

# Preprocessing-based Kinodynamic Motion Planning Framework for Intercepting Projectiles using a Robot Manipulator

Ramkumar Natarajan<sup>1\*</sup>, Hanlan Yang<sup>1\*</sup>, Qintong Xie<sup>2</sup>, Yash Oza<sup>1</sup>, Manash Pratim Das<sup>1</sup>,  
Fahad Islam<sup>1</sup>, Muhammad Suhail Saleem<sup>1</sup>, Howie Choset<sup>1</sup>, and Maxim Likhachev<sup>1</sup>

**Abstract**—We are interested in studying sports with robots and starting with the problem of intercepting a projectile moving toward a robot manipulator equipped with a shield. To successfully perform this task, the robot needs to (i) detect the incoming projectile, (ii) predict the projectile’s future motion, (iii) plan a minimum-time rapid trajectory that can evade obstacles and intercept the projectile, and (iv) execute the planned trajectory. These four steps must be performed under the manipulator’s dynamic limits and extreme time constraints ( $\leq 350\text{ms}$  in our setting) to successfully intercept the projectile. In addition, we want these trajectories to be smooth to reduce the robot’s joint torques and the impulse on the platform on which it is mounted. To this end, we propose a kinodynamic motion planning framework that preprocesses smooth trajectories offline to allow real-time collision-free executions online. We present an end-to-end pipeline along with our planning framework, including perception, prediction, and execution modules. We evaluate our framework experimentally in simulation and show that it has a higher blocking success rate than the baselines. Further, we deploy our pipeline on a robotic system comprising an industrial arm (ABB IRB-1600) and an onboard stereo camera (ZED 2i), which achieves a 78% success rate in projectile interceptions.

## I. INTRODUCTION

In this paper, we study the problem of intercepting projectiles using robot manipulators. This is a challenge that finds unique and exciting applications in various sports. Recent breakthroughs in this field include instances where a robot can handle various types of spin in table tennis matches against human opponents [1], and instances where robots have skillfully intercepted flying, rolling, and bouncing balls [2]. These scenarios represent the exciting intersection of sports and robotics. In our problem, our primary goal is to intercept incoming projectiles with robot manipulators in real-time with onboard processing. We aim to demonstrate how robotic systems, equipped with kinodynamic planning capabilities, can plan smooth trajectories while minimizing torque requirements.

The time-critical nature of the task calls for an intelligent framework that can maximize the robot’s 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 350 milliseconds from the time the projectile is first detected until it hits the body of the robot. Within this duration, our framework should be

capable of performing three major tasks: (i) detecting and accurately predicting the motion of the incoming projectile, (ii) querying a motion planner for a trajectory that would enable the robot to safely intercept the projectile, and (iii) executing the trajectory returned by the motion planner. The fast nature of the task, combined with the physical limitations of the robot (typical executions of blocking maneuvers consume about 250 of the 350 milliseconds available to us), calls for a real-time, yet optimal, planning framework. In this paper, we present our pipeline that comprises (i) an on-board stereo camera-based perception module that provides estimates of the incoming projectile and its predicted trajectory and (ii) a preprocessing-based planning framework that guarantees to return a blocking trajectory if it exists (based on the perception estimates) within a significantly small time budget (1ms). The contributions of our work can be summarized as follows:

- The formulation of the problem of robot protection against incoming projectiles and the development of the framework consisting of real-time perception, motion planning, and execution.
- A preprocessing-based motion planning module that guarantees to return blocking trajectories within an extremely small time window.
- 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 an ABB’s IRB-1600 robot arm and a ZED 2i stereo camera setup, as shown in Fig. 1.

## II. RELATED WORK

Several works in the literature have investigated tasks that are similar to the task of intercepting projectiles. In this section, we will discuss these works as well as their advantages and disadvantages. In [3], the authors presented a framework that robot manipulators can use to catch a ball. Specifically, the catching problem is formulated as a bilevel optimization problem, which produces a joint trajectory that can successfully complete the catching task. The advantage of using an optimization-based trajectory generation algorithm is that trajectories can be quickly generated in simple environments, even for high-dimensional planning problems. There are three key disadvantages in this work compared to our work: (i) as [3] solves an optimization problem online while the projectile is in flight, it is not guaranteed to return a solution within a strict time limit; (ii) [3] assumes that

<sup>1</sup>The Robotics Institute, School of Computer Science, Carnegie Mellon University, USA. {rnataraj, hanlany, yoza, mpratimd, fi, choset, mlikhach}@andrew.cmu.edu

<sup>2</sup>Department of Engineering Science, University of Oxford, UK. qintong.xie@univ.ox.ac.uk

