2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

A Stochastic Dynamic Motion Planning Algorithm for Object-Throwing

Avishai Sintov and Amir Shapiro

***Abstract*— A novel algorithm is proposed for offline motion planning of a robotic arm to perform a throw task of an object to reach a goal target. The planning algorithm searches for a throw trajectory that could be performed under kinematic and dynamic (i.e., kinodynamic) constraints. We parameterize the throw trajectory by a time-invariant high-dimensional vector. Then, the kinodynamic and target constraints are formulated in terms of time and the parameterization vector. These constraints form time-varying subspaces in the parameterization space. We present a stochastic method for finding a feasible and optimal solution within the subspace. The method generates a number of random points within the parameterization space and checks their feasibility using an adaptive search. The algorithm is guaranteed under a known probability to find a solution if one exists. We present simulations and experiments on a 3R manipulator to validate the method.**

I. Introduction

A throw motion with the aim of having an object reach a desired target task (position and orientation) out of the robots reach at some time is a key component in many tasks such as juggling, dynamic regrasping, non-prehensile manipulations, and rapid part transfer/delivery. Research on robotic throwing motion includes motion planning with simple low degree-of-freedom robotic manipulators [1], [2] or the imitation of the human arm motion pitching an object [3], [4]. Sampling-based motion planners are commonly used for throwing applications [5], a notion that we use in this work.

In such motion we encounter a problem of generating a feasible trajectory under the kinodynamic constraints for the end-effector to throw the object to the target. That is, finding a trajectory that the arm can follow. This is a motion planning problem. The common motion planning methods for such problems are probabilistic sampling as analytical solutions are difficult to obtain if not intractable [6]. Probabilistic sampling methods are generally easy to implement with relatively low complexity, good results, and efficiency in high degrees of freedom systems. The most common probabilistic methods are the Probabilistic Roadmaps (PRM) [7] and the Rapidly-exploring Random Trees (RRT). The RRT was first introduced in [8] as a randomized approach for kinodynamic planning. It incrementally builds a roadmap tree in the state space while integrating the control inputs to ensure that the kinodynamic constraints are satisfied. Other versions and extensions of the RRT are RRT* [9] and TB-RRT [10]. The major problem with sampling methods is that the algorithm does not know when to terminate its operation, how long the algorithm should try to solve the problem before reporting no solution [11]. This problem could cause long computation time or miss feasible solutions. The proposed algorithm in this paper tries to overcome this problem as well.

We present a novel offline method for finding an optimal solution for the throw motion taking the kinodynamic constraints into account. The key component of the algorithm is parameterizing an analytic trajectory function. We formulate the kinodynamic constraints of the problem in the parameterization space. The formulated set of constraints defines a time-varying subspace of the parameters space. An easy-to-use numerical method is proposed for finding a set of parameters that defines an optimal throw trajectory. The numerical method is a stochastic algorithm where the probability to find a solution, if one exists, is defined.

The paper is organized as follows. Section II defines the motion planning problem. In Section III, we propose a parameterization for the throw motion. Section IV formulates the kinodynamic constraints. The formulated constraints define the feasibility problem, which is solved in Section V. In Section VI, we present simulations and experiments, respectively, of a 3R manipulator in a throw motion.

II. Problem Definition

The equations of motion of a fully-actuated n-joint manipulator are given by

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + G(\mathbf{q}) + \Gamma(\dot{\mathbf{q}}) = \mathbf{u}, \qquad (1)$$

where $\mathbf{q}(t) = [\varphi_1(t)\ \varphi_2(t)\ \varphi_3(t)]^T \in \mathcal{Q}$ is the configuration of the system at time t, $\mathcal{Q} \subseteq \mathbb{R}^n$ is the configuration space of the manipulator and φ_i is the angle of joint i, $\mathbf{u}(t) = [u_1(t)\ u_2(t)\ u_3(t)]^T \in \mathbb{R}^n$ is the torque control vector, M is an $n \times n$ inertia matrix, C is the $n \times n$ the centrifugal and Coriolis matrix, and G is an $n \times 1$ vector of joint torques due to gravitation force. Γ is a vector of Coulomb and viscous friction torques at the joints given by

$$\Gamma(\dot{\mathbf{q}}) = sgn(\dot{\mathbf{q}})\Gamma_c + \dot{\mathbf{q}}\Gamma_v, \qquad (2)$$

where Γ_c and Γ_v are the Coulomb and viscous friction coefficients [12]. Moreover, we define $\mathcal{T} \subseteq \mathbb{R}^m$ to be the task space of the manipulators end-effector, that is, its position and orientation relative to the worlds coordinate frame ($m = 3$ in a planar manipulator and $m = 6$ in a spatial manipulator).

