Revisiting Ball Catching with a Mobile Manipulator: A Discrete Trajectory Planning Approach

Ke Dong[1], Florian Shkurti[2] and Angela P. Schoellig[1]

***Abstract*— Catching a ball with a mobile manipulator is a classic robotics benchmark that requires continuously interleaving prediction and re-planning in order to make a successful grasp mid-air. Existing solutions to this problem typically make use of sequential quadratic programming, which is sensitive to initial conditions. We revisit this task by showing that the number of degrees-of-freedom that are relevant to the optimization problem is small enough to search via a simple, two-step, grid search and verification scheme. Through extensive simulations and experiments on a real robot manipulator we show that our method leads to at least as high catching success rate as sequential quadratic programming, while being much simpler to implement and without relying on heuristics for initializing the optimizer. Our findings suggest that this classic benchmark should be updated to more high-dimensional scenarios and arbitrary objects.**

I. Introduction

Real-time trajectory planning for catching an object mid-air is a task that robots are expected to perform in a fraction of a second, while interleaving the prediction of the object's trajectory with the optimization of the desired catching time and catching pose of the end-effector. Catching is perceived to be difficult because every component of the state, including the mobile base, the robot arm, and the object in the environment, are all in motion, and there is little room for error in prediction and control. In addition, there are complex trajectory constraints that need to be resolved in real-time, including joint motion limits and collision avoidance, both in terms of the body of the robot and in terms of avoiding dynamic obstacles in the environment.

Catching has served as an interesting benchmark in robotics because it is a prerequisite skill for other fast and dynamic robot movements such as juggling, playing table tennis or other similar games, and fluid human robot interaction in general.

Existing solutions [1], [2] address the problem of catching by formulating it as a Sequential Quadratic Programming (SQP) problem, constrained by non-linear equality and inequality conditions. Owing to the nature of non-linear optimization, these algorithms are computationally expensive, sensitive to initialization, and suffer from local minima, due to the non-convex kinematics constraint functions.

[1] Ke Dong and Angela P. Schoellig are with the Dynamic Systems Lab (www.dynsyslab.org) at the University of Toronto Institute for Aerospace Studies (UTIAS), Canada. Email: ke.dong@robotics.utias.utoronto.ca, schoellig@utias.utoronto.ca

[2] Florian Shkurti is with Department of Computer Science at the University of Toronto, Canada. Email: florian@cs.toronto.edu

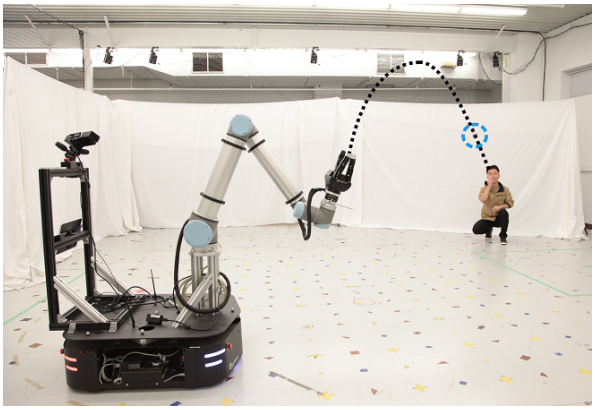

Fig. 1. A human throws a ball from a distance of about 4 m towards the robot with a typical speed of 6 m/s. The duration of the flying ball's trajectory is around 1 second. Video of the mobile manipulator catching the ball with a cup supported by a three-finger gripper is available at http://tiny.cc/udpscz.

In this paper we present and evaluate a simple discrete search trajectory planning algorithm, which is fast, robust to initial conditions, and easy to implement. We show that the number of unconstrained optimization variables that are required to generate catching motions is small enough that simple grid search is sufficient to produce good catching solutions, without resorting to nonlinear optimization. The key ideas here are to first generate a large number of catching configuration samples via discrete search and analytic Inverse Kinematics (IK) solvers, and then check the feasibility of every catching configuration, at the end selecting to execute the one that has the lowest cost.

A. Related work

A large body of work has been devoted to the trajectory generation and autonomous control of fast manipulator movements, such as for object catching and hitting. Early pioneering work in this field used heuristic methods for selecting the catching point and generating the joint trajectory via interpolation [3], [4]. Recent work uses non-linear optimization methods to solve this problem. In some of the most compelling experiments in this area [1], a ball catching system from DLR consisting of a 7DOF arm and a four-finger hand is presented. Later, an omni-directional mobile base with two identical manipulators is shown in [5], which can catch two balls, thrown simultaneously. To the best of our knowledge, this is the only other work on mobile manipulator ball catching in recent years. These two papers from DLR use sequential quadratic programming for optimization, which is computationally expensive and requires careful initialization

of the optimization parameters. This system also requires 32-core parallel programming for real-time performance.

