2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Time-Based RRT Algorithm for Rendezvous Planning of Two Dynamic Systems

Avishai Sintov and Amir Shapiro

Abstract— **The work presented in this paper proposes a new method for time based motion planning of a dynamic system to reach a dynamical goal within a specified time. The method enables trajectory planning based on the dynamics of the system with time constraint. Moreover, this method allows rendezvous planning of two dynamic systems where only one is controlled. To reach this objective, we introduce a new concept termed Time-Based RRT (TB-RRT) which is an extended version of the Rapidly-exploring Random Tree (RRT). The concept of the TB-RRT is to add time parameters to the nodes in the tree such that each node denotes a specific state in a specific time. The algorithm was implemented in two applications to demonstrate the approach and validate its feasibility; The first application is a one degree of freedom bat hitting a ball and the second application is a three degrees of freedom manipulator catching a moving object. Simulation results show the system accurately following the planned trajectory and the robot catching the object in time.**

I. Introduction

Many applications in robotics demand not only motion planning of a feasible trajectory for a robot to move from one state (configuration and velocity) to another, but demand also that it should be completed within a specific time. Such applications range from industrial and civil areas to military ones. In industrial production lines, robotic arms could regrasp parts by tossing them in the air and catching them in a desired configuration. Flying drones could be catched in air instead of performing landing to avoid structure damage due to the landing shocks. Performing car passing on road by an autonomous car should be done within a set time range to avoid collision. All of these applications and more demand that the trajectory will take into account another uncontrolled dynamic system (referred in this paper as the dynamic goal) in a specific time.

To deal with the presented problem, two main aspects are to be considered; the first is the time-based trajectory planning to account for the dynamic goal and the second is the tracking problem of the uncontrolled dynamic goal to estimate its trajectory. In this work we focus on the first aspect and assume full knowledge of the goals trajectory, that is, accurate measurement and prediction of the objects position and velocity at all times.

The trajectory planning problem has been widely researched. The most common methods for time based trajectory planning are the point to point polynomial trajectories and cubic splines [1]. However, these methods do not take the dynamics and constraints of the system into account. Potential fields are also a known method for path planning [2]. However, specific time constraints are not taken into account. Many other works presents trajectory planning under cost minimization such as jerk [3] or time [4]. Another method is the Rapidly-exploring Random Tree (RRT) [5] which is a probabilistic method for trajectory planning in a complex environment taking the dynamics of the system into account. The RRT approach is the base approach of this work.

Another problem dealt in this paper is rendezvous planning of two dynamic systems where only one is controlled. Many solutions to this problem were presented; The proportional navigation algorithm [6], [7] is a commonly used method to account for a moving object such as a UAV or a missile. The algorithm is shown to be complete as it will intercept the object in finite time, however not in a specific desired time. Many other rendezvous algorithms for optimal trajectories were presented [8], [9] to find the best trajectory in a dynamic environment.

In the presented work we wish to address two main problems. The first problem is trajectory planning to reach a goal in a specific desired time. To solve this problem we present the Time-Based RRT (TB-RRT) method which incorporates the advantages of the known RRT with time constraint. The other problem is the rendezvous planning of a controlled dynamic system with an uncontrolled one. To solve the second problem we use the TB-RRT and sample the goals trajectory into the trees space.

The paper is organized as follows. Section II defines the problem discussed in this paper. Section III reviews the RRT algorithm. In Section IV we present our algorithm for finding a trajectory to a dynamic goal using a time based RRT algorithm. In Section V we present some applications for the method with two simulations of a bat hitting a ball and a manipulator catching a moving object. We conclude and suggest some future work in Section VI.

II. Problem Formulation

This section defines the motion planning problem. Given a dynamic system with the non-linear differential constraint

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{1}$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ and $\mathbf{u}(t) \in \mathbb{R}^m$ are the state and control inputs of the system, respectively, at time t. The initial state at $t = 0$ is given by $\mathbf{x}(0) = \mathbf{x_o}$. A set of constraints on the states $\mathbf{x}(t)$ are imposed to avoid obstacles and physical limits of the system. $\mathcal{X}_{free} \subset \mathcal{X}$ defines the subset of states within a metric space $\mathcal{X}$, free from obstacles and system's kinematic and dynamic limitations. The control inputs are bounded according to the limits of the systems actuators $|\mathbf{u}(t)| < \mathbf{u_{max}}$.