We impose several constraints on the system. First, we impose angle constraints on the arm joints defined by its workspace, e.g., mechanical limitations of the joints. This ensures that the configuration of the arm is within the allowed subspace $\mathcal{Q}_{WS} \subseteq \mathcal{Q}$. Second, obstacles in the workspace limit the motion of the end-effector. Therefore, the end-effectors task is restricted from the subspace $\mathcal{T}_{obs} \subset \mathcal{T}$. We

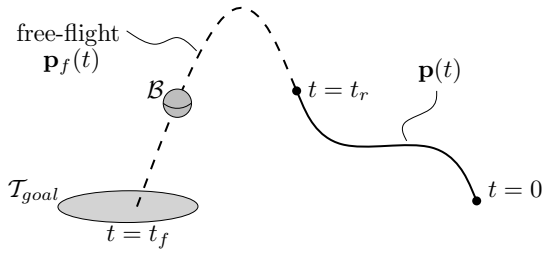

Fig. 1. Scheme of the throw trajectory of the end-effector and free-flight motion of the object.

impose the configuration of the robot to be within the allowed configuration space

$$\mathcal{Q}_{allowed} = \mathcal{Q}_{WS} \backslash (L^{-1} \circ \mathcal{T}_{obs}), \tag{3}$$

where $L : \mathbb{R}^n \to \mathbb{R}^m$ is a transformation from the configuration space to the task space, i.e., direct kinematics. Note that the inverse kinematics $L^{-1}(\cdot)$ can produce several solutions. We take the same particular one at each time. Third, the joint actuators limit the control inputs to the dynamic system in (1). Explicitly, we can write that

$$|\mathbf{u}(t)| \leq \mathbf{u_{max}}, \tag{4}$$

where $\mathbf{u_{max}}$ is the maximum allowed joint torque. Moreover, we add joint velocity constraints such that

$$|\omega(t)| \leq \omega_{\mathbf{max}} \ , \tag{5}$$

where $\omega_{\mathbf{max}}$ is the maximum allowed joint angular velocity. The task of the end-effector is denoted as $\mathbf{p}(t) \in \mathcal{T}$, where $\mathbf{p}(t) = L(\mathbf{q}(t))$ and $\dot{\mathbf{p}}(t) = J\dot{\mathbf{q}}(t)$ with J being the Jacobian matrix of the manipulator.

An object $\mathcal{B}$ is held by the end-effector. The objective of this problem is to generate a throw trajectory $\mathbf{p}(t) \in L \circ \mathcal{Q}_{allowed}$ of the manipulator to toss the object into mid-air such that it will finally reach a goal $\mathcal{T}_{goal} \subset \mathcal{T}$ in the manipulators task space at some finite time $t_f \in [0, \infty)$. Note that the goal task is beyond the reach of the end-effector; that is, $\mathcal{T}_{goal} \cap (L \circ \mathcal{Q}_{allowed}) = \varnothing$.

The above problem is therefore a motion planning problem under kinodynamic constraints. We present a novel stochastic-based algorithm for motion planning of the given manipulator to perform the throw task.

III. Trajectory Parameterization

In this section we present the parameterization of the desired throw trajectory $\mathbf{p}(t) \in \mathcal{T}$. The motion is solely defined by the throw motion from the initial pose at time $t = 0$ to the release pose at time $t = t_r$ (Figure 1). After release of the object, the motion of the object is defined as free-flight and un-controlled.

We propose a method for parameterization of the throw motion. We start with the parameterization of the release moment $t = t_r$. As mentioned, the free-flight trajectory is defined only by the release task $\mathbf{p}(t_r) = \mathbf{p}_r$ and velocity $\dot{\mathbf{p}}(t_r) = \mathbf{v}_r$. However, in order to perform a smooth release, the orientation of the end-effector at time of the release should be defined by the direction of velocity. Therefore, we present the following definition.

Definition 1. *Let the map $Z_p : \mathcal{T} \mapsto \mathbb{R}^{m_1}$ be a projection map of the task (position and orientation) of the end-effector onto its position space ($m_1 = 2$ for a planar robot and $m_1 = 3$ for a spatial robot). And let $Z_o : \mathcal{T} \mapsto \mathbb{R}^{m-m_1}$ be the projection map of the task of the end-effector onto its orientation space.*

Hence, the release moment can be parameterized by the end-effectors position $Z_p(\mathbf{p}_r)$ and task velocity $\mathbf{v}_r \in \mathbb{R}^m$ at time t_r. According to this definition, we constrain the orientation of the end-effector to be co-linear to its linear velocity. That is,

$$\langle \mathbf{v}_{ee}(Z_o(\mathbf{p_r})), \mathbf{v}_r^l \rangle = 1, \tag{6}$$

where $\mathbf{v}_{ee}(\cdot) : \mathbb{R}^{m-m_1} \to \mathbb{R}^{m_1}$ is a vector with direction co-linear to the end-effector's opening and $\mathbf{v}_r^l \in \mathbb{R}^{m-m_1}$ is the linear velocity vector of the end-effector at release time t_r.