Other types of high-speed catching and manipulation systems were shown in [6] as well as [7]. The presented system in [7] was able to catch fast-moving objects softly via a dynamical systems (DS) based control law. Arbitrary-shaped object catching is another interesting and more challenging task. The motion of complex objects like a hammer or a tennis racket can not be simply modelled as ballistic movement. Different methods have been used to learn the arbitrary-shaped object trajectory including rigid body dynamics model [8], learning from demonstration [9] and Gaussian mixture model [10]. The latter, in particular, was able to catch arbitrarily-shaped objects, as long as they had been seen during an offline modeling and training phase.

A similar sample-verification framework as ours is proposed in [11]. But it is used for quadrotor trajectory planning where no complex forward kinematics constraints are needed to be considered. Note that our method is also different from sampling-based trajectory planning algorithms such as probabilistic roadmaps (PRM) [12] and rapidly exploring random trees (RRT) [13]. They aim to find a sequence of feasible way-points to form the trajectory, and are usually time-consuming when finding an optimal trajectory is required. Our method, on the other hand, aims to find only one optimal point in the solution space and real-time performance is required.

B. Contributions

The contribution of this paper is two-fold. First, we provide a much simpler trajectory generation algorithm, compared to existing methods, for the task of catching balls with a mobile manipulator. Our method first use analytic IK solvers to reduce the number of optimization variables to just four, and then decomposes the problem into generating possible catching samples and checking for their feasibility. This approach is fast, robust to different initial conditions and considerably easier to implement than existing methods based on nonlinear optimization.

Our second contribution is an extensive evaluation and comparison of the proposed algorithm against the SQP planner, both in simulation and in real robot experiments. We note that SQP has been the method of choice for ball catching experiment in existing works. In these comparisons we show that our simple method can achieve at least as high catching success rate as SQP with slightly less computation time on average, both in simulation and in real robot experiments. This finding suggests that the ball catching benchmark task should be updated to more high-dimensional scenarios, e.g. catching arbitrarily-shaped objects mid-air using RGBD images or other similar observations.

II. System Overview and Problem Formulation

This section first presents the overview of our system, including the mobile manipulator and the flying ball. A formal statement of the optimal trajectory planning problem is then given.

A. System Overview

The mobile manipulator system considered in this paper consists of a 6-DOF manipulator rigidly mounted on a 3-DOF omnidirectional mobile base. The yaw rotation of the mobile base is not used in this paper since rotating a mobile base is usually slow. Thus the system has 8 DOF in total. The joint configuration vector of the robot is $\mathbf{q} = [\mathbf{q}_a, \mathbf{q}_b]^\top$, where $\mathbf{q}_a = [\theta_1, \theta_2, ..., \theta_6]^\top$ is the arm joint angles and $\mathbf{q}_b = [x_b, y_b]$ represents the Cartesian position of the mobile base. The kinematics model of the mobile manipulator is shown in Figure 2, and is assumed to be known.

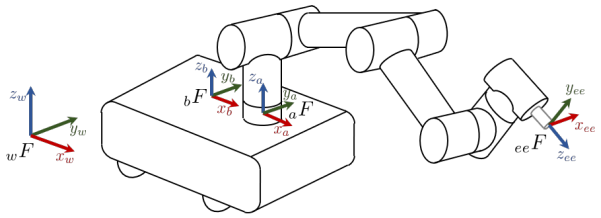

Fig. 2. The kinematics model of the mobile manipulator. The world frame ${}_wF$, base frame ${}_bF$, the arm frame ${}_aF$ and the end-effector frame ${}_{ee}F$ are presented. Adapted from [14].

The flying ball's movement is modeled as free-fall motion without air drag:

$$\ddot{\mathbf{b}} = -\mathbf{g}, \tag{1}$$

where $\mathbf{b} = [b_x, b_y, b_z]^T$ is the ball's position, and $\mathbf{g} = 9.81$ is the gravitational acceleration. Then, the ball's trajectory in the x, y, z axis can be fitted with parabola curves:

$$\begin{aligned} b_x(t) &= a_{0,x}t^2 + a_{1,x}t + a_{2,x}, \\ b_y(t) &= a_{0,y}t^2 + a_{1,y}t + a_{2,y}, \\ b_z(t) &= a_{0,z}t^2 + a_{1,z}t + a_{2,z} \end{aligned} \tag{2}$$

where $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2$ are curve coefficients and can be estimated via online least squares methods. Though air drag is reported to have non-negligible influence on the ball's trajectory prediction [1], [5], its influence on the trajectory can be handled during the online trajectory coefficient fitting process.