Another uncontrollable non-linear dynamic system is given

978-1-4799-3684-7/14/$31.00 ©2014 IEEE

which represents a moving goal to be reached. Its system is given by

$$\dot{\mathbf{q}}(t) = g(\mathbf{q}(t)) \tag{2}$$

where $\mathbf{q}(t) \in \mathbb{R}^n$ is the state of the system at time t. Its initial state is $\mathbf{q}(0) = \mathbf{q}_o$. The state of the goal does not have to be in $\mathbb{R}^n$ but should be transformed to it for compatibility reasons. The objective is to reach the dynamic goal, that is, the time-varying state $\mathbf{q}(t)$. The objective would be completed if at some time t_f the condition

$$t_f = \{\ t_f \in [0,\infty) \mid \mathbf{x}(t) \in \mathcal{X}_{free}\ \forall [0,t_f] \\ and\ \ \rho(\mathbf{x}(t_f), \mathbf{q}(t_f)) < \epsilon\ \} \tag{3}$$

is met. Where ρ is some metric criterion to be defined later and $\epsilon > 0$ is a predefined tolerance. Therefore, the motion planning problem is defined as follows. Given the initial state $\mathbf{x_o}$, the state constraints $\mathcal{X}_{free}$ and the control input limit $\mathbf{u_{max}}$, compute the control input sequence $\mathbf{u}(t), t \in [0, t_f]$ to form a trajectory $\mathbf{x}(t)$ for the dynamic system to reach the dynamic goal at some finite time $t_f \in [0,\infty)$.

III. RRT Background

The basic RRT algorithm was firstly introduced by LaValle [5] as a stochastic approach for trajectory planning to reach a goal state from an initial state under a differential constraint (such as in eq. (1)). The advantage of this method is in its simplicity of applying multiple constraints and avoiding obstacles.

The basics of the simple RRT algorithm is by sampling random states and extending the RRT toward them. In every iteration k, a random state $\mathbf{x_{rand}} \in \mathcal{X}$ is sampled. Then, a nearest-neighbor search finds the closest state $\mathbf{x_k}$ in the tree to the random point according to a defined metric ρ. By applying an input $|\mathbf{u_k}| < \mathbf{u_{max}}$ for small time Δt, an incremental motion is performed toward $\mathbf{x_{rand}}$ based on eq. (1) to a new state

$$\mathbf{x_{k+1}} = \mathbf{x_k} + f(\mathbf{x_k}, \mathbf{u_k})\Delta t \tag{4}$$

where Δt is the time interval used for discretization and $\mathbf{x_k}$ is the state at time $t = k\Delta t$. The differential constraint is discretized and integrated here with a first-order approximation. For better accuracy one can use an higher order method such as the fourth-order Runge-Kutta. The new state $\mathbf{x_{k+1}}$ is added to the tree only if $\mathbf{x_{k+1}} \in \mathcal{X}_{free}$. The goal state $\mathbf{q}_g$ is reached if $\rho(\mathbf{q}_g, \mathbf{x_k}) < \epsilon$ for a pre-defined small $\epsilon > 0$. The euclidean metric is the common one used for ρ. The iterations could be terminated under two approaches; An upper bound on the number of iterations will terminate the algorithm and the best solution be taken based on some optimality criteria. The second approach is a greedy one where the first solution found is taken. According to LaValle, the exploration rapidly and uniformly covers the space and converge in a finite time. The state space and the RRT graph of a simple Pendulum are presented in Figure 1 where a trajectory from the down position to the upright position is shown.

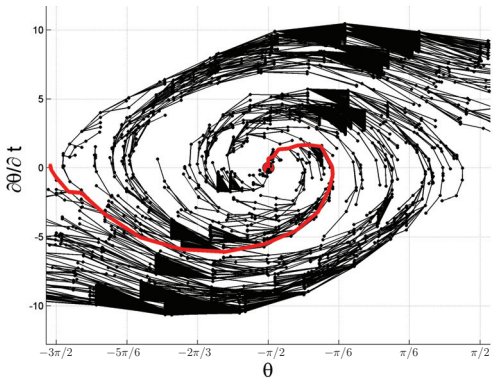

Fig. 1. A RRT graph of a pendulum and a trajectory from down position ($-\pi/2$ radians) to the upright position.