\*Contributed Equally

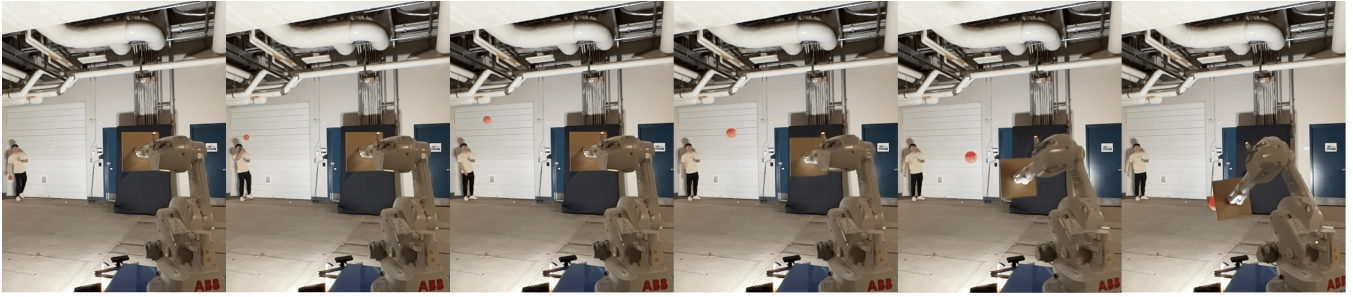


Fig. 1. ABB arm with a shield attached to its end-effector blocking a ball thrown at it.

there are no obstacles in the workspace; (iii) [3] uses a VICON system that provides a near ground-truth pose of the incoming object. Hence, the assumptions of the planning and perception problems in [3] are stronger than those of our framework as our pipeline avoids obstacles, performs object pose estimation using RGB-D sensors, and does not require a motion capture system. [4] and [5] use reinforcement learning in a model-free setting to learn policies that are capable of generating dynamic strokes for the table tennis robot. The main advantage of using machine learning-based methods is that the policies learned by the neural network can generalize to new data points that are similar to previously seen data points. However, these approaches require training a large set of data, especially if the scenes are allowed to have obstacles [6].

On the perception side, RGB-D cameras are very noisy and thus require special filtering techniques to detect incoming projectiles. Model-specific methods, such as [7] attempt 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. There also exist more generic approaches like [8] that do not have such requirements. But these methods require either a laser scanner or a VICON system [9] to detect the moving object. Recently, machine learning-based methods have also gained popularity for object detection [10]. For example, YOLO [11] can be utilized for object detection and tracking. However, it cannot detect the fast-moving ball at a distance. Therefore, this method requires substantial data for training, making it challenging to predict their performance on new data points. In our work, we employ a straightforward color detection algorithm to detect the projectile using a single stereo camera. This approach improves the practicality of our framework in real-world scenarios.

### III. PROBLEM FORMULATION

Consider a robot manipulator with a shield rigidly attached to the end-effector of the manipulator. The manipulator is tasked with protecting a specific object. A projectile is launched in the direction of the object to be protected, and the manipulator must intercept the projectile before it collides with that object. The state of the projectile,  $\rho$ , is represented as a tuple  $(\rho_p, \rho_v)$ , 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 object which is to be protected  $\mathcal{O}$ . To solve

this problem, we make certain simplifying assumptions:

- The manipulator always starts from a “home” configuration  $s_{home}$ .
- The projectile is launched from within the field of view of the camera, allowing us to get early estimates of its trajectory  $\rho$ .
- At any point in time, only a single projectile is launched in the direction of  $\mathcal{O}$ .

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  $\mathcal{O}$ . Let  $\mathcal{T}_d$  be the time duration for the perception module to collect enough frames and finish estimating the trajectory. Finally, let  $\mathcal{T}_p$  be the time taken to query the motion planner for a manipulator trajectory, and  $\mathcal{T}_e$  be the time taken by the manipulator to execute that trajectory.

In order for the manipulator to successfully intercept the projectile, 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:

$$\mathcal{T}_t \geq \mathcal{T}_d + \mathcal{T}_p + \mathcal{T}_e \quad (1)$$

## IV. METHOD

### A. System Overview

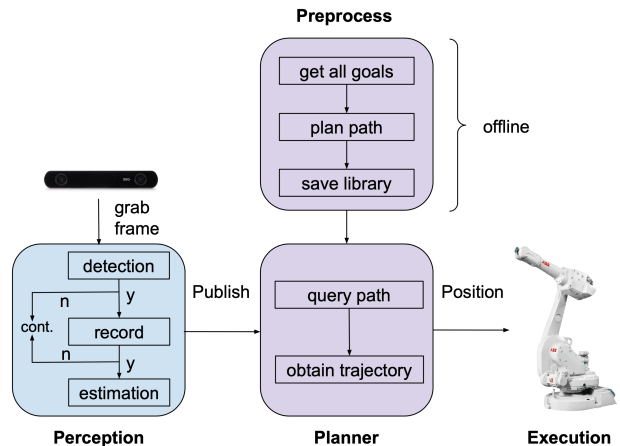


Fig. 2. Flowchart showing the overall pipeline.