B. Problem Statement

To successfully catch the ball, the trajectory planner needs to first select the catching configuration $\mathbf{z} = [\mathbf{q}_f, t_f]^\top$ where $\mathbf{q}_f$ is the final robot joint configuration and t_f is the catching time, and then generate joint trajectories in the joint space that lead the end-effector to the desired pose. Pre-selected trajectory parameterization methods are usually used in this field so that given the initial and final joint position, the joint trajectory can be uniquely generated and determined [1], [5], [15]. In this way, the only problem left here is to find the catching configuration.

Following this line of work, the main problem to be addressed in this paper is to search for the optimal catching configuration $\mathbf{z}$, given joint motion limits, collision avoidance requirements, a trajectory parameterization method, and a user-defined cost function. This can be formulated as the following optimization problem:

$$\begin{aligned} \mathbf{q}_f^*, t_f^* = \arg\min_{\mathbf{q}_f, t_f} \quad & C(\mathbf{q}_f, t_f) \\ \text{s.t.} \quad & {}^{ee}_{w}\mathbf{FK}_p(\mathbf{q}_f) = \mathbf{b}_p(t_f), \\ & {}^{ee}_{w}\mathbf{FK}_n(\mathbf{q}_f) = -\mathbf{b}_v(t_f), \\ & \mathbf{g}_{box}(\mathbf{q}_f, t_f) \leq \mathbf{0}, \\ & \mathbf{g}_{coll}(\mathbf{q}_f, t_f) \leq \mathbf{0}, \end{aligned} \tag{3}$$

where ${}^{ee}_{w}\mathbf{FK}_p(\cdot)$, ${}^{ee}_{w}\mathbf{FK}_n(\cdot)$ are the robot's forward kinematics functions that return the end-effector's position and normal vector in the world frame, while $\mathbf{b}_p(\cdot), \mathbf{b}_v(\cdot)$ are the ball's trajectory prediction functions that return the ball's position and velocity vector. We impose box constraints $\mathbf{g}_{box}$ to restrict the robot's joint position, velocity and acceleration within safe ranges:

$$\begin{aligned} q_{im} \leq q_i(t) \leq q_{iM}, \\ \dot{q}_{im} \leq \dot{q}_i(t) \leq \dot{q}_{iM}, \\ \ddot{q}_{im} \leq \ddot{q}_i(t) \leq \ddot{q}_{iM}, \end{aligned} \tag{4}$$

where $q_{im}, \dot{q}_{im}, \ddot{q}_{im}$ and $q_{iM}, \dot{q}_{iM}, \ddot{q}_{iM}$ are minimum and maximum joint position, velocity and acceleration respectively. We also set $\mathbf{g}_{coll}$ to be the collision avoidance constraints, determined by a geometric model of the robot and environment.

III. Methodology

This section presents our discrete search trajectory planning algorithm. It starts with the trajectory parameterization method and constraint simplification used in this paper for real-time performance. Then the detailed procedure of the trajectory planning algorithm is presented.

A. Trajectory Parameterization and Constraint Simplification

Many trajectory parameterization methods have been used in literature for different considerations, such as minimal acceleration, path smoothness, and maximal reachable space [1], [5], [2]. In this paper, we use the trapezoidal velocity ramp to maximize the reachable space.

This trapezoidal velocity ramp trajectory consists of an acceleration phase and a cruise phase. For the i-th robot joint, it will first accelerate to the cruise velocity $\dot{q}_{i,f}$ with the maximum joint acceleration $\ddot{q}_{iM}$, and then maintain this speed until the catching time. A dedicated stopping trajectory will be appended after the catching to make the robot stop. This happens after catching and is not considered in the optimization problem.

The trapezoidal velocity ramp trajectory is parameterized by two parameters: the duration time $t_{i,a}$ and cruise velocity $\dot{q}_{i,f}$. To determine these two parameters and generate the joint trajectory, the kinematics equations in Eqn 5 need to be solved, which is obtained from the relationship between position, velocity and acceleration for an uniform acceleration motion [1].

$$\begin{aligned} q_{i,0} + \frac{1}{2}(\dot{q}_{i,f} + \dot{q}_{i,0})t_{i,a} + \dot{q}_{i,f}(t_f - t_{i,a}) = q_{i,f}, \\ \dot{q}_{i,f} - \dot{q}_{i,0} = \ddot{q}_{iM} t_{i,a}, \end{aligned} \tag{5}$$

where $q_{i,0}, \dot{q}_{i,0}$ are the initial joint position and velocity, $q_{i,f}$ is the final joint position at the catching time, and t_f is the catching time.

