1.41	0.17	0.13	0.07	0.08
SE	46.85	1.30	0.16	0.14	0.06	0.08

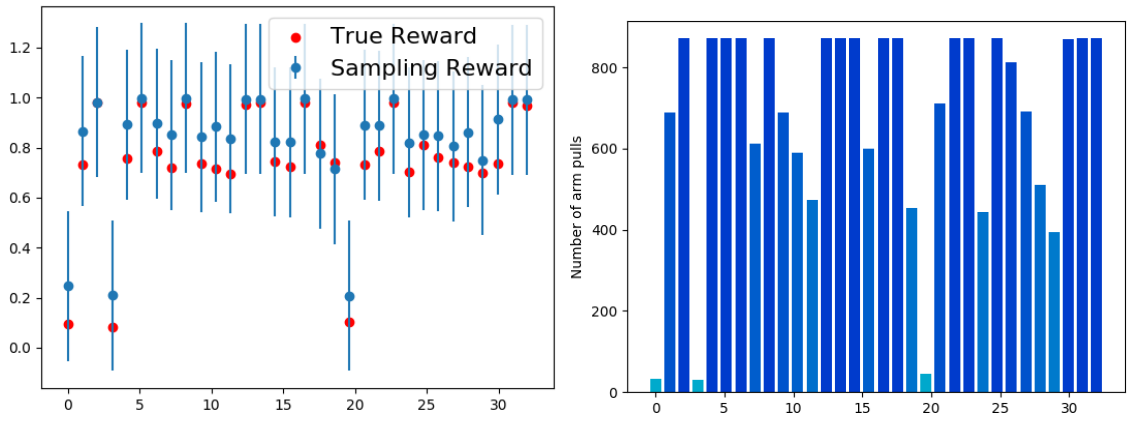


Figure 2.1: (L): True vs Empirical Reward in NV (R): Number of times each arm was sampled until termination in SE

Table 2.2: Comparison against baselines (seconds)

	No MA	Our 4-set	MACRO-FF	MARVIN
Planning Time	46.8597	0.06797	12.3813 (0.07932)	11.8961 (9.4367)

For running the (ϵ,δ)-optimal algorithms, we enumerate the entire power-set of the set of five macro-actions obtained previously. All the experiments are performed by setting ϵ = 0.3 and δ = 0.3 (70% confidence).

Naive algorithm

The naive algorithm is run using the above-mentioned ϵ and δ. For each of the arms, 134 different planning problems are sampled (based on the derivation in [8]) from the distribution D. Figure 2.1(L) is a plot that depicts the sample mean and ϵ (in blue) and the true mean (in red). The naive algorithm correctly identifies an ϵ-optimal arm. In the first row in Table 2.1, we can see the planning times for the best set of macro-actions for each cardinality of the set. We observe that the planning times are lowest for the set containing four macro-actions.

Successive Elimination algorithm

For successive elimination, the number of samples drawn for each arm is as shown in the Figure 2.1(R). During the runtime, SE is able to eliminate 17 of the 32 arms. It draws 872

samples for each of the arms that is remaining at the last iteration, and correctly identifies an ϵ-optimal arm. In the second row in Table 2.1, we can see the planning times for the best set of macro-actions after running the SE algorithm. We again observe that the planning times are lowest for the set containing four macro-actions.

Comparison against other methods

We compare our method with MACRO-FF and Marvin. MACRO-FF gives out a ranked set of macro-actions that can be used for any graph-search algorithm. Marvin gives out a set of macro-actions, but does not rank them. We evaluate the performance of MACRO-FF and Marvin based on two metrics - (1) planning time obtained while appending each macro-action individually to the action set, and (2) planning time obtained by appending a set of four macro-actions to the action set. We choose to append sets of macro-actions containing four elements because of the fact that set of four macro-actions had the least planning time when we ran our algorithm. Table 2.2 contains the values of planning times obtained using our algorithm, MACRO-FF and Marvin. The planning times mentioned in the parenthesis is the planning time obtained by appending four macro-actions to the set of basic motion primitives. We can see that our method gives lower planning times as compared to the other two methods.

2.6 Conclusion

We present an algorithm for computing a set of macro-actions that are Probably Approximately Correct. Our approach computes this set by solving a best-arm identification problem for a multi-armed bandit. In this work, since the multi-armed bandit problem has 2^N arms when given N macro-actions, we heuristically select our candidate set of N macro-actions. This leaves room both for selecting this set more intelligently, and using a more efficient best-subset selection algorithm for future work.

Chapter 3

Shield CTMP

In this work, we focus on the problem of intercepting a projectile moving towards a robot equipped with a manipulator holding a shield. To successfully perform this task, the robot needs to (i) detect the incoming projectile, (ii) compute a trajectory that can intercept the projectile, and (iii) execute the found trajectory. These three steps need to be executed as fast as possible ($\leq$ 1 second in our setting) in order to maximize the number of episodes where the projectile is successfully intercepted. To this end, in this work we propose (i) a Constant-Time Motion Planning-based planning framework that queries a pre-computed database of trajectories that can be used to intercept the projectile, and (ii) a perception pipeline that continually provides state estimates of a fast-moving projectile within a short time window. Further, as improved state estimates of the object are received from the perception pipeline, our planning algorithm provides replanned trajectories that can help intercept the projectile. We evaluate our approach both in simulation and on the physical PR2 robot with RGB-D cameras. On the physical robot, our pipeline is capable of intercepting projectiles moving at speeds of up to 10 meters/second using only RGB-D cameras, achieving a success rate of 70$\%$.