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. The extreme time constraints limit the use of online real-time planning methods [12]. To this end, we propose an approach that relies heavily on precomputation, similar to the Constant-Time Motion Planning (CTMP) class of algorithms [13].

At a high level, each frame captured by the camera is processed within the perception module shown in Figure 2. Once a sufficient number of frames are detected, those frames and their timestamps are used to estimate the trajectory of the projectile. The derived positions and velocities of the projectile are then published to the planner. Subsequently, by querying a motion library computed offline from solutions to a series of planning problems, we obtain a trajectory for the manipulator to block the projectile. The trajectory is then executed to intercept the projectile with the shield held by the manipulator and prevent any potential collisions with the object  $\mathcal{O}$ . In the remainder of the section, we explain the proposed approach and its building blocks in detail and describe the whole algorithm.

### B. Proposed Approach

In our approach, we define two cuboidal *domes* around the object  $\mathcal{O}$ , an inner dome  $D_i$  and an outer dome  $D_o$ .  $D_i$  approximates the geometry of the object  $\mathcal{O}$  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  $\rho$ , we first identify the pair of cells through which  $\rho$  passes. Second, we look up the corresponding path  $\pi$  from the look-up table in constant time. With this approach, the size of the goal region  $\mathcal{G}$  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  $\mathcal{G}$ .

### C. The Building Blocks

1) *Dome Specifications*: The geometry of  $D_i$  is such that it tightly encapsulates the object  $\mathcal{O}$ . In other words, it overestimates the geometry of  $\mathcal{O}$  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  $\mathcal{O}$ . We define  $D_i$  as a cuboid. The outer dome  $D_o$  captures the reachable workspace of the robot. For simplicity, we also define  $D_o$  as a cuboid and place it concentric with  $D_i$ . A larger  $D_o$  would allow more freedom for the robot, but would also increase the demand on preprocessing. 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 manipulator can reach the side it faces at full extension. Although our approach is simple, it can easily be generalized to different robot models. Fig. 3 shows the two domes configured for the ABB robot arm. The volume between  $D_o$  and  $D_i$  is where the robot manipulates the shield to intercept the incoming projectiles.

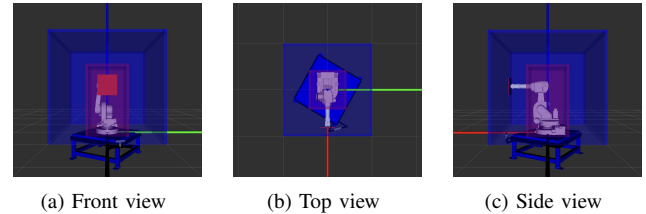


Fig. 3. (a) and (b) show the inner (red) and outer (blue) domes surrounding the robot. (c) shows the ABB's robot arm with a shield attached to its end-effector (in simulation).

2) *Dome Discretization and Shield Geometry*: Each side of  $D_i$  and  $D_o$  is discretized into cells. The discretization is correlated with the shape and size of  $\mathcal{S}$ . We use a square-shaped  $\mathcal{S}$  in our setup. The discretization of the two domes is shown in Figure 4 (a). It should also be noted that the size of the shield in our application can be made significantly larger to decrease the preprocessing effort. But doing this is not always practical and could violate the task and environment constraints. The volume around the straight line connecting a cell from  $D_i$  to a cell from  $D_o$  constitutes a *tunnel*. A line segment connecting the centers of the cells that form 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 cell pair are blocked by it. This idea is illustrated in Figure 4 (b).

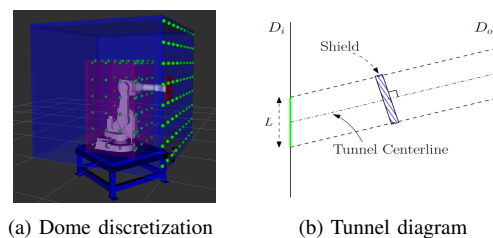


Fig. 4. (a) shows the centers of the cells on both domes. For  $D_o$  we only show the discretization for the front side. (b) shows a tunnel formed by a pair of cells (shown in green) in  $D_o$  and  $D_i$ .