The box constraints over the entire joint trajectory in Eqn 4 can be simplified due to this parameterization approach. The joint's maximum acceleration is used in the acceleration phase, thus the acceleration limit is automatically satisfied. The initial point and end point of the trapezoidal curve are the position and velocity extrema points. Thus, only the end point are necessary for checking, since the initial position and velocity are given. Then, the box constraints in Eqn 4 can be simplified as below:

$$\begin{aligned} q_{im} \leq q_{i,f} \leq q_{iM}, \\ \dot{q}_{im} \leq \dot{q}_{i,f} \leq \dot{q}_{iM}, \end{aligned} \tag{6}$$

A complete collision avoidance check is very time-consuming and unnecessary here since no obstacles are considered. We do, however, use a similar collision heuristic as [1], namely only the origin of the end-effector's frame is tested for whether it is inside the collision-free space, and even that is done only for the final catch configuration $\mathbf{q}_f$. The collision-free space is a semi-cylinder with its bottom centered in the arm frame's origin as shown in Figure 3, which is to prevent the end-effector from colliding with the ground floor and the mobile base:

$$\begin{aligned} {}_aP_{ee,x}^2 + {}_aP_{ee,y}^2 \leq R_{coll}^2 \\ 0 \leq {}_aP_{ee,z} \leq H_{coll}, \end{aligned} \tag{7}$$

where $[{}_aP_{ee,x}, {}_aP_{ee,y}, {}_aP_{ee,z}]$ is the end-effector's position in the arm frame, R_{coll} is the cylinder's radius and H_{coll} is the height. Though self-collision is not considered here, it never happens during the real robot experiment, which is also reported in [1].

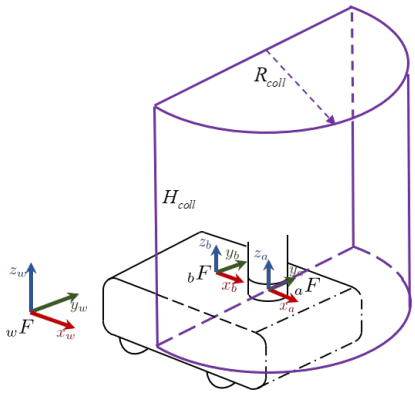

Fig. 3. The geometric model of the collision-free space. This is a semi-cylinder with its bottom centered in the arm frame's origin.

B. Discrete Search Trajectory Planning Algorithm

1) Searching over Catching Times and Poses: The most direct way to solve the optimization problem in Eqn 3 is to take uniform samples on the nine-dimensional catching configuration space, which suffers from the curse of dimensionality. The equality constraints, namely, the three-dimensional position specification and the two-dimensional

orientation requirement of the end-effector, manage to reduce the dimension of the solution space from 9 to 4. This fact can be explicitly used by an analytic IK solver, which can help us determine five joint variables in the kinematics chain given the five-dimensional end-effector specification. In other words, we only need to take samples on the catching time t_f and three robot joint variables.

Though different selection choices exist, this paper chooses the sixth joint of the arm q_6 and mobile base's Cartesian position q_7, q_8. As can be seen in Figure 2, changing the sixth joint will make the end-effector rotate along its z axis, which has no influence on the ball catching task. Thus, the sixth joint of the arm is fixed to a constant value $\bar{q}_6$.

Given the sample space $\mathscr{S} = [q_{7m}, q_{7M}] \times [q_{8m}, q_{8M}] \times [0, T_{max}]$, we uniformly take samples $\mathbf{s}^k = [q_7^k, q_8^k, t_f^k]^\top$ with step size $\delta x, \delta y, \delta t_f$. Here T_{max} is a parameter for maximum catching time and q_{7m}, q_{7M}, q_{8m}, q_{8M} are the mobile base's position limits. Then the IK solver can determine the arm joint position:

$$\mathbf{q}_a^k = IK(\mathbf{b}_p(t_f^k), -\mathbf{b}_v(t_f^k), \mathbf{s}^k), \tag{8}$$

If an IK solution is found, $\mathbf{s}^k$ and $\mathbf{q}_a^k$ are put together to form the catching configuration sample $\mathbf{z}_k$. Otherwise, sample $\mathbf{s}^k$ is discarded. Then, for every catching configuration sample, Eqn 5 is solved to generate the corresponding trapezoidal joint trajectories.