Defining the throw initial condition is sufficient to describe the motion of the object after release. However, not all motions of the end-effector can satisfy these conditions, if such exist. The kinodynamic constraints limit the motions that could be performed. Therefore, with the assumption that at the beginning of the motion the manipulator is at rest $\dot{\mathbf{p}}(0) = \mathbf{v_0} = 0$, we parameterize the initial task of the end-effector $\mathbf{p}(0) = \mathbf{p}_0$.

The time duration from time $t = 0$ to the release time t_r could also be used as a parameter. This adds another degree of freedom to the motion such that excessive velocities or torques might not be needed. Therefore, the parameterization of the throw motion is fully defined by the following parameterization vector (P-vector)

$$\sigma = \begin{pmatrix} \mathbf{p}_0^T & Z_p(\mathbf{p}_r)^T & \mathbf{v}_r^T & t_r \end{pmatrix}^T \in \Omega, \tag{7}$$

where $\Omega \subseteq \mathbb{R}^{2m+m_1} \times \mathbb{R}^+$. Vector σ is constructed of the free parameters of the throw motion, and by defining their values we define a single unique motion. A P-vector example for a planar robotic arm is presented next.

Example 1. Given a fully actuated planar robotic arm, its end-effectors task is defined as $\mathbf{p}(t) = [p_x(t) \ p_y(t) \ \theta(t)]^T$, where p_x and p_y are the end-effectors position in the plane and θ is its orientation angle with respect to the x-axis. Therefore, from (7) we acquire the following P-vector:

$$\sigma = \begin{pmatrix} p_{x_0} & p_{y_0} & \theta_0 & p_{x_r} & p_{y_r} & v_{x_r} & v_{y_r} & v_{\theta_r} & t_r \end{pmatrix}^T. \tag{8}$$

We add the co-linearity constraint from (6) as $\theta_r = Atan2(v_{y_r}, v_{x_r})$.

The motion from task $\mathbf{p}_0$ to task $\mathbf{p}_r$ is defined by cubic functions as $\mathbf{p}(t) = M \cdot \mathbf{z}(t)$, where $\mathbf{z}(t) = \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix}^T$ is a vector of monomials and M is an $m \times 4$ matrix of coefficients. Given a P-vector σ, matrix M could be fully defined according to the conditions at initial and release time instances: $\mathbf{p}(0) = \mathbf{p}_0$, $\dot{\mathbf{p}}(0) = 0$, $\mathbf{p}(t_r) =$

$\mathbf{p}_r,\ \dot{\mathbf{p}}(t_r) = \mathbf{v}_r$, where $\mathbf{p}_0, \mathbf{p}_r, \mathbf{v}_r$ are given from vector σ and therefore trajectory $\mathbf{p}(t)$ depends not only on time but on σ as well. Thus, the trajectory should actually be denoted as $\mathbf{p}(t,\sigma) = M(\sigma)\mathbf{z}(t)$. From this, we can formulate the joints position in terms of σ with the inverse kinematics: $\mathbf{q}(t,\sigma) = L^{-1}(\mathbf{p}(t,\sigma))$. It is optional to use a higher order polynomial function such that the additional coefficients are added to the P-vector in (7). This will add more degrees of freedom for defining the motion and perhaps more solutions.

IV. Constraints Formulation

In the previous section we defined the throw trajectory. Next, we formulate the constraints presented in Section II in terms of time and vector σ. Particularly, we formulate the set of constraints G, which defines the feasibility region for the throw motion, and present the set of constraints K that defines whether the object reaches its goal.

Based on the definition of the throw trajectory in the task space, we can re-write the systems constraints presented in Section II. We classify the constraints into two classes: the class derived from the kinodynamic constraints and the class that defines if the current trajectory leads the thrown object to its desired goal task $\mathcal{T}_{goal}$.