The size of the cell 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 in the pose of  $\mathcal{S}$  that blocks the tunnel. This tolerance is also needed for possible planning and execution errors. We performed a thorough geometric analysis of the magnitude of the reduction in cell size that is needed to account for these errors. These analyses have been omitted 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

a strong assumption if the distance between  $D_i$  and  $D_o$  is small compared to the distance from which attacks are launched. This assumption can be further relaxed by reducing the cell size needed to account for the projectile shape to line-segment approximation error. We leave this analysis for future work.

---

### Algorithm 1 Generate Trajectory Database

---

```

1: Inputs: Motion planner  $\mathbf{P}$ , Home configuration  $\mathbf{x}^S$ , Dome configuration  $\mathbf{C}$ 
2: Output: Trajectory database  $\mathbf{D}$ 
3: procedure PREPROCESS( $\mathbf{P}$ ,  $\mathbf{x}^S$ ,  $\mathbf{C}$ )
4:    $\mathbf{L} \leftarrow \text{COMPUTELINESEGMENTS}(\mathbf{C})$ 
5:   for  $l_i$  in  $\mathbf{L}$  do
6:      $\mathbf{T}_i \leftarrow \text{COMPUTETARGETPOSES}(l_i)$ 
7:      $\text{TrajBuffer} = []$ 
8:     for  $t$  in  $\mathbf{T}_i$  do
9:       if  $\text{IK}(\mathbf{x}^S, t)$  succeeds then
10:        if  $\mathbf{P}(\mathbf{x}^S, t)$  succeeds then
11:          Add resulting plan to  $\text{TrajBuffer}$ .
12:    Add least-time trajectory in  $\text{TrajBuffer}$  to  $\mathbf{D}$ .
```

---

3) *Goal Condition:* The goal  $g \in \mathcal{G}$  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 4 (b)). For ease of planning, we allow a 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.

4) *Motion Planner:* We use a hybrid kinodynamic motion planner called INSAT (INterleaved Search and Trajectory optimization) that combines the benefits of heuristic search and trajectory optimization. We use a heuristic search-based planning approach with motion primitives (see, e.g., [14], [15], [16]) because they have strong theoretical properties. To keep motion planning efficient and tractable, INSAT [17], [18], [19] interleaves heuristic search in the low-dimensional (low-D) subspace with trajectory optimization in the full-dimensional (full-D) planning space. In our setup, the low-D subspace is the configuration space of the arm. Thus, the states and the transitions in low-D implicitly define a graph  $G = (S, E)$  where  $S$  is the set of all states and  $E$  is the set of all transitions defined by the motion primitives.

#### D. Offline Preprocessing and Online Querying

1) *Preprocessing Stage:* In the preprocessing stage, for each pair of cells, the tunnel centerlines are computed. Each tunnel is checked for feasibility and 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 for the manipulator (COMPUTETARGETPOSES method on line 6 in Algorithm 1). Our algorithm computes the paths from  $\mathbf{x}^S$  to cover  $\mathcal{G}$  by satisfying the goal criteria described above.

2) *Query Stage:* In the query stage, the corresponding centerline is identified for a given query  $g \in \mathcal{G}$ . This process involves first computing the points of intersection of the projectile with  $D_o$  and  $D_i$  and then identifying the cells containing these points. Subsequently, the associated path is retrieved from the lookup table constructed in the previous step.

---

### Algorithm 2 INSAT

---

```

1: procedure MAIN( $\mathbf{x}^S, \mathbf{x}^G$ )
2:    $\mathbf{x}_L^{\text{next}} = \mathbf{x}^S$ 
3:   while  $\phi_{\mathbf{x}^S, \mathbf{x}^G}$  is EMPTY do
4:     Pick the next node  $\mathbf{x}_L^{\text{next}}$  to expand  $\triangleright$  Low-D heuristic search
5:     Generate the successors  $\mathbf{X}_L^{\text{new}}$  of  $\mathbf{x}_L^{\text{next}}$   $\triangleright$  Low-D heuristic search
6:     for  $\mathbf{x}_L^{\text{new}}$  in  $\mathbf{X}_L^{\text{new}}$  do
7:       Get the ancestors  $\mathbf{X}_L^{\text{pred}}$  of  $\mathbf{x}_L^{\text{new}}$ 
8:       for  $\mathbf{x}_L^{\text{pred}}$  in  $\mathbf{X}_L^{\text{pred}}$  do
9:          $\phi_{\mathbf{x}_L^{\text{pred}}, \mathbf{x}_L^{\text{new}}} = \text{TRAJOPT}(\mathbf{x}_L^{\text{pred}}, \mathbf{x}_L^{\text{new}})$ 
10:        if isValid( $\phi_{\mathbf{x}_L^{\text{pred}}, \mathbf{x}_L^{\text{new}}}$ ) then
11:           $\phi_{\mathbf{x}_L^{\text{pred}}, \mathbf{x}_L^{\text{new}}} = \text{WARMSTARTOPT}(\phi_{\mathbf{x}_L^{\text{pred}}, \mathbf{x}_L^{\text{new}}}, \phi_{\mathbf{x}_L^{\text{pred}}, \mathbf{x}_L^{\text{new}}})$ 
12:          if isValid( $\phi_{\mathbf{x}_L^{\text{pred}}, \mathbf{x}_L^{\text{new}}}$ )  $\cup$   $c(\phi_{\mathbf{x}_L^{\text{pred}}, \mathbf{x}_L^{\text{new}}}) < c(\mathbf{x}_L^{\text{new}})$  then
13:             $c(\mathbf{x}_L^{\text{new}}) = c(\phi_{\mathbf{x}_L^{\text{pred}}, \mathbf{x}_L^{\text{new}}})$ 
14:            Set  $\mathbf{x}_L^{\text{pred}}$  as the parent of  $\mathbf{x}_L^{\text{new}}$ 
15:            Store  $\phi_{\mathbf{x}_L^{\text{pred}}, \mathbf{x}_L^{\text{new}}}$ 
16:   return  $\phi_{\mathbf{x}^S, \mathbf{x}^G}$ 
```