3.1 Related Work

There are several works in literature that have tackled the problem of object pose estimation, fast trajectory generation and execution. The work that is most closely related to our approach is [13]. Here, an experience-based planner [28] is used to generate a set of

root-paths, which are stored for online lookup. In addition, the authors also allow the use of a motion planner online, whose trajectories have been verified during the offline phase. Though our approach is similar, our task does not allow for any computation online that is more than a few milliseconds. One of the earliest in the domain of preprocessing-based motion planning is Probabilistic Roadmaps [15]. [10] is a comparative study of probabilistic roadmap planners.

[4] performs a similar task to ours, where the robot performs a ball catching maneuver using a mobile manipulator. A VICON motion-capture system is used to obtain very precise and high-frequency position and velocity estimates of the incoming objects. For the planning module, the authors formulate the catching problem as a bi-level optimization problem, which produces a joint configuration that can successfully complete the task. A key difference between [4] and our work is that they solve an optimization problem online while the projectile is in flight, and the optimization is not guaranteed to return a solution within a strict time-limit. Additionally, in our setting, we do not use a motion capture system as we believe that it is practically infeasible to assume that we know a very precise and high-frequency estimate of the incoming projectile. And moreover in a real-setting, given that there will be error in estimation, the planner should account for that. Thus, in our setting, we assume access only to off-the-shelf RBG-D cameras and a non-mobile platform.

RGB-D cameras are very noisy and thus require special filtering techniques to detect incoming projectile. Model specific methods like [1] attempts to fit a known 3D model to the point cloud to efficiently find the location of the projectile. However, this assumes that the model of the projectile is known. More generic approaches like [18] does not have the requirement.

Machine learning based methods have also been used to learn policies. [3], [22] use reinforcement learning in a model-free setting to learn policies that are capable of playing dynamic strokes for the table-tennis task. In contrast to our work, these works require significantly more data in order to learn useful policies.

3.2 Problem Formulation

Consider a manipulator connected to a robot body with a shield rigidly attached to the end-effector of the manipulator. A projectile is launched in the direction of the robot, and the manipulator is tasked with intercepting the projectile before it collides with the robot.

The state of the projectile, ρ, is represented as a tuple (ρ_p, ρ_v), where $\rho_p \in \mathbb{R}^3$ represents the position, and $\rho_v \in \mathbb{R}^3$ represents the velocity of the projectile. The goal of the problem is to intercept the projectile before it collides with the robot. To solve this problem, we make certain simplifying assumptions -

- The manipulator always starts from a “home" configuration s_{home}.
- When the manipulator is in motion, there are no objects other than the robot body and the incoming projectile that the manipulator can collide with.
- The projectile is launched from within the field of view of the camera, allowing us to get early estimates of its trajectory ρ.
- At any point in time, only a single object is launched in the direction of the robot.

Let $\mathcal{T}_t$ be the time of flight of the projectile, measured from the time it is first observed to when it collides with the robot. Let $\mathcal{T}_d$ be the time when the first state estimate of the projectile is obtained by the planning module from the perception module. Finally, let $\mathcal{T}_{const}$ be the time taken to retrieve a trajectory from the database, and $\mathcal{T}_e$ be the time taken by the manipulator to execute that trajectory.

In order for the manipulator to successfully intercept the object, two conditions need to be satisfied. First, given a long enough time-out, a motion planner must be able to compute a trajectory from s_{home} to a goal configuration which can intercept the projectile. Second, the manipulator also needs to ensure that it reaches the goal configuration before the projectile passes that location (in $\mathbb{R}^3$). Specifically, the following equation needs to be satisfied -

$$\mathcal{T}_t \geq \mathcal{T}_d + \mathcal{T}_e + \mathcal{T}_{const} \tag{3.1}$$

The goal of the perception module is to provide a projectile estimate ρ from time-series detections of the incoming projectile. It should also provide a new projectile estimate if a better estimate is available. Finally, the projectile estimate would be extrapolated to predict where would the incoming projectile hit the robot before it collides. Thus, we are interested in minimizing the prediction error.

3.3 CTMP-based Motion Planning

The constraints laid out by the problem call for deriving an approach that minimizes time spent on any online operation, hence budgeting as much time as possible for execution and perception. To this end, we propose an approach that is based on the Constant-Time Motion Planning (CTMP) class of algorithms [12]. At a high-level, we first solve a set of planning problems, and store their solutions in a database. During execution time, we observe the incoming projectile, and, if available, execute a trajectory that would intercept the projectile and prevent it from coming in collision with the robot body. In the remainder of the section, we briefly go over Constant-Time Motion Planning (Section 3.3.1), discuss a straw man approach (Section 3.3.2) that turns out to be impractical for our task, then discuss the proposed approach and its building blocks in detail (Section 3.3.3), and finally provide the complete algorithm (Section 3.3.5).

3.3.1 Constant-Time Motion Planning

A CTMP approach (as formailzed in [12]) develops a strategy to solve a set of planning problems $\mathcal{P}$ offline and use the pre-computed solutions to $\mathcal{P}$ online to perform the required task. In our setting, we have a fixed start stare s_{home} and a set of goals $\mathcal{G}$ which the robot needs to reach to in order to protect itself from the incoming object. To formally analyze our approach, we use the following definitions (originally presented in [12], presented here for completeness).