We start with the kinodynamic constraints. First, the allowed configuration space $\mathcal{Q}_{WS}$ constraint is rewritten by a_1 analytical inequalities

$$\Theta_k(\mathbf{q}(t,\sigma)) \leq 0,\ \forall k = 1, ..., a_1\ . \tag{9}$$

For example, if we limit the joints maximum angle, (9) can be $\mathbf{q}(t,\sigma) - \theta_e \leq 0$. The same can be done to the task space constraint where $\mathcal{T}_{obs}$ is formulated to a_2 analytical constraints

$$\Lambda_k(\mathbf{p}(t,\sigma)) \leq 0,\ \forall k = 1, ..., a_2\ . \tag{10}$$

The torques constraint in (4) is imposed by the equations of motion such that $\mathbf{u}(t,\sigma)$, which is calculated using (1) with $\mathbf{q} = \mathbf{q}(t,\sigma)$, is limited by an upper boundary $\mathbf{u}_{max}$. Similarly, the angular velocity $\dot{\mathbf{q}}(t,\sigma) = J^{-1}\dot{\mathbf{p}}(t,\sigma)$ is constrained according to (5) by an upper bound ω_{max}.

The above re-formulated inequalities could now be written as

$$G_k(t,\sigma) \leq 0,\ \forall k = 1, ..., a, \tag{11}$$

where G_k is the k^{th} component of

$$G(t,\sigma) = \begin{pmatrix} \Theta(t,\sigma) \\ \Lambda(t,\sigma) \\ |\mathbf{u}(t,\sigma)| - \mathbf{u}_{max} \\ |\dot{\mathbf{q}}(t,\sigma)| - \omega_{max} \end{pmatrix} \tag{12}$$

and $a = a_1 + a_2 + 2n$.

The second kind of constraints are the ones that check if the free-flight trajectory passes through the goal. After release of the object at time t_r, the object has a trajectory of a projectile under gravity. We neglect other forces such as drag. Thus, the objects trajectory is

$$\mathbf{p}_f(t) = -\frac{1}{2}\mathbf{a}_g(t - t_r)^2 + \dot{\mathbf{p}}(t_r)(t - t_r) + \mathbf{p}(t_r), \tag{13}$$

where $\mathbf{a}_g$ is the vector of gravitation. The parameters in trajectory (13) are parameters of σ as defined in (7) and therefore we can rewrite $\mathbf{p}_f(t) = \mathbf{p}_f(t,\sigma)$. We describe the tolerance for reaching the goal by an analytical expression for $\mathcal{T}_{goal}$ in terms of σ. In fact, the goal can have the form of an hyper-rectangle in the task space such as $\mathbf{p}_{goal}^{min} < \mathbf{p}_f(t,\sigma) < \mathbf{p}_{goal}^{max}$ or a sphere, according to need. Moreover, we can formulate a constraint on $\dot{\mathbf{p}}_f(t_f,\sigma)$ to impose a hitting direction of the target. Notice that we constrain not only the position of the object in the target but also its orientation. In general, we can formulate these constraints in the form of

$$K_k(t,\sigma) \leq 0,\ \forall k = 1, ..., b\ . \tag{14}$$

Hence, a trajectory parameterized by σ hits the target if at some time $t_g > t_r$ inequality (14) is satisfied. Inequality (14) could easily be solved analytically for $t = t_f$ and if $t_f > t_r(\sigma)$, then this trajectory hits the target.

The inequalities in (11) and (14) define the feasible region of the dynamic system to perform the throw task in terms of time and the desired trajectory represented by the P-vector σ. That is, we obtained a set of analytical constraints that defines a time-varying region in Ω-space.

V. Search algorithm

In previous section we have defined the constraints of the problem as a time-varying subspace of Ω-space. A solution for the above problem is a feasible and optimal vector $\sigma^* \in \Omega$.

A. The feasibility problem

First we define the feasibility set. Let $\Sigma \subset \Omega$ be a user defined allowed region for σ. The range of set Σ is chosen according to the workspace of the robotic arm. The range for the time parameter in Σ is chosen according to the allowed time frame for the motion.

Definition 2. *A set $\Omega_f \subset \Omega$ is a feasible set in $t \in [0, t_r]$ if $\Omega_f \subseteq \Sigma$ and all $\sigma \in \Omega_f$ satisfy inequality* (14) *at some time $t_f > t_r$ and inequality* (11) *for any $t \in [0, t_r]$.*

That is, a vector $\sigma \in \Omega$ is in the feasibility set Ω_f if it is in Σ, it satisfies $G_k(t,\sigma) \leq 0$ for all $k = 1, ..., a$, $t \in [0, t_r(\sigma)]$, and satisfies $K_k(t,\sigma)$ for all $k = 1, ..., b$ at some time $t_f > t_r(\sigma)$. The release time value t_r is a function of σ and is taken from its last component as defined in (7). The definition of the feasibility set is illustrated in Figure 2 with a two-dimensional abstraction of the σ-space. Now, the goal is to find an optimal vector σ^* in the feasibility set Ω_f. Formally, the feasibility problem is as follows:

Problem 1. *Find the vector $\sigma^* \in \Omega_f$ such that*

$$\begin{aligned} \sigma^* = arg\min_{\sigma} \quad & H(\sigma) \\ \textit{subject to} \quad & \sigma^* \in \Omega_f, \end{aligned} \tag{15}$$

where $H(\sigma)$ is some cost function to minimize.