---

#### E. Kinodynamic Planning using INSAT

Kinodynamic planning is a class of problems for which velocity, acceleration, and inertial/force/torque bounds must be satisfied, together with kinematic constraints such as avoiding obstacles. However, controllers in fully actuated systems like the vast majority of commercial manipulators do not require a fully dynamically feasible trajectory to track them accurately. In such systems, even a velocity controller is able to track trajectories generated with smooth splines [20] at high accuracy. As a result, planning in the space of the manipulator’s joint space and its derivatives simplifies into finding the parameters of the choice of basis splines. This dramatic reduction in the planning complexity resulting from the spline representation of the manipulator trajectory enables us to use a recent global kinodynamic planning algorithm called INSAT [17] as the preprocessing planner. In this section, we will first explain our choice of splines and provide a high-level overview of INSAT. We refer the reader to [17], [18], [19] for the details of the algorithm and [21] for the parallelized version named PINSAT.

1) *B-Splines:* B-splines are smooth and continuous piecewise polynomial functions made of finitely many basis polynomials called B-spline bases. A  $k$ -th degree B-spline basis with  $m$  control points can be calculated using the Cox-de Boor recursion formula [22] as

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) \quad (2)$$

where  $i = 0, \dots, m$ ,  $\frac{t - t_i}{t_{i+k-1} - t_i}$  and  $\frac{t_{i+k} - t}{t_{i+k} - t_{i+1}}$  are the interpolating coefficients between  $t_i$  and  $t_{i+k}$ . Let us define a non-decreasing knot vector  $\mathbf{T}$  and the set of control points  $\mathbf{P} = \mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m\}$  where  $\mathbf{p}_i \in \mathbb{R}^n$ ,  $i = 0, \dots, m$ . Then a B-spline trajectory can be uniquely determined by the degree of the polynomial  $k$ , the knot vector  $\mathbf{T}$ , and the

set of control points  $\mathbf{P}$  called de Boor points.

$$\mathbf{q}(t) = \sum_{i=0}^m \mathbf{p}_i N_{i,k}(t) \quad (3)$$

The pseudocode of INSAT is given in Alg. 2. INSAT alternates between searching a low-D discrete graph and performing trajectory optimization in high-D to produce smooth full-D trajectories. The low-D variables in the algorithm are denoted with subscript  $L$  and the full-D trajectories connecting two lifted spaces of low-D state are  $\mathbf{x}$  and  $\mathbf{x}'$  is denoted as  $\phi_{\mathbf{x}\mathbf{x}'}$ . For the TRAJOPT step we solve the following optimization problem in Eq. 4 using the aforementioned B-spline representation. The finite parameterization of this problem and how its optimization is interleaved with the low-D search is explained in high detail in [21].

$$\min \quad w_1 t_f + w_2 \int_0^{t_f} \|\dot{\mathbf{x}}(t)\|_2 dt \quad (4a)$$

$$\text{s.t.} \quad \mathbf{x} \in \mathcal{C}^n \quad (4b)$$

$$\mathbf{x}(t) \in \mathcal{X}^{free} \quad (4c)$$

$$\dot{\mathbf{x}}(t) \in [\dot{\mathbf{x}}_{min}, \dot{\mathbf{x}}_{max}] \quad (4d)$$

$$t_f \in [t_{min}, t_{max}] \quad (4e)$$

$$\mathbf{x}(t_0) = \mathbf{x}_0, \mathbf{x}(t_f) = \mathbf{x}_f \quad (4f)$$

$$\dot{\mathbf{x}}(t_0) = \dot{\mathbf{x}}_0, \dot{\mathbf{x}}(t_f) = \dot{\mathbf{x}}_f \quad (4g)$$

The optimization is performed on the control points of the B-splines. The precise details of the optimization are not mentioned here due to the page limit. Once the preprocessing module's request is received, INSAT kicks in to find a smooth B-spline from start to goal. When the optimization output is valid in terms of dynamic limits but results in a collision, we recover the trajectory by caching the iterates and returning the best trajectory to the preprocessor [21].