Definition 1: CTMP Algorithm. Let ALG be a motion planning algorithm, $\mathcal{T}_{bound}$ a user-controlled time bound and $\mathcal{T}_{const} < \mathcal{T}_{bound}$ a small time constant whose value is independent of the size and complexity of the motion planning problem. ALG is said to be a CTMP algorithm if it is guaranteed to answer any motion planning query within $\mathcal{T}_{bound}$.

Definition 2. Reachability. A goal $g \in \mathcal{G}$ is said to be reachable from a state s_{start} if $\mathcal{P}$ can find a path to it within a (sufficiently large) time bound $\mathcal{T}_{\mathcal{P}}$

Definition 3. Goal Coverage. A reachable goal $g \in \mathcal{G}$ is said to be covered by the CTMP algorithm for a state s_{start} if (in the online phase) it can plan from s_{start} to $\mathcal{G}$ while

satisfying Definition 1.

We are now equipped to define CTMP-Completeness.

Definition 4. CTMP-Completeness. An algorithm is said to be CTMP-complete if it covers all the reachable goals $g \in \mathcal{G}$ for s_{start}.

3.3.2 Straw man Approach

To ensure protection against all possible attacks, we have $\mathcal{G}$ as a discretized set of all projectiles that can intersect the robot body. A projectile ρ is represented as $(\rho_p, \rho_v) \equiv \{p_x, p_y, p_z, v_x, v_y, v_z\}$, that is the position and velocity of the incoming object in $\mathbb{R}^3$. Thus, the dimensionality of $\mathcal{G}$ is six. With this representation of $\mathcal{G}$, firstly defining the bounds of $\mathcal{G}$ is difficult for the given problem setup as we do not want to make strong assumptions about the range of incoming attacks. Second, loosely bounded $\mathcal{G}$ with a fine-enough granularity could be extremely large because of its high dimensionality. Additionally, to only preprocess for the attacks that would hit $\mathcal{B}$, it would require substantial rejection sampling, increasing the preprocessing overhead.

The naive CTMP approach thus computes and stores paths for all projectiles in $\mathcal{G}$ which requires a massive amount of preprocessing time and memory given the size of $\mathcal{G}$, making it infeasible for practical purposes.

3.3.3 Proposed Approach

Our approach differs from the straw man approach in the representation of $\mathcal{G}$. In our approach, we define two *domes* around the robot body, an inner dome D_i and an outer dome D_o. D_i approximates the geometry of the robot and D_o captures the robot's reachable space so that it can intercept the projectiles with the shield $\mathcal{S}$ positioned anywhere in the 3D space between D_i and D_o. The two domes are discretized into cells. Our planning approach is divided into two stages - the preprocessing stage and the query stage. In the preprocessing stage, for each pair of cells (with one cell from D_i and one cell from D_o), we plan a path to a pose of $\mathcal{S}$ that can block all projectiles passing through the pair of cells. These paths are stored in a lookup table mapping the pair of cells to the corresponding path. In the

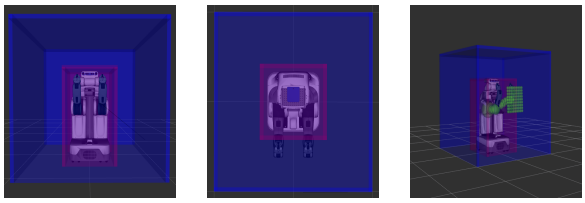
query stage, for an incoming projectile ρ, we first identify the pair of cells through which ρ passes. Second, we look up the corresponding path π from the look up table in constant time. For replanning, additional paths are computed from the states on these paths. This process runs recursively as newly computed paths create new replannable states which also must be processed. To facilitate replanning, besides the pair of cells, the current state of the robot is also appended to the key of the lookup table.

With this approach, the size of the $\mathcal{G}$ becomes equal to the total number of pairs of cells. Note that these cells are computed only with two-dimensional discretization of the domes' surfaces as opposed to six-dimensional discretization of the space of projectiles. This greatly reduces the size of $\mathcal{G}$ compared to the straw man approach.

3.3.4 Proposed Approach Building Blocks

Domes Specification

The geometry of D_i is such that it tightly encapsulates the robot body, or in other words over approximates the geometry of the robot body with a simple shape. With this, we simplify the problem setup by making a more conservative requirement that D_i must be protected instead of the robot body. We define D_i as a cuboid.



(a) Front view (b) Top view (c) Side view

Figure 3.1: (a) and (b) show the inner (red) and outer (blue) domes surrounding the robot. (c) shows the PR2 robot with a shield attached to its arm (in simulation).

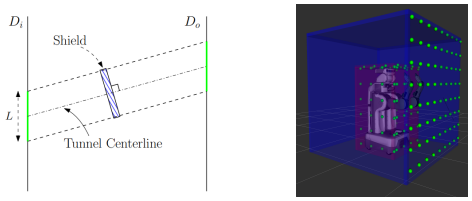
The outer dome D_o captures the reachable workspace of the robot. For simplicity, we also define D_o as a cuboid. We keep D_o co-centric with D_i but it is larger than D_i. A larger D_o would allow more freedom to the robot but would also increase the preprocessing demand. On the other hand, a smaller D_o would restrict the robot and limit its protection capability. We choose the size of D_o such that the robot's manipulator can reach the side it faces at full arm extension. While our approach is simple, a more rigorous reachability analysis could be performed to optimize for the geometry of D_o. Figure 3.2 shows the two

domes configured for the PR2 robot's body. The volume between the D_o and D_i is where the robot manipulates the shield to intercept any incoming projectiles.