2) Feasibility Verification: This step checks the feasibility constraints of every catching configuration sample. By feasibility constraints, we refer to the inequality constraints in Eqn 3. Though a check over the entire trajectory should be done to ensure feasibility, only the final catching configuration is checked herein, due to the simplification made in Section III-A. Thus, given the catching configuration sample $\mathbf{z}^k$, the simplified inequality constraints in Eqn 6 and 7 are calculated to determine feasibility. If there is more than one feasible sample, the cost function is used to select the optimal one, otherwise, no output is generated and the algorithm fails to find a catching configuration. Our trajectory planning algorithm is summarized in Algorithm 1.

C. Discussion on Analytic IK Solutions

Though the general inverse kinematics solution is still an open problem in the robotics field, the analytic inverse kinematics solutions for common types of end-effector pose specification and robot kinematics chains can be obtained, via kinematic decoupling and geometric analysis. For example, for manipulators having six joints with the last three joints intersecting at a point, an analytic IK solution is available [16]. Some toolbox libraries, such as IKFast [17], also provide analytic solutions for common end-effector pose specification such as 6D transformation, 3D rotation and 3D translation. Thus, having an analytic IK solution for a specific manipulator platform without kinematic redundancy should not be an unreasonable assumption.

Algorithm 1 Discrete Search Trajectory Planning

Input: Robot's initial joint position $\mathbf{q}_0$ and velocity $\dot{\mathbf{q}}_0$, sample space $\mathscr{S}$, sampling step size $\delta x, \delta y, \delta t_f$, ball trajectory prediction function $\mathbf{b}_p(\cdot), \mathbf{b}_v(\cdot)$, IK solver $IK(\cdot)$.
Output: the optimal catching configuration $\mathbf{z}^*$ or *None*.

```
Initialize set S ← ∅, Z ← ∅ and Q ← ∅.
Take uniform samples s^k = [q_7^k, q_8^k, t_f^k] on 𝒮 with step size δx, δy, δt_f, and add them to the set S.
for the k-th element s^k in S do
    if IK(b_p(t_f^k), −b_v(t_f^k), s^k) has a solution q_a^k then
        z^k = [q_a^k, s^k]^⊤, and add z^k into the set Z
    end if
end for
for the j-th element z^j in Z do
    Calculate joint trajectory parameters (Eqn 5)
    Verify feasibility constraints (Eqn 6 and 7)
    if feasible then
        add z^j into the set Q
    end if
end for
if set Q is not empty then
    return z* = min C(z^j)
else
    return None
end if
```

IV. Computation Time and Trajectory Cost

This section presents the computation time and trajectory cost evaluation of the proposed discrete-search planning algorithm under different ball trajectories, and its comparison with the SQP planning algorithm.

A. General Setup

We first start by fully specifying the nonlinear optimization problem in Eqn 3, where the cost function and constraint parameters are left to be determined. Different cost functions can be used for this ball catching task, but here we choose to penalize the movement of the arm and the mobile base:

$$C(\mathbf{q}_f, t_f) = w_1 \|\mathbf{q}_{a,0} - \mathbf{q}_{a,f}\|_2^2 + w_2 \|\mathbf{q}_{b,0} - \mathbf{q}_{b,f}\|_2^2, \tag{9}$$

where $\|\cdot\|$ is the Euclidean distance, and w_1, w_2 are weight parameters. The catching time t_f is not used here. Since the control performance of the mobile base is usually worse than the arm, we let $w_1 = 1, w_2 = 5$ to further limit the mobile base's movement. The forward kinematics model, joint motion limits and collision-free space of the robot presented in Sec V-A are used here as nonlinear constraints.

To evaluate the planning algorithms, the robot's mobile base is put in the origin with a fixed arm joint configuration. 9000 different ball trajectories are generated in such a way that they will fly to a sphere centered on the end-effector after 0.7s, which is reachable for the mobile manipulator within $0.7s$. During the test, all 9000 ball trajectory prediction functions are called by the planning algorithms. If the algorithm can output a catching configuration within a given

computation threshold for a ball trajectory input, and this configuration can generate a valid joint trajectory, i.e. Eqn 5 has a valid solution, then this is regarded as a success.

Both our search-based planning algorithm and the SQP-based algorithm are implemented in C++. For our search algorithm, the candidate sample space is $\mathscr{S} = [-0.35, 0.35] \times [-0.35, 0.35] \times [0, 1]$ with sampling step size $\delta x = 0.05, \delta y = 0.05, \delta t_f = 0.05$, and we fix the sixth arm joint angle to be $\pi/2$. This will generate at most 3920 raw catching configurations at every run.