In other words, the above general problem is finding an optimal vector σ^* that is feasible and minimizes some cost

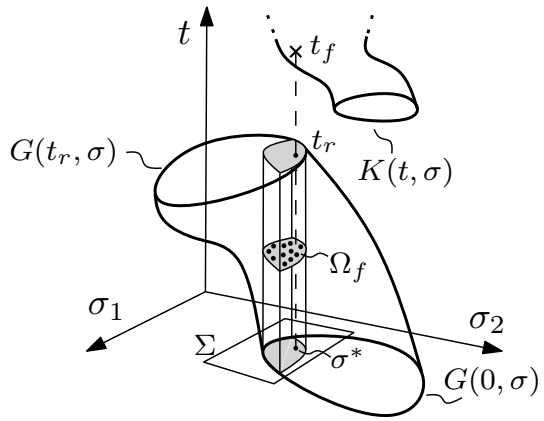

Fig. 2. A 3D illustration of the time-varying subspace defined by $G(t,\sigma)$.

function $H(\sigma)$ to be determined. Examples for such criteria are a minimal time motion or distance maximization from the constraints boundary. In the next subsection we present an algorithm for finding the feasibility set and choosing an optimal solution from it. The probability to find a solution if one exists is also presented.

B. Feasibility search algorithm

A search algorithm is now presented to find a set Ω_f of feasible vectors. The domain formed by the set of constraints in inequality (11) is non-linear, non-convex, and not continuous due to composition of several constraints. Therefore, an analytical solution of the reachable set is only possible in rare and simple instances. We present a numerical search algorithm to find a set of P-vectors satisfying the above constraints. Further, we can choose one vector from the set that best minimizes the cost function.

Algorithm 1 Throw trajectory search algorithm

Input: Sets of constraints G, K, probability P_{max}, tolerance ε_b, and allowed subset $\Sigma \subseteq \Omega$.
Output: Optimal solution σ^*.

```
Calculate number of random points N such that the probability to find a solution is more than 1 − P_max.
Generate the set P = {σ_1, ..., σ_N} of N uniformly distributed random P-vectors within Σ.
for i = 1 → N do
    if ¬(Is_feasibile(σ_i, G, K, ε_b)) then
        Remove σ_i from P.
    end if
end for
Select σ* ∈ P which best minimizes cost function H(σ).
return σ*.
```

The basis of the algorithm's operation is generating a set of N random points within the user-defined subspace Σ and checking each for its feasibility. The throw trajectory search algorithm is presented in Algorithm 1. The algorithm's input is the allowed set Σ in Ω determined by the user and the set of constraints of inequalities (11) and (14). The first step of the algorithm is to determine the number of random P-vectors N such that the probability to find a solution is more than a user defined probability $1 - P_{max}$. The calculation of N such that the probability not to find a solution if one exists is P_{max} can be shown to be

$$N \geq -\frac{1}{\rho}\log(P_{max}) \ , \tag{16}$$

where ρ is a user-defined value representing the allowed ratio between the volume of Ω_f and the volume of the pre-defined region Σ. The determination of ρ is a resolution of how close we allow the desired P-vector to be to the constraints boundaries.

The next step is to sample N random points $\mathcal{P} = \{\sigma_1, ..., \sigma_N\}$ uniformly distributed in Σ. The allowed region formed by Σ is a hyper-rectangle in Ω, and therefore we sample points in each axis of Ω within the boundaries defined by Σ. Such sampling provides a Poisson distribution over the volume of Σ. The next step is going over all the N points in $\mathcal{P}$ and filtering out those that are not feasible. Such an operation is conducted by function $Is\_feasible$ in Algorithm 2. The release time t_{r_i} is determined for each point σ_i checked. First, we check if the trajectory of σ_i forms a free-flight trajectory that hits the target. This is done by using (14) for $t = t_{f_i}$ and checking if $t_{f_i} > t_{r_i}$. If not, this P-vector is rejected. Next, we check if the constraints are satisfied for a discrete time $t_i = \{0, \Delta t_1, \Delta t_1 + \Delta t_2, ..., t_{r_i}\}$. Those that do not satisfy the constraints are eliminated from set $\mathcal{P}$. However, scanning the constraint $G(t_i, \sigma_i)$ for