## F. Perception Module

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

- 1) A single stereo camera is placed on the 8020 aluminum T-slotted profiles, mounted on the robot arm's pedestal. Consequently, precise camera calibration is important to establish the extrinsic rigid body transformation between the camera frame and the robot's base frame.
- 2) The perception system should detect the rapidly moving projectile and provide an accurate projectile estimate of the projectile in real-time during its flight.

We solve the first challenge with hand-eye camera calibration (IV-F.2) and solve the second challenge by performing a least-squares model fitting based on all observations (IV-F.3).

1) *Object Detection*: To detect the incoming projectile, we employ RGB color filtering, as the projectile is characterized by a distinct colored ball. We establish specific lower and upper color thresholds within the HSV color space and then process each frame to generate a binary mask that isolates the ball.

2) *Depth Estimation*: To obtain the 3D global coordinates relative to the robot's base, we calibrate the camera to obtain the transformation between the left camera frame and the robot's base frame. Masked point cloud information and the confidence map are retrieved from the ZED SDK, with our Python API acting as a wrapper around the SDK. We filter outliers and eliminate pointclouds outside the predefined virtual bounding box. Then, we compute the mean values for  $X$ ,  $Y$ , and  $Z$  to derive the 3D centroid point. Only frames that exceed a minimum pixel threshold are considered to reduce noise. Since the ZED camera does not directly measure the depth but estimates it using stereo geometry, some depth points may not be entirely accurate. To address this, we utilize ZED SDK's built-in confident map function and drop depth points with low confidence. Consequently, we perform depth filtering, only retaining frames where the estimated depth falls within the specified distance threshold. The positions in these frames are recorded along with the timestamps, and this iterative process continues until the required number of frames is collected.

3) *Projectile Estimation*: To reconstruct the 3D trajectory of the ball, we use the Ohno method [23]. First, we fit multiple position detections at various time intervals to a projectile equation of motion. It is assumed that the motion is on the  $X - Z$  plane (with  $Z$  pointing vertically upward). Movements perpendicular to this plane that can occur as a result of wind or other external forces are ignored. Once launched, the motion of the ball is governed purely by gravity, which is expressed as

$$X(t) = X(0) + tV_X(0) \quad (5)$$

$$Y(t) = Y(0) + tV_Y(0) \quad (6)$$

$$Z(t) = Z(0) + tV_Z(0) - \frac{1}{2}gt^2 \quad (7)$$

where  $(X(t), Y(t), Z(t))$  and  $(V_X(t), V_Y(t), V_Z(t))$  denote the position and velocity of the ball at time  $t$ ,  $g$  denotes acceleration due to gravity ( $g = 9.81m/s^2$ ). Assuming parabolic motion for the projectile, the estimated position at time  $t$  depends only on the initial position  $(X(0), Y(0), Z(0))$  and the initial velocity  $(V_X(0), V_Y(0), V_Z(0))$ . We can then estimate the values of initial position and initial velocity  $\theta = (X(0), Y(0), Z(0), V_X(0), V_Y(0), V_Z(0))$  by defining an error function that minimizes the sum of squared errors:

$$\theta^* = \arg \min_{\theta} \sum_{t=1}^M \{(X_t - X(t))^2 + (Y_t - Y(t))^2 + (Z_t - Z(t))^2\} \quad (8)$$

where  $X(t), Y(t), Z(t)$  are the estimated positions from Eq. 5-7 and  $X_t, Y_t, Z_t$  are observed positions from the detected frames in IV-F.2.

## V. EXPERIMENTS AND RESULTS

### A. Evaluation in Simulation

We first evaluate the motion planning module in simulation. The simulation environment configuration is shown in Fig. 5, where the yellow cube denotes the inner dome and the red cube denotes the outer dome. A pink cylindrical pole

is an obstacle in the environment. We randomly generated 200 projectiles for the experiment and verified that they intersect with the inner dome and outer dome. The launch distance for each projectile was uniformly sampled from a range of 6–12m. These projectiles have an average time of flight of 1.069s. Note that the time constraint is relaxed due to the addition of the obstacle, forcing the manipulator to take longer to execute a trajectory. All experiments were carried out on an AMD Threadripper Pro 5995WX workstation. We compared our planning framework with two efficient online planners: the Rapidly-exploring Random Tree-Connect (RRT-Connect) [24] and the Edge-based Parallel A\* (ePA\*SE) [25]. The RRT-Connect is known to be efficient in solving single-query path planning problems, while ePA\*SE is a recent work that leverages the power of parallelization to speed up the search. All online planners were given a 2-second budget to plan. We used Toppra [26] to generate a time-optimal trajectory from the geometric path from online planners. The preprocessing time of our planning framework is 5 hours.