Domes Discretization and Shield geometry

Each side of D_i and D_o are discretized into cells. The discretization is correlated with the shape and size of $\mathcal{S}$. We use a square-shaped $\mathcal{S}$ of in our setup. The discretization of the two domes is shown in Figure 3.2 (b).

The protrusion of each pair of cells (with one cell from D_i and one cell from D_o) along a straight line into the volume between D_o and D_i constitutes a *tunnel*. A line segment connecting the centers of the cells forming the tunnel is called *centerline*. Our key idea is that if $\mathcal{S}$ is positioned such that it fully blocks this tunnel, all possible attacks that cross the pair of cells are blocked by it. This idea is illustrated in Figure 3.2 (a).



(a) Tunnel diagram (b) Dome discretization

Figure 3.2: (a) shows a tunnel formed by a pair of cells (shown in green) in D_o and D_i. (b) shows the centers of the cells on both domes. For D_o we only show the discretization for the front side.

The size of the cells is proportional to the size of $\mathcal{S}$. Specifically, we choose the cell size to be smaller than the size of $\mathcal{S}$ to allow some tolerance for the pose of $\mathcal{S}$ that blocks the tunnel. This tolerance is needed for possible planning and execution errors. We performed a thorough geometric analysis of the magnitude of reduction in cell size that is needed to account for these errors. The related derivations and equations have been left out of this thesis for brevity.

We approximate the portion of the projectile that lies between the two domes by a line segment. This approximation is made under the assumption that the objects move in a straight line within that region and therefore do not breach the boundaries of the tunnel which they enter. This is not too strong of an assumption if the distance between D_i and D_o is small compared to the distance from which the attacks are launched. This assumption can be lifted by performing an analysis for the reduction in cell size needed to account for the projectile to line segment approximate error. We leave this analysis for future work.

Goal Condition

The goal $g \in G$ is defined as the centerline of a tunnel. For the motion planner, the goal condition is any pose of $\mathcal{S}$ along the centerline, such that $\mathcal{S}$ is oriented orthogonal to it (see Figure 3.2 (a)). For ease of planning, we allow small tolerance in $SE(3)$ for the goal pose.

In our implementation, we sample equidistant points along the tunnel's centerline and compute $SE(3)$ poses at each point which are orthogonal to it. The motion planner then attempts to plan to each of these poses sequentially, until it succeeds. If it fails to do so, then the corresponding $\mathcal{G}$ is marked as unreachable.

Motion Planner

We use a heuristic search-based planning approach with motion primitives (see, e.g, [20], [19], [31]) as they have strong optimality guarantees. The states and the transitions implicitly define a graph $= (S, E)$ where S is the set of all states and E is the set of all transitions defined by the motion primitives. We use the Anytime Repairing A* algorithm [20] to find a path in from a given state s to a goal pose. ARA* is an anytime heuristic search algorithm that first computes a sub-optimal path to the goal, and then refines that path such until it finds an optimal path, or the time runs out.

3.3.5 Algorithm Details

We are now ready to describe the preprocessing and query stages of our CTMP approach for solving the shield-based protection task.

Preprocessing Stage

In the preprocessing stage, for each pair of cells, the tunnel centerlines are computed. Each tunnel is checked for feasibility. Namely, the tunnels whose volume snaps to zero anywhere along the length are discarded because no incoming object coming through such a tunnel can reach the tunnel's end on D_i. The centerlines of all the feasible tunnels constitute the goal region $\mathcal{G}$ We pick a constant number of equidistant goals on the line segment to plan to for the manipulator ($computeTargetPoses$ method on line 4 in Algorithm 4).

First, our algorithm computes the paths from s_{home} to cover $\mathcal{G}$ by satisfying the goal criteria described above. These paths are then time-profiled and then discretized evenly in

time to get a set of *replannable* states. The algorithm then recursively computes additional paths to cover the goal regions from these replannable states.

Even though our dome-based formulation makes the number of goals in $\mathcal{G}$ tractable such that they can be computed using the heuristic-search based planner, computing the replanning trajectories in the same manner is expensive. The reason for this is because a new perception update could arrive at any arbitrary time when the manipulator is executing the previous trajectory. As a result, to obtain the set of replanning trajectories, we compute the linear interpolation from each replannable state to all goals in $\mathcal{G}$ (lines 4 and 5 in Algorithm 5). During the offline phase, we also collision-check the resulting trajectory with the rest of the robot body. As this linear interpolation is an inexpensive operation (time for computing the interpolation is in the order of a few milliseconds), we do not store the resulting trajectory. Instead, we maintain a 3D tensor, where the first axis represent the current goal prescribed by the perception stack, the second axis represents the new goal based on the perception update, and the third axis represents the index of the wapoint on the current trajectory. If a path exists from the current waypoing to the new goal, that index is marked as *True*, else it is marked as *False*. In cases when the interpolation fails (could be because of a self-collision or the manipulator hitting the joint limits), we invoke a motion planner that attempts to compute the required trajectory. If successful, the trajectory is stored in the database.

The outcome of the preprocessing stage is a lookup table.

$$\mathcal{M} : S \times G \rightarrow \{\pi_1, \pi_2, ...\}$$

Query Stage

In the query stage, for a given query $g \in G$, in the first step, the corresponding centerline is identified. This is done by first computing the points of intersection of the projectile on D_o and D_i and then finding the the cells within which the points lie. In the second step, the associated path is retrieved from $\mathcal{M}$.