Algorithm 2 $Is\_feasibile(\sigma_i, G, K, \varepsilon_b)$

Input: σ_i, the sets of constraints G, K and the tolerance ε_b.
Output: Boolean: **True** if σ_i is feasible, and **False** if not.

```
Extract t_{r_i} from the last component of σ_i.
Calculate t_{f_i} from K according to (14).
if ¬( t_{f_i} > t_{r_i} ) then
    return False.
end if
Calculate S_max. // using optimization prob. in (19).
while ( t ≤ t_{r_i} ) do
    Calculate G̃_{i,t} = max_j {G_j(t, σ_i)}.
    if ¬( G̃_{i,t} < ε_b ) then
        Return False.
    else
        Calculate Δt = −G̃_{i,t} / S_max.
        t = t + Δt.
    end if
end while
Return True.
```

$t_i = \{0, \Delta t, 2\Delta t, ..., t_{g_i}\}$ where Δt is a constant value is rather risky. With rather large step size Δt, the value of G might ascend over 0 and descend below again within the discretized step size. On the other hand, too small step sizes could be unnecessary and result in an overly long runtime. Therefore, we propose simple adaptive step size methods

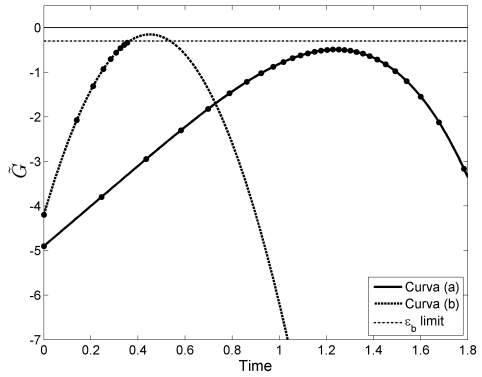

Fig. 3. Adaptive check of two curves; In curve (a) a smaller step size is taken as the curve approaches the zero line. Curve (b) is invalid as the discretized points ascend above the ε_b limit.

to fine-tune the time steps and diagnose or rule out such scenarios. First we define the maximal constraint value.

Definition 3. *The constraint value of a feasible point σ_i at time t_i is defined to be*

$$\tilde{G}_{i,t_i} = \max_j \{G_j(t_i, \sigma_i)\} \ , \tag{17}$$

where G_j is the j^{th} component of the constraint vector G.

That is, the constraint value is the shortest distance from point σ_i to the boundary of the closest constraint at all time steps $t_i = [0, t_{g_i}]$. Assume that the change rate of the constraint value is bounded by

$$\frac{\Delta \tilde{G}_{i,t_i}}{\Delta t_i} \leq S_{max}, \ \forall t_i \in [0, t_{g_i}] \ . \tag{18}$$

That is, the maximum slope of the constraint value $\tilde{G}_{i,t_i}$ is S_{max}. Under this assumption we can say that if at time t_i the constraints are satisfied, $\tilde{G}_{i,t_i} < 0$, then the minimum time for the constraint to reach 0 is $t_i + \Delta t_{min}$, where $\Delta t_{min} = -\frac{\tilde{G}_{i,t_i}}{S_{max}}$. Therefore, as we get closer to a boundary of a constraint, we decrease Δt such that reaching above the zero line in that time frame is not possible. Figure 3 illustrates the selection of Δt to be smaller as it approaches the zero line and larger when receding from it. However, in this adaptive approach, even though $\tilde{G}_{i,t_i}$ passes the zero line, the algorithm will never do so as it will continue to decrease Δt. Therefore, we bound such that the algorithm will stop checking the current σ_i (and remove it) if $\varepsilon_b < \tilde{G}_{i,t_i} < 0$. The tolerance ε_b is user-defined with the same considerations previously discussed for choosing ρ. This also serves as a safety distance, assuring the solution is far enough from the constraints boundaries. To calculate S_{max} we differentiate the constraint vector by time to acquire its slope $\frac{\partial G(t,\sigma)}{\partial t}$. S_{max} is the maximum slope of all components over all time and can be computed by the following maximization problem

$$\begin{aligned} S_{max} &= \max_{t,\sigma,j} S_j(t, \sigma) \\ \text{subject to} \ \ & 0 \leq t \leq \tilde{t}_r, \ \sigma \in \Sigma, \end{aligned} \tag{19}$$