The analytic IK solver $IK(\cdot)$ used here is based on the analytic IK solution $IK_{ur10}(\cdot)$ for the UR10 arm in [18]. It can output the six-dimensional joint configuration $\mathbf{q}_a = IK_{ur10}(\mathbf{T}^a_{ee})$ when given the homogeneous transformation matrix from the end-effector to the arm base frame $\mathbf{T}^a_{ee}$.

The SQP-based planning algorithm is implemented with the Nlopt library. For a fair comparison, the same trajectory generation method and constraint simplification stated in Sec III-A are also used here. For the initial guess to the SQP algorithm, the initial catching configuration is set to be the current robot configuration, while the initial catching time is manually set to be $0.5s$. Different initial catching times were tested and there was no obvious influence on the SQP algorithm's result. This initialization approach is reported to have reasonable performance in [2].

All experiments below, including the real robot experiment in Sec V-B, are running on a laptop computer with an Intel Core i7-8850H CPU running at 2.6 GHz.

B. Evaluation results

TABLE I

PERFORMANCE COMPARISON BETWEEN OUR DISCRETE SEARCH ALGORITHM AND THE SQP PLANNER.

Algorithm	Success rate	Computation time(s)	Trajectory cost
DS(0.05, 0.05, 0.05)*	99.49%	0.0070	1.0931
DS(0.10, 0.10, 0.05)	99.33%	0.0022	1.1080
DS(0.05, 0.05, 0.01)	99.60%	0.0385	1.0608
SQP	99.27%	0.0016	1.0223

* DS(0.05, 0.05, 0.05) refers to our discrete search trajectory planning algorithm with sampling step size $\delta x = 0.05, \delta y = 0.05, \delta t_f = 0.05$.

The performance of the two algorithms is shown in Table I. It can be seen that the discrete search algorithm enjoys slightly higher success rate than the SQP algorithm, but also has slightly higher cost. The reason is that the discrete search algorithm will traverse the whole solution space in a discrete fashion. This can make it find a reasonable solution but it may not the best solution. As for the computation time, the SQP is the fastest one and the discrete search algorithm does not scale well when the step size is small. But with appropriate step size like DS(0.10, 0.10, 0.05), the simple discrete search algorithm can achieve similar performance to the SQP.

V. ROBOT EXPERIMENT AND RESULTS

This section presents the setup of the real robot experiments results. The system architecture is first introduced, then the experimental results are presented.

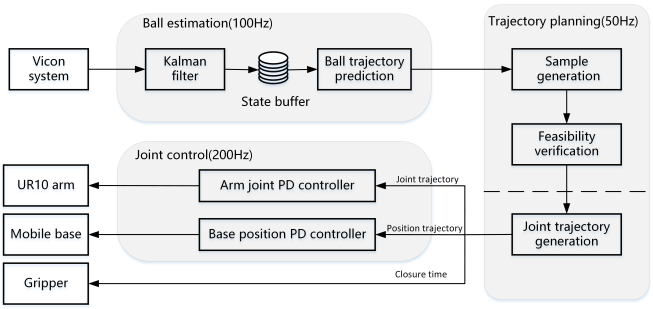

Fig. 4. The diagram of the whole robot system. It consists of three modules including ball estimation, trajectory planning and joint control. These three modules are running on the same laptop to avoid the time synchronization problem.

A. System architecture

Our system consists of three modules including ball estimation, trajectory planning and joint control. The whole diagram is shown in Figure 4. The ball is made visible to the Vicon motion capture system via retro-reflective tapes. Its position and velocity are estimated by a Kalman filter. A buffer of $N = 30$ ball states is kept for fitting the ball's trajectory.

The mobile manipulator used in this experiment includes a 6-DOF UR10 arm and a 3-DOF Ridgeback omni-directional mobile base. The joint motion limits of the arm and the base are presented in Table II, and the collision-free model parameters are $R_{coll} = 1.36m, H_{coll} = 2m$. PD controllers are used to control the arm and the base via velocity commands, which are then translated into joint torques and wheel velocities via the arm and the base's on-board controllers, respectively.

The optimization problem settings in Sec IV-A are used here for trajectory generation. The sample space for the discrete search algorithm here is $\mathscr{S} = [-0.21, 0.21] \times [-0.21, 0.21] \times [0, T_{max}]$. The maximum catching time T_{max} is the time left for the ball to fall into the ground, which can be solved by the free-fall motion model in run time:

$$b_{z,0} + \dot{b}_{z,0} T_{max} - 0.5 g T^2_{max} = 0 \tag{10}$$

The sample step size used in the experiment is $\delta x = 0.05, \delta y = 0.05, \delta t_f = 0.05$. With the reduced sample space for the mobile base (the maximum x position reduced from 0.36 to 0.21), the DS(0.05, 0.05, 0.05) here has similar computation time as DS(0.10, 0.10, 0.05) in Table I.