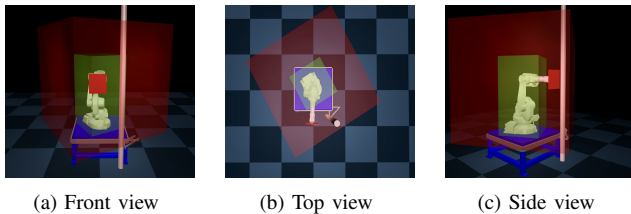


Fig. 5. (a), (b), and (c) are the front, top, and side views of the system in the Mujoco simulator.

Metrics	Our approach	RRT-Connect	ePA*SE
Finds a solution	70.33%	59.81%	50.71%
Successfully Blocks	68.9%	27.27(27.27)%	9.57(11)%
Query time (ms)	$0.109 \pm 0.033$	$17.5 \pm 31.7$	$231 \pm 377$
Execution time (s)	$0.31 \pm 0.2$	$1.3 \pm 0.97$	$1.8 \pm 0.87$

TABLE I  
PROPOSED METHOD vs THE BASELINES IN SIMULATION.

The simulation results for 200 projectiles are shown in Table I. The first row represents the ratio of successful interception solutions found by the planner. The second row indicates the ratio of successful blocking instances, defined as cases where the combined planning query time and execution time are shorter than the time of flight of the incoming projectile. The third and fourth rows present the mean and standard deviation of the planner query time and trajectory execution time. The simulation results demonstrate the superior performance of our method compared to the two baseline approaches. Specifically, our method achieves an exceptional success rate, nearly 2.5x of RRT-Connect and 7x that of ePA\*SE. In terms of the query time, our method is faster than RRT-Connect by a factor of 160 and ePA\*SE by 2119, affirming our planner’s ability to swiftly compute manipulator trajectory paths within a short time window. Moreover, the execution time is substantially shorter than that of the two baselines, ensuring that the shield reaches the goal configuration in time for successful interception. Since the baselines are not constant time planners, we also computed

the successful interception rate assuming zero query time for them (shown within parentheses in the second row). Table II shows the joint-wise RMS torque in Nm across the simulation experiments. Notice a significantly lower net torque in our method for every joint. This is a direct benefit of interleaving search and optimization as opposed to post-processing geometric paths with path parameterizations subject to certain forms of kinematic and dynamic constraints.

Joint ID	1	2	3	4	5	6
RRT-C	$102 \pm 40$	$130.2 \pm 49$	$43.6 \pm 24$	$0.35 \pm 1.3$	$0.13 \pm 0.35$	$0.015 \pm 0.12$
ePA*SE	$61.4 \pm 38$	$131 \pm 52$	$39.5 \pm 14$	$0.34 \pm 1.3$	$0.11 \pm 0.25$	$0.012 \pm 0.09$
Ours	$11 \pm 11$	$24.5 \pm 21$	$16.2 \pm 16$	$0.031 \pm 0.02$	$0.036 \pm 0.03$	$0.0003 \pm 0.0003$

TABLE II  
JOINT-WISE RMS TORQUE (NM) FOR SIMULATION EXPERIMENTS.

### B. Experiments on Robot Hardware without Obstacles

We tested the full system with integrated perception on the ABB’s IRB-1600 robot arm, equipped with an onboard stereo camera ZED 2i, in an indoor environment. We configured the range of attacks to be 6–8m. In this setup, the time of flight of the ball from its first detection by the perception system to its interception with the robot is roughly 350ms. Of this time, on average, 82.9ms is allocated to the perception system, 212.2ms is dedicated to the execution of the robot’s trajectory, and 50ms is attributed to other system overheads. This allocation leaves only 4.9ms on average for the planner. Nonetheless, our planner successfully generates a plan within this time budget.

We performed 50 throws and achieved a blocking success rate of 78%. Among the 11 throws that were unsuccessful, the causes of these failures were traced back to issues within the perception and execution modules. Specifically, in 8 instances, inaccuracies in the perception module arose due to motion blur or noisy depth estimations. The remaining three failures were attributed to slow execution processes. This occurred when the execution time for specific trajectories either outpaced the perception module’s ability to provide timely updates or when executing the entire trajectory consumed more time than the actual flight time of the ball.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we have presented and evaluated a preprocessing-based kinodynamic motion planning framework to intercept projectiles using a robot manipulator. We tested our overall pipeline, which consists of a perception module, a planner module, and an execution module on a physical system made of an ABB industrial arm and a ZED stereo camera. In the future, we would like to extend this work to a mobile base and intercept a sequence of projectiles separated by a short time interval.