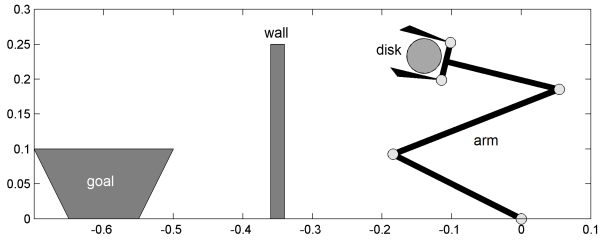

Fig. 4. The throw goal.

where S_j is the j^{th} component of the constraints derivative $S(t, \sigma) = \frac{\partial G(t,\sigma)}{\partial t}$ and $\tilde{t}_r$ is the maximum possible goal time based on the allowed time interval given in Σ. Problem (19) could be computed analytically using Kuhn-Tucker conditions or numerically.

We have acquired the feasibility set $\mathcal{P}$, which is the discrete representation of Ω_f. The Final step of the algorithm is to select the vector $\sigma^* \in \mathcal{P}$ that best minimizes the cost function $H(\sigma)$. The optimal vector acquired is in fact the joint trajectory $\mathbf{q}(t, \sigma^*)$, which satisfies the kinodynamic constraints and minimizes the cost function. From here we need to apply a non-linear control method to follow the computed trajectory.

VI. Simulations & Experiments

In this section we will demonstrate the algorithm's operation with simulations and experiments on a planar 3R robotic arm. For simulations of the proposed method, the algorithm was implemented in Matlab[1] on an Intel-Core i7-2620M 2.7GHz laptop computer with 8GB of RAM. The simulations were conducted with a dynamic model of the arm.

The aim of the simulation is to throw a planar disk into a goal basket where a wall prevents direct access to the basket as seen in Figure 4. The P-vector of this throw problem is the one defined in (8). The physical limitations of the arm are defined such that $-150^o \leq \phi(t) \leq 150^o$ and $-2[Nm] \leq \mathbf{u}(t) \leq 2[Nm]$. These limits along with the workspace limitations were used to formulate the constraints in the form of (11). The goal constraint of (14) is defined by the $x - y$ borders of the goal basket.

Based on the lengths of the links and joint capabilities, we define set Σ such that the generated random σ's are within a hyper-rectangle defined by $\sigma_{max} = [0.6m \ 0.6m \ 2\pi \ 0.6m \ 0.6m \ 1m/s \ 1m/s \ 4rad/s \ 3s]$ and $\sigma_{min} = [-0.6m \ 0m \ 0 \ -0.6m \ 0m \ -1m/s \ -1m/s \ -4rad/s \ 0s]$. We allow maximum motion time of $t_{max} = 3s$. We arbitrarily chose the probability not to find a solution to be $P_{max} = 2 \times 10^{-10}$ and the allowed ratio to be $\rho = 9 \times 10^{-4}$. Hence, according to (16) the minimum number of points was selected and is $N = 24,814$. With these terms and constraints, after runtime of 0.48 seconds, 21 feasible solutions were found. The optimal solution with a minimum time cost function was selected. The planned

[1]Matlab is a registered trademark of The Mathworks, Inc.

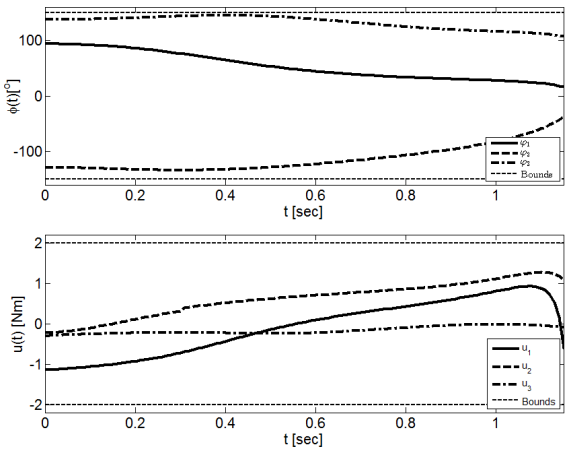

Fig. 5. The trajectory (top) and torque signal (bottom) to the arm's joints to throw the disk into the goal.