IV. Dynamic Motion Planning

In the previous section we presented the basic RRT approach for kinodynamic planning. We now present the Time-Based RRT (TB-RRT) to impose a time constraint for reaching a dynamic goal.

A. State-Time Space

We expand the notion of the state space to contain varying time. That is, we define the *State-Time Space* to be a space where a time coordinate is added to the state coordinates of the system. Therefore, a state-time vector is constructed as

$$\tilde{\mathbf{x}} = \begin{bmatrix}\mathbf{x}^T & t\end{bmatrix}^T \tag{5}$$

where $\tilde{\mathbf{x}} \in \mathbb{R}^{n+1}$. A state-time vector indicates the configuration and velocity of a system at time $t \in [0,\infty)$. The state-time space is now denoted as $\hat{\mathcal{X}} \subset \mathbb{R}^{n+1}$.

With the new state-time vector we can now modify eq. (4) to be

$$\tilde{\mathbf{x}}_{\mathbf{k+1}} = \tilde{\mathbf{x}}_{\mathbf{k}} + \begin{pmatrix} f(\tilde{\mathbf{x}}_{\mathbf{k}}, \mathbf{u_k}) \\ 1 \end{pmatrix} \Delta t \quad . \tag{6}$$

That is, the next node $\tilde{\mathbf{x}}_{\mathbf{k+1}}$ in the tree is updated to its new time by adding Δt relative to its parent node $\hat{\mathbf{x}}_{\mathbf{k}}$. Moreover, we constrain the new node in the tree to be in the allowed state-time space, i.e., $\tilde{\mathbf{x}}_{\mathbf{k+1}} \in \tilde{\mathcal{X}}_{free}$. The region of $\tilde{\mathcal{X}}_{free} \subset \tilde{\mathcal{X}}$ is acquired straight forward from $\mathcal{X}_{free}$ with additional limitations on the time. Meaning, we can limit the nodes in the tree to a time frame. Moreover, $\tilde{\mathcal{X}}_{free}$ could be used to model dynamic obstacles as it varies in time.

This representation of a state-time vector enables growing a tree where each node has knowledge of its state at a specific time. Therefore, a goal state-time could be defined and a trajectory could be planned by growing a tree, searching for a node close enough to the goal state-time. A node is considered to be close enough to a goal state if a predefined metric ρ^* (to be defined) is smaller than a specified overall tolerance. The tolerance must incorporate an allowed time tolerance. Such method enables trajectory planning reaching a goal in a desired time with the time tolerance accuracy.

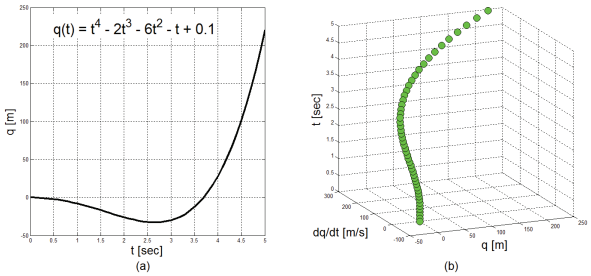

Fig. 2. Example of (a) a one degree of freedom motion by time and (b) its representation in state-time space.

B. State-Time of the Dynamic Goal

We now need to refer to the matter of the dynamic goal. As mentioned, we assume full knowledge of the goals dynamics. That is, $\mathbf{q}(t) \in \mathcal{X}\ \forall t \in [0, t_g]$ could be fully predicted. The basic idea is to sample the goals trajectory in Δt_{goal} time intervals and acquire the goal set $\tilde{\mathcal{X}}_{goal}$. However, we have no need for samples which are not in the allowed space $\tilde{\mathcal{X}}_{free}$. Therefore, we sample $\mathbf{q}(t)$ to form

$$\tilde{\mathcal{X}}_{goal} = \{\tilde{\mathbf{q}}_1, ..., \tilde{\mathbf{q}}_z\} \subset \tilde{\mathcal{X}}_{free} \tag{7}$$

where a sampled state-time vector i is calculated to be

$$\tilde{\mathbf{q}}_i = \begin{bmatrix} \mathbf{q}(i \cdot \Delta t_{goal}) \\ i \cdot \Delta t_{goal} \end{bmatrix} \tag{8}$$

and $z = 1, 2, ..., \lfloor \frac{t_g}{\Delta t_{goal}} \rfloor$. For convenience, in this work we choose the systems time intervals Δt to be equal to the goals time intervals Δt_{goal}. Figure 2 shows an example for sampling of a one degree of freedom motion by time $q(t)$ (Figure 2(a)) to the state-time space $\tilde{\mathcal{X}}_{goal}$ (Figure 2(b)).