Specifically, for the initial estimate received by the planning module, we query $\mathcal{M}$ to obtain a manipulator trajectory from s_{home} to g_1. For all the remaining estimates, we use the tensor to check if a transition from g_i to g_j is possible. If its is, then the corresponding trajectory is executed by the manipulator.

Algorithm 4 - Generate trajectory database

Inputs: Set of goals $\mathcal{G}$, Planner $\mathcal{P}$, Home configuration s_{home}
function ComputeBaseTrajectories($\mathcal{G}, \mathcal{P}, s_{home}$)
    for g_i in $\mathcal{G}$ **do**
        $\mathcal{T}_i \leftarrow ComputeTargetPoses(l_i)$
        $TrajBuffer$ = []
        for t in $\mathcal{T}_i$ **do**
            if $\mathcal{P}(s_{home}, t)$ exists **then**
                Add resulting plan to $TrajBuffer$.
            else
                Goto next target pose in $\mathcal{T}_i$
            end if
        end for
        Add the least-time trajectory in $TrajBuffer$ to $\mathcal{M}$.
    end for

Algorithm 5 - Generate replanning database

Inputs: Set of goals $\mathcal{G}$, Planner $\mathcal{P}$, Trajectory Database $\mathcal{M}$ containing initial plans
function ComputeReplanTrajectories($\mathcal{G}, \mathcal{P}, s_{home}$)
    for t_i in $\mathcal{M}$ **do**
        for wp_j in t_i **do**
            for g_i in $\mathcal{G}$ **do**
                if $\mathcal{LI}(wp_j, l_k)$ is valid **then**
                    Add resulting linearly interpolated trajectory to $\mathcal{M}$
                else
                    if $\mathcal{P}(wp_j, l_k)$ exists **then**
                        Add resulting trajectory to $\mathcal{M}$

3.3.6 Theoretical Analysis

Lemma 1(Completeness): Given a start state s, and the configuration space of the manipulator $\mathcal{X}$, our algorithm is guaranteed to find a path, if one exists, from any intermediate configuration to a goal $g \in \mathcal{G}$.

Proof(Sketch): During the first stage of the preprocessing phase (Algorithm 4), we compute and store trajectories to all feasible goal states. Once the robot starts executing a particular trajectory, a perception update can require the robot to execute a different trajectory at any instant. To support this, during the second preprocessing phase (Algorithm 5), we recompute linear interpolation-based trajectories to all feasible goal states. If the linear interpolation failes, we attempt to reach that goal via a motion planner, hence ensuring that all feasible goals are reached. Hence, our algorithm is CTMP-Complete.

3.4 Perception Module

Predicting the trajectory of the incoming projectile is crucial for a successful intercept. This problem introduces three challenges:

1. A projectile moving through the air experiences drag force, and hence we need to determine the drag parameters to accurately model the projectile motion for prediction.
2. To have a good coverage and detection accuracy, multiple sensors might be placed in the world relative to the robot. These sensors would require accurate calibration to determine the extrinsic rigid body transform between the detection frame and the body frame of the robot.
3. The perception system should update the projectile estimate as it obtains more information about the projectile during its flight to improve the accuracy of the prediction.

We solve the first two challenges with detection-based calibration (3.4.2), and solve the third challenge by performing a least-squares model fitting based on all observations (3.4.1).

3.4.1 Projectile Estimation

The detection of the projectile's position is a fairly straightforward task, wherein the point cloud corresponding to the projectile is filtered out and the centroid of the filtered cloud is

assumed to be its position.

To predict the motion of the projectile, we fit multiple position detections at various time intervals to a projectile equation of motion. The motion of the incoming projectile is assumed to be on the $x - z$ plane (with z pointing vertically upwards). Movements perpendicular to this plane which can occur due to wind effects or other external forces are ignored. We utilize the air drag based projectile equation of motion to model the motion of the projectile along each axis.

$$p(t) = p_{\text{init}} + \frac{v_{\text{init}} TV}{g}\left(1 - e^{-gt/TV}\right) \tag{3.2}$$

Equation 3.2 represents the model of the projectile's motion along the x-axis, where, p_{init}, v_{init} represent the x-component of the initial position and velocity of the projectile, TV, the terminal velocity, and g, the gravitational acceleration. We assume the terminal velocity of the projectile is known (which can be computed from the object's mass). Given, that we are able to accurately detect the position of the projectile, p_{init} can be identified as well, leaving v_{init} to be the only unknown. From Equation 3.2 it is easy to see that given the initial position detection (p_{init}) and a detection at a later time $p(t)$, v_{init} can be computed and hence the motion of the projectile. To reduce the impact of noise, an average over multiple detections are taken to estimate v_{init}. A similar computation is done to predict the z-component of the projectile's motion.

3.4.2 Detection-based calibration

The goal of this module is to compute the transformation between the external sensors say i and the body-frame F_B of the robot denoted by $T_{F_i}^{F_B}$. Each sensor provides us a projectile estimate at time t, denoted by $\hat{\rho}_{p,i}(t)$ in its own frame. $\hat{\rho}_{p,i}(t)$, $\hat{\rho}_{p,s}(t)$ are the 3-D position vectors of the projectile as seen by external sensor i and on-board sensor s, respectively in their homogeneous coordinates. We solve for an optimization function pair-wise for each sensor i.

$$\min_{T_{F_i}^{F_B}} \sum_{t=0}^{t_h} \|T_{F_i}^{F_B} \hat{\rho}_{p,i}(t) - \hat{\rho}_{p,b}(t)\|^2 \tag{3.3}$$

We assume the mass of the projectile and solve for the above optimization using

Simplicial Homology Global Optimization (SHGO) [7]. We used a zero-order optimization as we found it to be numerically stable for the air-drag based equations of motion. The global-optimization helps us search for the global minimum.