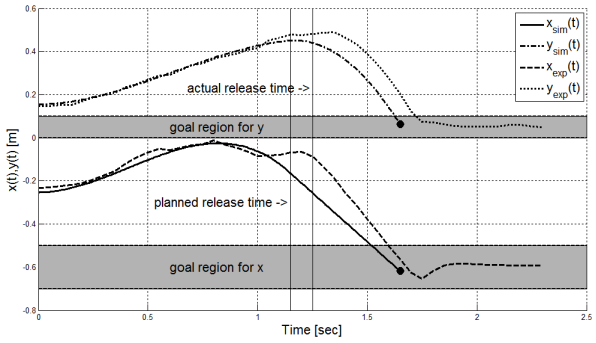

Fig. 6. The planned trajectory of the disk to reach the goal and the measured one in the experiments.

arm's trajectory and control signal are presented in Figure 5 where satisfaction of the physical constraints can be seen.

An experimental setup was designed and built composed of a planar robotic arm with three MX-106 Dynamixel actuators. The arm moves parallel to a 20^o inclined air hockey table so that the thrown object can slide with minimum friction. A motion capture system (by OptiTrack) was set to track the moving object's trajectory. The object's planned and measured motion is shown in Figure 6. Due to joint inaccuracies and backlashes, the motion is not exactly as planned but the object reaches the goal as planned. Snapshots of the throw experiment are shown in Figure 7.

VII. Conclusions

A novel algorithm was presented for throw motion planning under kinodynamic constraints. The key component of the algorithm is the parameterization of the throw motion into a high-dimensional vector. The algorithm generates random P-vectors such that the probability to find a solution if one exists is $1 - P_{max}$. Moreover, an adaptive step size search algorithm was presented to select the feasible P-vectors satisfying the kinodynamic constraints.

The runtime of the algorithm is very short and enables fast and efficient computation. Moreover, it provides future research in implementation of the algorithm to real-time

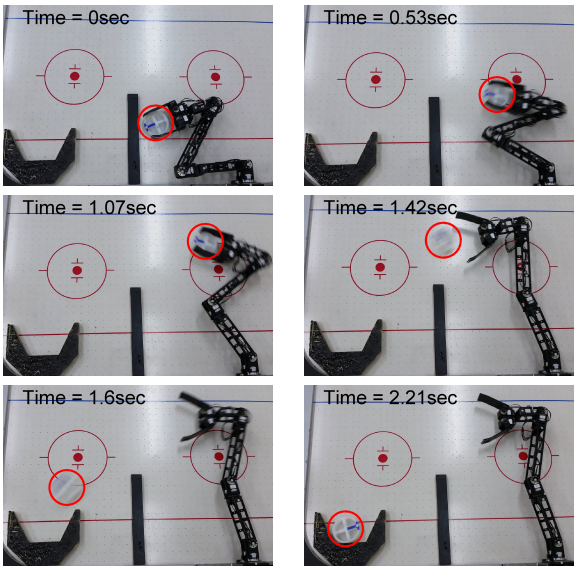

Fig. 7. Snapshots of the throw experiment.

motion planning problems. It can be shown that the overall complexity of the algorithm is $O\left(-\frac{t_{max}S_{max}}{\varepsilon_b}N\right)$.

Acknowledgements

The research was partially supported by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University of the Negev.

References

[1] H. Miyashita, T. Yamawaki, and M. Yashima, "Control for throwing manipulation by one joint robot," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2009, pp. 1273–1278.

[2] K. Lynch and M. T. Mason, "Dynamic nonprehensile manipulation: Controllability, planning and experiments," *International Journal of Robotics Research*, vol. 18, no. 1, pp. 64–92, January 1999.

[3] J. Kim, "Motion planning of optimal throw for whole-body humanoid," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, Dec 2010, pp. 21–26.

[4] U. Mettin, A. Shiriaev, L. Freidovich, and M. Sampei, "Optimal ball pitching with an underactuated model of a human arm," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA),*, May 2010, pp. 5009–5014.

[5] Y. Zhang, J. Luo, and K. Hauser, "Sampling-based motion planning with dynamic intermediate state objectives: Application to throwing," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2012, pp. 2551–2556.

[6] J. Canny, *The Complexity of Robot Motion Planning*, ser. ACM doctoral dissertation award. MIT Press, 1988.

[7] L. Kavraki, P. Svestka, J.-C. Latombe, and M. 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.

[8] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[9] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *CoRR*, vol. abs/1005.0416, 2010.

[10] A. Sintov and A. Shapiro, "Time-based RRT algorithm for rendezvous planning of two dynamic systems," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2014.

[11] K. Hauser, "Motion planning for legged and humanoid robots," Ph.D. dissertation, Stanford, CA, USA, 2008.

[12] B. Bona and M. Indri, "Friction compensation in robotics: an overview," in *Proceedings of the IEEE Conference on Decision and Control*, Dec 2005, pp. 4360–4367.