C. Time-Based RRT

In the previous subsections we have defined the state-time space and the sampling strategy for the dynamic goal. We present how the TB-RRT is grown. The algorithm proposed is a greedy one which stops once a solution is found. However, it could be expanded to find an optimized one by finding all solutions in K iterations and choosing the one which best minimizes some optimization criteria.

Algorithm 1 presents the main steps in expansion of the TB-RRT and searching for a trajectory from initial state-time $\tilde{\mathbf{x}}_\mathbf{o}$ to any goal state-time $\tilde{\mathbf{q}}_\mathbf{i} \in \tilde{\mathcal{X}}_{goal}$. Similar to the original RRT algorithm, the tree $\mathcal{T}$ is initialized with the start state-time (step 1). The FOR loop (steps 2-15) is set to run a predefined number of iterations K. This enables the user to limit the run in terms of runtime or memory usage. However, the FOR loop could be replaced by an infinite loop as there is a stoping condition to terminate the run once a solution is found. The algorithm samples a random state-time vector $\tilde{\mathbf{x}}_\mathbf{rand}$ (step 3) within $\tilde{\mathcal{X}}$ and finds its nearest node $\tilde{\mathbf{x}}_\mathbf{i} \in \mathcal{T}$ to it (step 4). An input $\mathbf{u}_\mathbf{i}$ is selected to impose motion from $\tilde{\mathbf{x}}_\mathbf{i}$ towards $\tilde{\mathbf{x}}_\mathbf{rand}$ (step 5). That is, by applying the chosen input $\mathbf{u}_\mathbf{i}$ on the differential constraint of eq.(6), we obtain a new node $\tilde{\mathbf{x}}_{\mathbf{i+1}}$. The control input $\mathbf{u}_\mathbf{i}$ will be chosen such that $\tilde{\mathbf{x}}_{\mathbf{i+1}} \in \tilde{\mathcal{X}}_{free}$. Now, we have to check if moving in this direction and magnitude is feasible (steps 7-14). Therefore, we check if the new node $\tilde{\mathbf{x}}_{\mathbf{i+1}}$ is in $\tilde{\mathcal{X}}_{free}$. If true, it adds the new node to the tree and stores connection between $\tilde{\mathbf{x}}_\mathbf{i}$ to $\tilde{\mathbf{x}}_{\mathbf{i+1}}$ with control input $\mathbf{u}_\mathbf{i}$. If the new node is not feasible, the algorithm continues to a new iteration.

Algorithm 1 Search for trajectory to dynamic goal

Input: $\tilde{\mathbf{x}}_\mathbf{o}$, $\tilde{\mathcal{X}}_{free}$ and $\tilde{\mathcal{X}}_{goal}$.
Output: Trajectory and control inputs from $\tilde{\mathbf{x}}_\mathbf{o}$ to $\tilde{\mathbf{x}}_\mathbf{goal}(t)$.