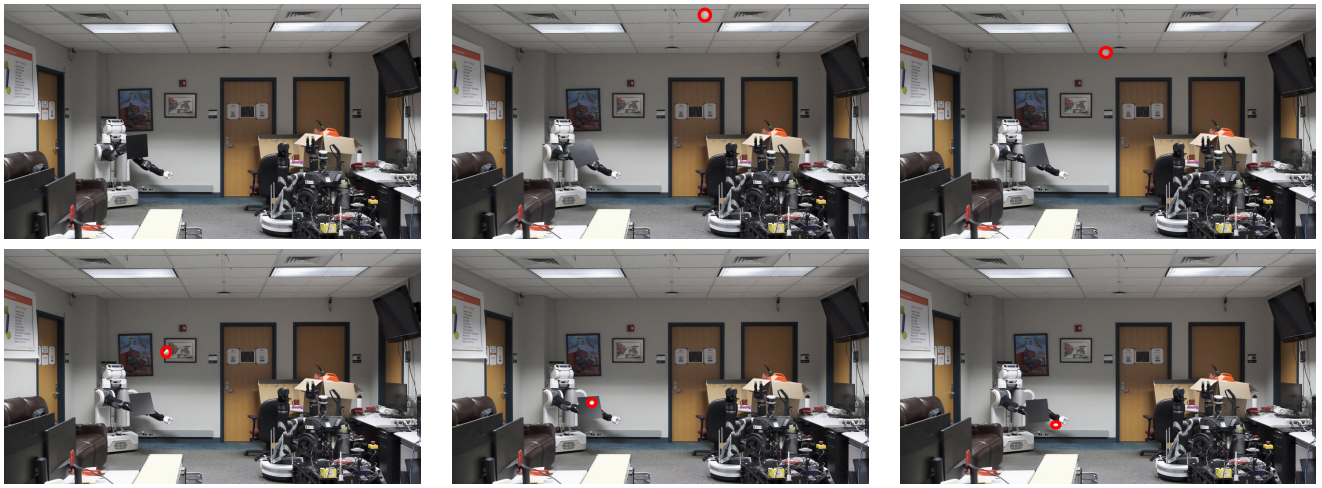


Figure 3.3: Snapshots from a video showing PR2 robot deflecting a white ball thrown at it (sequenced top left to bottom right).

3.5 Experiments and Results

In order to test our framework, we performed several experiments, both in simulation and in the real world (example of a real world run shown in Figure 3.3). In simulation, we mimiced the perception module by randomly sampling projectiles from all directions around the robot. This allowed us to simulate projectiles that might be hard to test in the real world given our environmental constraints. In this section, we will first go over the experiments carried out to analyze the performance of the perception module. Next, we will discuss the experiments performed to test the system in simulation, and in the real world.

3.5.1 Perception Module

In this experiment, we conduct 60 throws on the real-robot and compute the accuracy of the projectile estimate based on four methods 3.1. We use eucledian distance between the location on the inner plane where the projectile hit and the predicted location on the inner

plane. The first method makes a prediction based on 5 detections of the projectile from an external camera which is placed 8m away from the robot. We also measure the percentage of throws in this the error was higher than 25cm. With the discretization of intercept locations and the dimensions of the shield, a perception error of 25cm would cause interception to fail. We observe, that with just one addition projectile detection from an onboard camera decreases the prediction error. We see a significant decrease in potential failures as well. Since the onboard camera is working at 30fps, waiting for an additional detection takes 33ms. Thus, there is a tradeoff between the accuracy and time delay between projectile estimate updates. We used only the first additional detection from the oboard camera.

Table 3.1: Prediction accuracy as number of detections increase

Methods	**Prediction Error (cm)**	**Prediction Error >25 cm (%)**
Only external camera (EC)	15.63	14.28
EC + 1 detection from onboard camera	9.62	0
EC + 2 detections from onboard camera	7.33	0
EC + 3 detection from onboard camera	6.37	0

3.5.2 Motion Planning Module

The PR2 robot is surrounded by 2 domes, the inner dome and the outer dome. The outer dome has a length = 0.75 m, breadth = 0.75 m, and a height of 1.6 m. The inner dome has a length = 0.4 m, breadth = 0.4 m, and a height of 1.1 m. In total, there are 15,072 tunnels that connect a cell from the outer dome to a cell on the inner dome. The planner was able to compute valid trajectories for 4,078 tunnels. These trajectories were generated by setting a time-out of 0.5 seconds for the motion planner. For our experiments, we uses the ARA* planner, with the starting sub-optimality factor of 100, and the final sub-optimality factor of

Table 3.2: Evaluating the planning module in simulation

Outer dome planes	**Our approach**	**IK-based approach**
Front plane	575	533
Right plane	358	344

1. The planner was able to converge to a sub-optimality factor of 1 for most trajectories. These trajectories were generated by inflating the default velocity and acceleration limits by a factor of 3 in order to operate the robot as fast as possible within the physical constraints.

3.5.3 Evaluation in simulation

We evaluate the motion planning module first in simulation. We sample projectiles that intersect the all planes surrounding the robot. The launch distance for each projectile was uniformly sampled from a range of 8 meters to 12 meters. The time between the estimate obtained for the first projectile and the second projectile was sampled from a uniform distribution in the range of 0.25 to 0.55 seconds. The points on both the inner and the outer domes were also randomly picked.