## VII. ACKNOWLEDGEMENTS

This work was supported by grants W911NF-18-2-0218 and W911NF-21-1-0050 of the ARL-sponsored A2I2 program.

## REFERENCES

- [1] J. Tebbe, L. Klamt, Y. Gao, and A. Zell, "Spin detection in robotic table tennis," in *2020 IEEE international conference on robotics and automation (ICRA)*, pp. 9694–9700, IEEE, 2020.
- [2] P. Cigliano, V. Lippiello, F. Ruggiero, and B. Siciliano, "Robotic ball catching with an eye-in-hand single-camera system," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 5, pp. 1657–1671, 2015.
- [3] B. Büuml, T. Wimböck, and G. Hirzinger, "Kinematically optimal catching a flying ball with a hand-arm-system," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2592–2599, 2010.
- [4] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters, "Learning to play table tennis from scratch using muscular robots," *arXiv preprint arXiv:2006.05935*, 2020.
- [5] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.
- [6] S. M. Kakade, *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.
- [7] A. Agarwal, Y. Han, and M. 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)*, pp. 10633–10640, IEEE, 2020.
- [8] R. A. Kromer, A. Abellán, D. J. Hutchinson, M. Lato, T. Edwards, and M. Jaboyedoff, "A 4d filtering and calibration technique for small-scale point cloud change detection with a terrestrial laser scanner," *Remote Sensing*, vol. 7, no. 10, pp. 13029–13052, 2015.
- [9] A. Pfister, A. M. West, S. Bronner, and J. A. Noah, "Comparative abilities of microsoft kinect and vicon 3d motion capture for gait analysis," *Journal of medical engineering & technology*, vol. 38, no. 5, pp. 274–280, 2014.
- [10] D. M. Ramík, C. Sabourin, R. Moreno, and K. Madani, "A machine learning based intelligent vision system for autonomous object detection and recognition," *Applied intelligence*, vol. 40, no. 2, pp. 358–375, 2014.
- [11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.
- [12] H. Yang, S. Mukherjee, and M. Likhachev, "A-epa\* se: Anytime edge-based parallel a\* for slow evaluations," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 16, pp. 163–167, 2023.
- [13] F. Islam, O. Salzman, A. Agarwal, and M. Likhachev, "Provably constant-time planning and replanning for real-time grasping objects off a conveyor belt," *The International Journal of Robotics Research*, vol. 0, no. 0, p. 02783649211027194, 0.
- [14] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara\*: Anytime a\* with provable bounds on sub-optimality," *Advances in neural information processing systems*, vol. 16, 2003.
- [15] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.
- [16] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artificial intelligence*, vol. 1, no. 3-4, pp. 193–204, 1970.
- [17] R. Natarajan, H. Choset, and M. Likhachev, "Interleaving graph search and trajectory optimization for aggressive quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5357–5364, 2021.
- [18] R. Natarajan, G. L. H. Johnston, N. Simaan, M. Likhachev, and H. Choset, "Torque-limited manipulation planning through contact by interleaving graph search and trajectory optimization," 2022.
- [19] R. Natarajan, G. L. H. Johnston, N. Simaan, M. Likhachev, and H. Choset, "Long horizon planning through contact using discrete search and continuous optimization," 2024.
- [20] D. Constantinescu and E. A. Croft, "Smooth and time-optimal trajectory planning for industrial manipulators along specified paths," *Journal of robotic systems*, vol. 17, no. 5, pp. 233–249, 2000.
- [21] R. Natarajan, S. Mukherjee, H. Choset, and M. Likhachev, "Pinsat: Parallelized interleaving of graph search and trajectory optimization for kinodynamic motion planning," 2024.
- [22] N. M. Patrikalakis, "Approximate conversion of rational splines," *Computer Aided Geometric Design*, vol. 6, no. 2, pp. 155–165, 1989.
- [23] Y. Ohno, J. Miura, and Y. Shirai, "Tracking players and estimation of the 3d position of a ball in soccer games," *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, vol. 1, pp. 145–148 vol.1, 2000.
- [24] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2, pp. 995–1001, IEEE, 2000.
- [25] S. Mukherjee, S. Aine, and M. Likhachev, "epa\* se: Edge-based parallel a\* for slow evaluations," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 15, pp. 136–144, 2022.
- [26] H. Pham and Q.-C. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 645–659, 2018.