1: Initialize tree $\mathcal{T}$ with $\tilde{\mathbf{x}}_\mathbf{o}$.
2: **for** $k = 1 \rightarrow K$ **do**
3: $\quad\tilde{\mathbf{x}}_\mathbf{rand} \leftarrow$ Random state-time vector.
4: $\quad\tilde{\mathbf{x}}_\mathbf{i} \in \mathcal{T} \leftarrow Nearest\_Neighbor(\mathcal{T}, \tilde{\mathbf{x}}_\mathbf{rand})$.
5: $\quad$Select $\mathbf{u}_\mathbf{i}$ to move from $\tilde{\mathbf{x}}_\mathbf{i}$ towards $\tilde{\mathbf{x}}_\mathbf{rand}$.
6: $\quad\tilde{\mathbf{x}}_{\mathbf{i+1}} \leftarrow \tilde{\mathbf{x}}_\mathbf{i} + [f(\tilde{\mathbf{x}}_\mathbf{i}, \mathbf{u}_\mathbf{i}) \quad 1]^T \Delta t$.
7: $\quad$**if** $\tilde{\mathbf{x}}_{\mathbf{i+1}} \in \tilde{\mathcal{X}}_{free}$ **then**
8: $\quad\quad$Add $\tilde{\mathbf{x}}_{\mathbf{i+1}}$ to $\mathcal{T}$.
9: $\quad\quad$Add edge $\tilde{\mathbf{x}}_\mathbf{i} \xrightarrow{\mathbf{u}_\mathbf{i}} \tilde{\mathbf{x}}_{\mathbf{i+1}}$ to $\mathcal{T}$.
10: $\quad\quad$**if** $\rho^*(\tilde{\mathbf{x}}_{\mathbf{i+1}}, \tilde{\mathcal{X}}_{goal}) < \epsilon$ **then**
11: $\quad\quad\quad$Calculate path $\mathcal{P}$ in $\mathcal{T}$ from $\tilde{\mathbf{x}}_\mathbf{o}$ to $\tilde{\mathbf{x}}_{\mathbf{i+1}}$.
12: $\quad\quad\quad$**return** $\mathcal{P}$.
13: $\quad\quad$**end if**
14: $\quad$**end if**
15: **end for**

The last step of the algorithm is to check whether the new node $\tilde{\mathbf{x}}_{\mathbf{i+1}}$ is close enough to a state-time vector in $\tilde{\mathcal{X}}_{goal}$ to be considered as the same. Most implementation of the RRT use the simple Euclidean metric between two vectors. However, such metric does not take into account the unit difference in the state-time vector components. It is problematic to treat angle units and time units with the same weight. Therefore, our criterion for two vectors to be considered to the same is if they are both within an hyper-rectangle (in $n+1$ dimension). The size of the hyper-rectangle is defined by the allowed tolerance vector ϵ. Each components unit in ϵ respectively correspond to the state-time vector $\hat{\mathbf{x}}$. We define our metric function ρ^* used in this algorithm to implement this by

$$\rho^*(\hat{\mathbf{x}}_{\mathbf{i+1}}, \tilde{\mathcal{X}}_{goal}) = |\tilde{\mathbf{x}}_{\mathbf{i+1}} - \tilde{\mathbf{q}}_\xi| \tag{9}$$

where $\tilde{\mathbf{q}}_\xi = Nearest\_Neighbor(\tilde{\mathcal{X}}_{goal}, \tilde{\mathbf{x}}_{\mathbf{i+1}})$. That is, we find the nearest point in $\tilde{\mathcal{X}}_{goal}$ to the new point and check if they are both in the same hyper-rectangle with edge lengths of ϵ (step 10).

If the above condition is true, a node in the tree was found which is close enough to a state-time vector in $\tilde{\mathcal{X}}_{goal}$. Therefore, the final step will be to find a path in the tree from its root (the initial state-time) to the goal node (step 11). This could be done with any tree search algorithms such as A^*, *DFS*, *BFS*, etc [10].

Figure 3 presents a TB-RRT of a pendulum and its trajectory toward a static goal. A funnel shape tree could be seen starting from the initial state-time. For each node, its siblings could only be in direction where time increases. That is, the tree is constructed such that the system can reach many states in the tree and time will always increase. The box at the goal state-time illustrates the allowed tolerance of the algorithm.

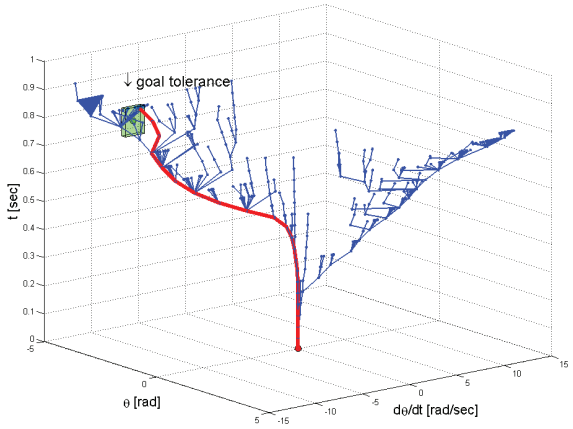

Fig. 3. A TB-RRT of a Pendulum and a trajectory to the goal state-time. The box represents the allowed goal region $\mathcal{X}_{goal}$.