TABLE II

JOINT LIMITS OF THE MOBILE MANIPULATOR PLATFORM

Joint	q_{min}	q_{max}	$\dot{q}_{max}$	$\ddot{q}_{max}$
arm 1	$-180(^\circ)$	$180(^\circ)$	$86(^\circ/s)$	$458(^\circ/s^2)$
2	$-180(^\circ)$	$180(^\circ)$	$86(^\circ/s)$	$458(^\circ/s^2)$
3	$-180(^\circ)$	$180(^\circ)$	$86(^\circ/s)$	$458(^\circ/s^2)$
4	$-180(^\circ)$	$180(^\circ)$	$115(^\circ/s)$	$458(^\circ/s^2)$
5	$-180(^\circ)$	$180(^\circ)$	$115(^\circ/s)$	$458(^\circ/s^2)$
6	$-180(^\circ)$	$180(^\circ)$	$180(^\circ/s)$	$458(^\circ/s^2)$
base x	$-2(m)$	$2(m)$	$0.9(m/s)$	$1.5(m/s^2)$
base y	$-2(m)$	$2(m)$	$0.9(m/s)$	$1.5(m/s^2)$

B. Robot Experimental Results

The geometric setup in Figure 1 is used for testing. The robot is placed around $(-1,-1,0)$ in the world frame, and the ball is thrown around $(2.5,2.5,0)$ towards the robot. A half-cut bottle is mounted on the gripper for catching the ball. To test the two algorithms, the ball is thrown 75 times for each of the two algorithms [1]. All 150 throws in total, are made by a human operator, and the distribution of ball trajectories is shown in Figure 5. Figure 6 and 7 show the arm and the mobile base's desired and actual position trajectory for a flying-ball trajectory when using the discrete search planning algorithm. The success rate and average computation time are presented in Table III. Videos of the experiment are available at http://tiny.cc/udpscz.

TABLE III

PERFORMANCE COMPARISON OF THE DISCRETE SEARCH AND SQP TRAJECTORY PLANNING ALGORITHMS ON REAL ROBOT EXPERIMENTS.

	Success rate	Computation time(s)
DS(0.05,0.05,0.05)	76.00%	0.00608
SQP	69.33%	0.00657

Surprisingly, the simple discrete search planning algorithm achieves higher success rate and slightly smaller computation time than the SQP planner. The reason for the smaller computation time of the discrete search algorithm, which is in contrast to the result presented in Table I, is that as the ball flies, the maximal catching time T_{max} decreases, and thus fewer samples over the catching time are needed. This accelerates the discrete search algorithm for real robot experiments.

The results here experimentally validate that a simple discretization-based method can achieve similar performance as an non-linear optimization algorithm. This mainly results from the embedding of an analytic IK solver which reduces the number of optimization variable and accelerates the computation process. The comparison here also shows that ball catching with only position and orientation specification under reasonable simplification does not really require a sophisticated approach. Upgrade on this classical ball catching benchmark problem should be made.

VI. CONCLUSION

We re-examined the classic robotics benchmark problem of catching a ball, this time with an off-the-shelf mobile manipulator. The main insight brought forth by our paper is that due to the structure of the optimization problem, the solution space is 4- and not 9-dimensional, as it has been traditionally modeled. Therefore, a simple search-based trajectory optimization method, together with an analytic inverse kinematics solver are sufficient to plan dynamic ball catching trajectories in real time. In fact, we compare this simple method with existing approaches based on sequential quadratic programming and we find that it performs equally well in simulation, and surprisingly, outperforms existing methods in real robot experiments, both in terms of computation time, and in terms of success rate. Our findings suggest that the community should revise the definition of this classic benchmark problem, by raising the bar to more high-dimensional catching scenarios, where the goal is to catch arbitrary objects, or use high-dimensional observations.

[1]More than 75 throws are made in the experiment since some of the ball trajectories are infeasible for the robot to catch.

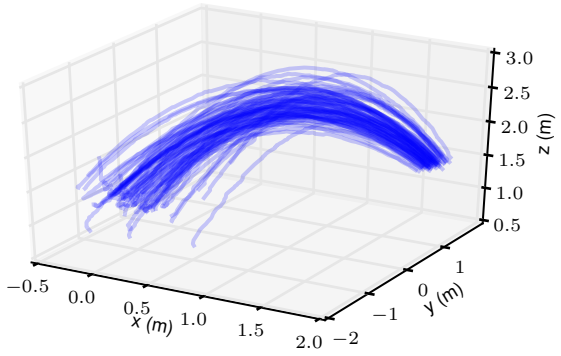

Fig. 5. The 75 ball trajectories for testing the discrete search sampling algorithm.