In order to evaluate our system, we sampled 1000 random projectiles for each plane of the outer dome. The back, left and the top planes had very few valid initial trajectories to begin with, and hence the number of projectiles that entered via those planes and were saved was also negligible.

We compare the performance of our planning module with one baseline approach. This baseline has access only to the linear-interpolation based trajectories for both the initial trajectories (used when the first perception estimate arrives) as well as during replanning. We use this baseline as this takes negligible time for preprocessing. Table 3.2 shows the result of throwing 1000 random projectiles at the robot from the front and the right planes. We see that our method does better than the baseline, mainly because of collision and joint limit constraints that invalidate the linear-interpolation operation for certain plans.

3.5.4 Experiments on Real Robot

We tested the full system with integrated perception on the PR2 robot and an external camera in a lab environment. We set up the range of attacks to be 8–10m. That roughly

translates to a flight time of one second. Out of which, on average, 200ms is used by the perception system, 700ms is the trajectory execution time for the robot and 50ms is taken up by other system overheads. That leaves only 50ms for the CTMP planner on average. Our CTMP planner successfully generates a plan within this time budget.

We performed 50 throws with replanning enabled, and 50 throws with replanning disabled. We obtained a success rate of 70% when replanning was enabled, and 60% with replanning disabled. Out of the 35 successful throws in which replanning was enabled, replanning was invoked 13 times. The reason for this is that the execution times for certain trajectories is lower than the time taken for the perception module to provide a new estimate. Out of the 15 cases in which the replanning module failed, replanning kicked in for 6 cases. The major reason these cases still fail is either that the perception error even after the second update was high, or that the execution of the full trajectory took more time than that of the time of flight of the ball.

3.6 Conclusion

In this chapter, we have presented a planning and perception framework for intercepting incoming projectiles. We developed a Constant-Time Motion Planning based algorithm that incorporates updated perception estimates for performing replanning. In addition, we also demonstrated the use of multiple RGB-D cameras to obtain continuous state estimate of the incoming projectile. We deployed our framework in simulation as well as in the real world, and discussed our results. In the future, we would like to extend this work to a mobile manipulator, where the robot can choose to move its base, or to use its arm to intercept projectiles. In addition, it would be useful to develop more reactive Constant-Time Motion Planning algorithms that can be used for such dynamic tasks.

Chapter 4

Conclusion and Future Work

In this thesis, we investigated two ways in which preprocessing-based methods can be used to successfully tackle challenging robotic manipulation tasks. The preprocessing step has various advantages, as well as several disadvantages, which must be kept in mind while choosing the algorithm for a given task.

In the analysis of the proposed algorithms and the experimental evaluations, we saw that preprocessing-based methods can be extremely beneficial when robotic manipulators are performing repetitive tasks in known environments. As discussed in the first part of this thesis, for solving constrained manipulation tasks, such as opening a closed container, a useful set of macro-actions can be learnt that can greatly speed up the planning problem. These macro-actions can be useful for solving not just a selected few planning problems, but a distribution of planning problems. Similarly, in the second part of the work, we saw that a set of trajectories can be stored in the trajectory database, which can be used to intercept a range of incoming projectiles.

Preprocessing-based methods provide a way to store actions and trajectories that might be too expensive to compute online. This advantage of the preprocessing-based methods can be clearly seen in the projectile interception task. If the robot does not have access to preprocessed-trajectories, and it needs to move its arm in a direction in which any one of its joint exceeds the joint limits while moving in the direction of the goal, the robot will eventually fail to complete the task. The main cause of this failure is because planning in the configuration space of the robot can be computationally expensive, and could potentially not finish within the allocated time budget. Hence, preprocessing-based methods provide

access to high-quality trajectories to the robot, which can help the robot complete the task within the given constraints.

However, there are also some important disadvantages when it comes to employing such algorithmic techniques for motion planning. In tasks that are not very constrained, the overhead of preprocessing can be a wasted effort. For example, if the robot is tasked with performing a simple task in which the constraints can be easily met without invoking an expensive call to the motion planner, it might be redundant to spend resources on preprocessing.

This body of work opens up interesting directions for future research. One common assumption that was made in both parts of this thesis is that the robot is operating in a static environment. When we use this assumption, the data that was gathered during the preprocessing phase can be used as-is. When we relax this assumption, very interesting algorithmic challenges emerge as to what is the right kind of information that is to be learnt in order to help the robot successfully complete its task.

Another assumption that was made in both algorithms presented in this thesis is that the robot arm is set on a stationary base. This assumption should also be relaxed, as many robotic platforms have a moving base, which can be used to navigate and potentially solve the task more efficiently. Adding a navigation component would make the planning problem exponentially more complicated, as it increases the degrees of freedom of the robot, as well as adds additional kinematic and dynamic constraints that the robot has to adhere to. When planning in such high-dimensional spaces, and with additional constraints in place, the preprocessing step would have to be significantly optimized in order to be able to tractably gather the data that would be required online.