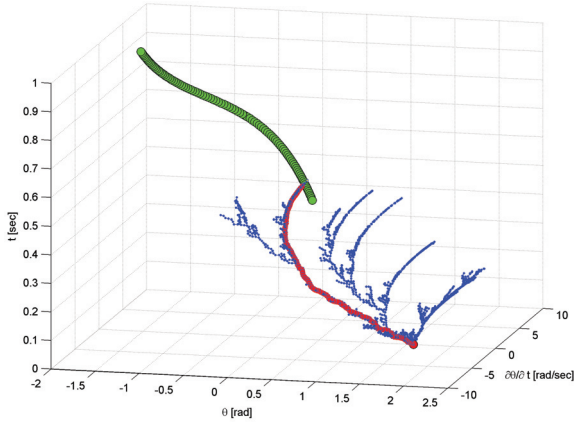

Fig. 4. The TB-RRT of a one degree bat hitting a ball. The green circles are the balls state-time vectors and the red path is the trajectory from the initial state-time to a goal state-time.

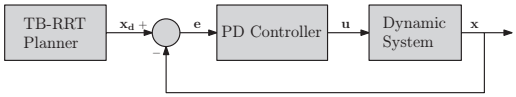

Fig. 5. The PD control scheme. The TB-RRT planner outputs the desired trajectory.

Once a node of the tree is positioned in the box, the algorithm terminates and the trajectory is calculated from initial state time to final state-time node. The thick red line indicates for the path in the tree.
The next section provides some example applications for the algorithm and presents simulations validating the method.

V. Applications and Simulations

In this section we present two applications and simulations of the method: the first is a simple one degree of freedom bat hitting a moving ball, the second is a motion planning of a three degrees of freedom manipulator to catch a moving object. A performance analysis is then presented.

A. One degree bat hitting a ball

The first simulation we conducted was applying the TB-RRT method to a one degree of freedom bat rotating on an horizontal plane. The bat with mass $m = 2kg$ and length $l = 0.2m$ has the following equation of motion:

$$\frac{1}{3}ml^2\ddot{\theta} = u \quad . \tag{10}$$

where the the bats angle is limited to be $0 \leq \theta \leq \frac{\pi}{2}$. Initial state-time of the bat is along the x-axis ($\pi/2$ radians) with zero velocity $\tilde{\mathbf{x}}_\mathbf{o} = [\frac{\pi}{2} \;\; 0 \;\; 0]^T$. The motor at the axis of the bat can only apply torque up to $|u_{max}| = 2Nm$.
The bat's goal is to hit a planar ball moving on a straight line with constant velocity. That is, the goals differential constraint is given by

$$\dot{\mathbf{q}} = \begin{pmatrix} Vx_b \\ Vy_b \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{q}_o = \begin{pmatrix} x_{b,o} \\ y_{b,o} \\ Vx_b \\ Vy_b \end{pmatrix} \tag{11}$$

where Vx_b and Vy_b are the balls constant velocities and $x_{b,o}, y_{b,o}$ are its initial positions at time $t = 0$. This representation should be transformed to a state-time space with reference to the bat. The position of the ball is transformed to polar coordinates where the angle denotes the angle the bat should have in order to hit the ball. The state-time tolerance for terminating the search is given by the tolerance vector $\epsilon = (2.8[^o] \;\; 0.2[\frac{rad}{s}] \;\; 0.03[s])^T$. For demonstration purpose and to demonstrate a full state goal, we set a constraint that at encounter, the bat will have the linear velocity at the contact point opposite but equal to the balls velocity component perpendicular to the bat. The offset of bat relative to the balls center due to the balls radius was taken into account. The motion of the ball was sampled only where it is located in the quarter circle representing the bats work space.
Under these properties and constraints, the algorithm was simulated and after 1,726 iterations outputted the TB-RRT shown in Figure 4. The green circles mark the sampled state-time vectors of the ball. The red curve presents the path solution from the initial state time to the goal one.
The algorithm outputted a set of discrete states in time and a set of control inputs. We may use the control inputs directly for the motor with no feedback control on the trajectory position. However, for overcoming disturbances and errors there is a need for feedback control. Therefore, we use the set of discrete states in time as the reference trajectory to a closed loop PD controller as seen in Figure 5. The TB-RRT planner outputs the desired trajectory from the initial state-time to the final. The reference signal is then fed into a simple PD controller with feedback to calculate the appropriate closed loop control input to the dynamic system. Figure 6 presents the trajectory of the bat. The dashed curve is the trajectory from the TB-RRT planner. The continues curve is the controlled trajectory simulated with SIMULINK. The ball's trajectory could also be seen as the dotted curve. The ball's curve starts at 0.46s because only then it enters the workspace of the bat. Rendezvous between the curves show that at $t_f = 0.505s$ the state-times of the bat and the ball are