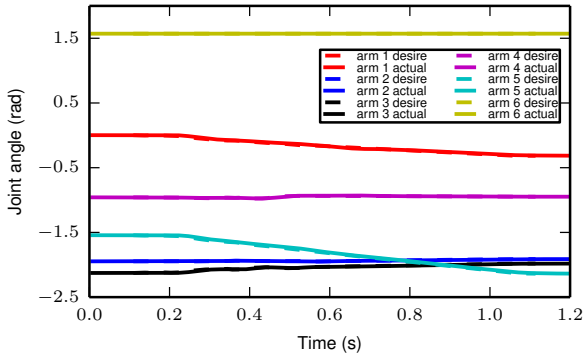

Fig. 6. The arm's desired and actual trajectory of a thrown ball when using the discrete search sampling algorithm.

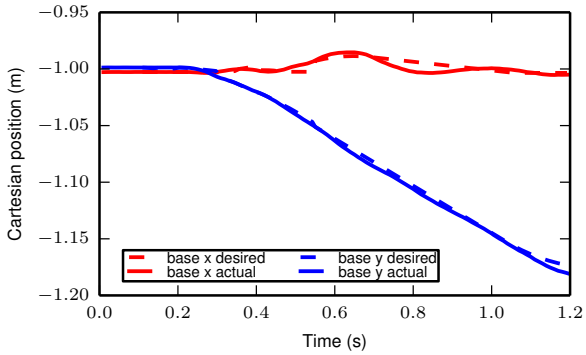

Fig. 7. The mobile base's desired and actual trajectory of a thrown ball when using the discrete search sampling algorithm.

ACKNOWLEDGMENT

The authors would like to acknowledge the Natural Sciences and Engineering Research Council (NSERC) for their generous support.

REFERENCES

[1] B. Bäuml, T. Wimböck, and G. Hirzinger, "Kinematically optimal catching a flying ball with a hand-arm-system," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2010, pp. 2592–2599.

[2] O. Koç, G. Maeda, and J. Peters, "Online optimal trajectory generation for robot table tennis," *Robotics and Autonomous Systems*, vol. 105, pp. 121–137, 2018.

[3] U. Frese, B. Bauml, S. Haidacher, G. Schreiber, I. Schäfer, M. Hahnle, and G. Hirzinger, "Off-the-shelf vision for a robotic ball catcher," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3. IEEE, 2001, pp. 1623–1629.

[4] C. Smith and H. I. Christensen, "Using cots to construct a high performance robot arm," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2007, pp. 4056–4063.

[5] B. Bäuml, O. Birbach, T. Wimböck, U. Frese, A. Dietrich, and G. Hirzinger, "Catching flying balls with a mobile humanoid: System overview and design considerations," in *IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2011, pp. 513–520.

[6] A. Namiki, Y. Nakabo, I. Ishii, and M. Ishikawa, "High speed grasping using visual and force feedback," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, May 1999, pp. 3195–3200.

[7] S. S. Mirrazavi Salehian, M. Khoramshahi, and A. Billard, "A dynamical system approach for catching softly a flying object: Theory and experiment," *IEEE Transaction on Robotics*, 2016.

[8] K. S. Bhat, S. M. Seitz, J. Popović, and P. K. Khosla, "Computing the physical parameters of rigid-body motion from video," in *European Conference on Computer Vision*. Springer, 2002, pp. 551–565.

[9] S. Kim and A. Billard, "Estimating the non-linear dynamics of free-flying objects," *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1108–1122, 2012.

[10] S. Kim, A. Shukla, and A. Billard, "Catching objects in flight," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1049–1065, Oct 2014.

[11] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[12] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[13] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[14] M. Jakob, "Position and force control with redundancy resolution for mobile manipulators," Master's thesis, University of Stuttgart, Stuttgart, Germany, 2018.

[15] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters, "Trajectory planning for optimal robot catching in real-time," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 3719–3726.

[16] M. W. Spong, S. Hutchinson, and V. Mathukumalli, *Robot Modeling and Control*. John Wiley and Sons, 2006.

[17] "Ikfast: The robot kinematics compiler," http://openrave.org/docs/0.8.2/openravepy/ikfast/, [Online; accessed 13-September-2019].

[18] K. P. Hawkins, "Analytic inverse kinematics for the universal robots ur-5/ur-10 arms," Georgia Institute of Technology, Tech. Rep., 2013.