Bibliography

[1] Aditya Agarwal, Yupeng Han, and Maxim Likhachev. Perch 2.0: Fast and accurate gpu-based perception via search for object pose estimation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10633–10640. IEEE, 2020. 3.1

[2] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24:581–621, 2005. 2.1, 2.5

[3] Dieter Büchler, Simon Guist, Roberto Calandra, Vincent Berenz, Bernhard Schölkopf, and Jan Peters. Learning to play table tennis from scratch using muscular robots. *arXiv preprint arXiv:2006.05935*, 2020. 3.1

[4] Berthold Bäuml, Thomas Wimböck, and Gerd Hirzinger. Kinematically optimal catching a flying ball with a hand-arm-system. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2592–2599, 2010. doi: 10.1109/IROS.2010.5651175. 3.1

[5] Alexandra Carpentier and Andrea Locatelli. Tight (lower) bounds for the fixed budget best arm identification bandit problem. In *Conference on Learning Theory*, pages 590–604, 2016. 2.4.1

[6] Andrew I Coles and Amanda J Smith. Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research*, 28:119–156, 2007. 2.1, 2.2.1, 2.5

[7] Stefan C Endres, Carl Sandrock, and Walter W Focke. A simplicial homology algorithm for lipschitz optimisation. *Journal of Global Optimization*, 72(2):181–217, 2018. 3.4.2

[8] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Pac bounds for multi-armed bandit and markov decision processes. In *International Conference on Computational Learning Theory*, pages 255–270. Springer, 2002. 2.4, 2.4.1, 2.5.1

[9] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of machine learning research*, 7(Jun):1079–1105, 2006. 2.4.1

[10] Roland Geraerts and Mark H Overmars. A comparative study of probabilistic roadmap planners. In *Algorithmic foundations of robotics V*, pages 43–57. Springer, 2004. 3.1

[11] Glenn A Iba. A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3(4):285–317, 1989. 2.1

[12] Fahad Islam, Oren Salzman, Aditya Agarwal, and Maxim Likhachev. Provably constant-time planning and replanning for real-time grasping objects off a conveyor belt. *The International Journal of Robotics Research*, 0(0):02783649211027194, 0. doi: 10.1177/02783649211027194. URL https://doi.org/10.1177/02783649211027194. 1, 3.3, 3.3.1

[13] Fahad Islam, Oren Salzman, Aditya Agarwal, and Maxim Likhachev. Provably constant-time planning and replanning for real-time grasping objects off a conveyor belt. *The International Journal of Robotics Research*, 40(12-14):1370–1384, 2021. 3.1

[14] Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On the complexity of best-arm identification in multi-armed bandit models. *The Journal of Machine Learning Research*, 17(1):1–42, 2016. 2.4.1

[15] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996. 3.1

[16] Gayane Kazhoyan, Simon Stelter, Franklin Kenghagho Kenfack, Sebastian Koralewski, and Michael Beetz. The robot household marathon experiment. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021. URL https://arxiv.org/abs/2011.09792. Accepted for publication. 1

[17] Richard E Korf. Macro-operators: A weak method for learning. *Artificial intelligence*, 26(1):35–77, 1985. 2.1

[18] Ryan A Kromer, Antonio Abellán, D Jean Hutchinson, Matt Lato, Tom Edwards, and Michel Jaboyedoff. A 4d filtering and calibration technique for small-scale point cloud change detection with a terrestrial laser scanner. *Remote Sensing*, 7(10):13029–13052, 2015. 3.1

[19] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8): 933–945, 2009. 3.3.4

[20] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. *Advances in neural information processing systems*, 16, 2003. 3.3.4

[21] Shie Mannor and John N Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research*, 5(Jun):623–648,

2004. 2.4.1

[22] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013. 3.1

[23] Muhammad Abdul Hakim Newton, John Levine, Maria Fox, and Derek Long. Learning macro-actions for arbitrary planners and domains. In *ICAPS*, volume 2007, pages 256–263, 2007. 2.1, 2.2.1

[24] Korbinian Nottensteiner, Arne Sachtler, and Alin Albu-Schäffer. Towards autonomous robotic assembly: Using combined visual and tactile sensing for adaptive task execution. *Journal of Intelligent & Robotic Systems*, 101(3):1–22, 2021. 1

[25] Evangelos Papadopoulos, Farhad Aghili, Ou Ma, and Roberto Lampariello. Robotic manipulation and capture in space: A survey. *Frontiers in Robotics and AI*, 8, 2021. ISSN 2296-9144. doi: 10.3389/frobt.2021.686723. URL https://www.frontiersin.org/article/10.3389/frobt.2021.686723. 1

[26] Kirstin H. Petersen, Nils Napp, Robert Stuart-Smith, Daniela Rus, and Mirko Kovac. A review of collective robotic construction. *Science Robotics*, 4(28):eaau8479, 2019. doi: 10.1126/scirobotics.aau8479. URL https://www.science.org/doi/abs/10.1126/scirobotics.aau8479. 1

[27] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *2017 ieee international conference on robotics and automation (icra)*, pages 1527–1533. IEEE, 2017. 1

[28] Mike Phillips, Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *International Symposium on Combinatorial Search*, volume 3, 2012. 3.1

[29] Mike Phillips, Victor Hwang, Sachin Chitta, and Maxim Likhachev. Learning to plan for constrained manipulation from demonstrations. *Autonomous Robots*, 40(1): 109–124, 2016. 2.5

[30] Mihail Pivtoraiko and Alonzo Kelly. Efficient constrained path planning via search in state lattices. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, pages 1–7, 2005. 2.2.1

[31] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1 (3-4):193–204, 1970. 2.3.2, 3.3.4

[32] Jan Quenzel, Matthias Nieuwenhuisen, David Droeschel, Marius Beul, Sebastian Houben, and Sven Behnke. Autonomous mav-based indoor chimney inspection with 3d laser localization and textured surface reconstruction. *Journal of Intelligent & Robotic Systems*, 93(1):317–335, 2019. 1