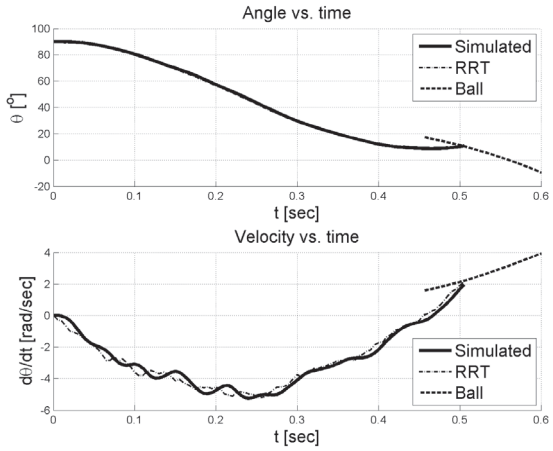

Fig. 6. The angle (up) and angular velocity (down) vs. time of the bat hitting a ball.

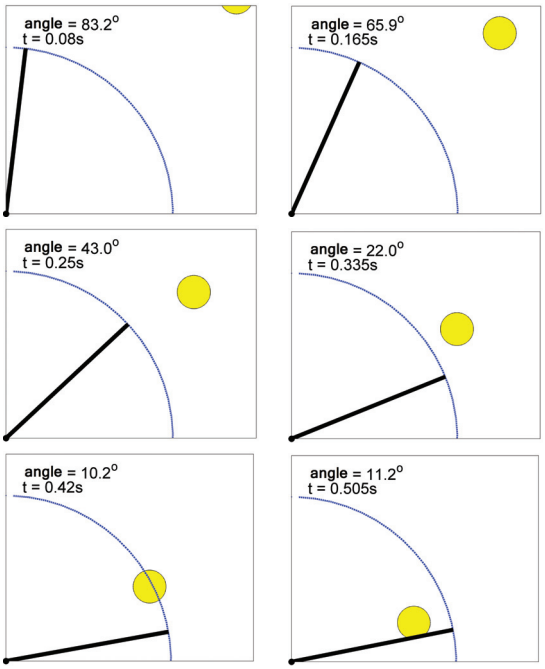

Fig. 7. The motion of the bat to hit the moving ball. The dashed line marks the workspace of the bat.

equal. Figure 7 shows snapshots of the bats motion to hit the ball.

B. Manipulator catching a moving object

In the second simulation we used a more complex dynamics. A vertical planar three degrees of freedom manipulator was chosen. Its dynamics is given by the following equation of motion

$$M(\mathbf{z})\ddot{\mathbf{z}} + C(\mathbf{z}, \dot{\mathbf{z}})\dot{\mathbf{z}} + G(\mathbf{z}) = \mathbf{u} \tag{12}$$

where $\mathbf{z}(t) = [\theta_1 \ \theta_2 \ \theta_3]^T$ is the joint configuration, $M(\mathbf{z})$ is the inertia matrix, $C(\mathbf{z}, \dot{\mathbf{z}})$ is the matrix of coriolis and centrifugal torques, $G(\mathbf{z})$ is the gravity vector and $\mathbf{u}(t)$ is the control input vector. The masses of the links are $m_1 = 1[kg],\ m_2 = 1.25[kg],\ m_3 = 0.75[kg]$ and their lengths are $L_1 = 0.2[m],\ L_2 = 0.25[m],\ L_3 = 0.15[m]$. Their center of masses are in the middle of each link. From this definition we perform order reduction such that $\mathbf{x_1} = \mathbf{z},\ \mathbf{x_2} = \dot{\mathbf{z}}$ and construct the systems state-time representation as follows:

$$\dot{\tilde{\mathbf{x}}} = f(\tilde{\mathbf{x}}, \mathbf{u}) = \begin{pmatrix} \mathbf{x_1} \\ M^{-1}\left(\mathbf{u} - C(\mathbf{x_1}, \mathbf{x_2})\mathbf{x_2} - G(\mathbf{x_1})\right) \\ 1 \end{pmatrix} \tag{13}$$

where $\tilde{\mathbf{x}} = [\mathbf{x_1}^T \ \mathbf{x_2}^T \ 1]^T$ is the state-time vector representation of $\mathbf{z}$. The upper two joints were constrain to be $-\frac{5}{6}\pi \leq (\theta_2, \theta_3) \leq \frac{5}{6}\pi$ and the joints velocity were limited up to 20 rad/sec. The joint torques were limited to a maximum of 5 N/m.

The manipulators goal is to catch a moving object such that at the moment of rendezvous the end-effector will have the same velocity (magnitude and direction) as the object. The object was given an arbitrary polynomial trajectory given by

$$\mathbf{r}(t) = \begin{pmatrix} -251.5t^4 + 219.8t^3 - 49.6t^2 + 0.2 \\ -298.7t^4 + 237.5t^3 - 45.5t^2 - 0.2 \end{pmatrix}, \tag{14}$$

where $\mathbf{q} = [\mathbf{r}^T \ \dot{\mathbf{r}}^T]^T$ is the objects state. This polynomial representation can be used to simulate the dynamics of a free flying object with the influence of gravitation and air drag damping. The goal objects differential constraint should be transformed to the state-time space as described. However, the manipulator state space is represented based on the joint angles and the object is based on its cartesian position. Therefore, the objects state space is transformed to the manipulators state space using the inverse kinematics of the manipulator. That is, the new sampled state space $\mathbf{q}^*$ of the object is given by

$$\mathbf{q}^* = \begin{pmatrix} h(\mathbf{r}) \\ J^{-1}\dot{\mathbf{r}} \end{pmatrix} \tag{15}$$

where $h(\mathbf{u})$ is the transformation from the end-effectors position to the joints configuration and J is the Jacobian matrix mapping joint velocity to end-effector velocity. Therefore, $\mathbf{q}^*$ is the state of the object represented in the manipulators state space.

Figure 8 presents a simulated trajectory of the manipulator to grasp the moving object. The trajectory was calculated with the TB-RRT algorithm and generated this solution after 21,988 iterations. In this case, after 0.4 seconds the manipulator was able to reach the object with the same velocity. Figure 9 presents snapshots of the manipulators motion to catch the moving object. It should be mentioned that a method to avoid collisions of the manipulators links were not implemented but could be done in the future by excluding parts of the state-time space from $\tilde{\mathcal{X}}_{free}$.

C. Analysis

Table I shows some performance results of the presented simulations. Runtime and iteration number statistics is shown after 10 simulation runs. Runtime is increased as well as the number of iterations when the state space dimension

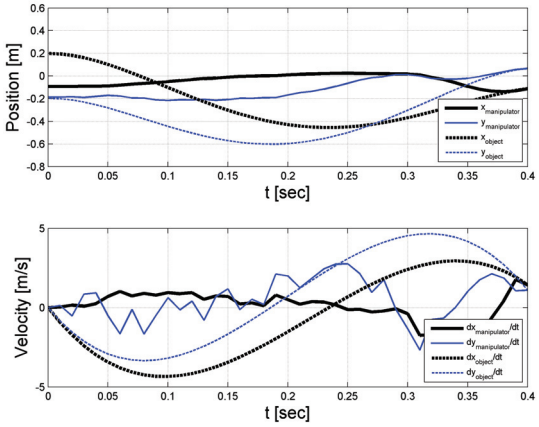

Fig. 8. The trajectory of the end-effector and the object. After 0.4 seconds they have the same position and velocity.

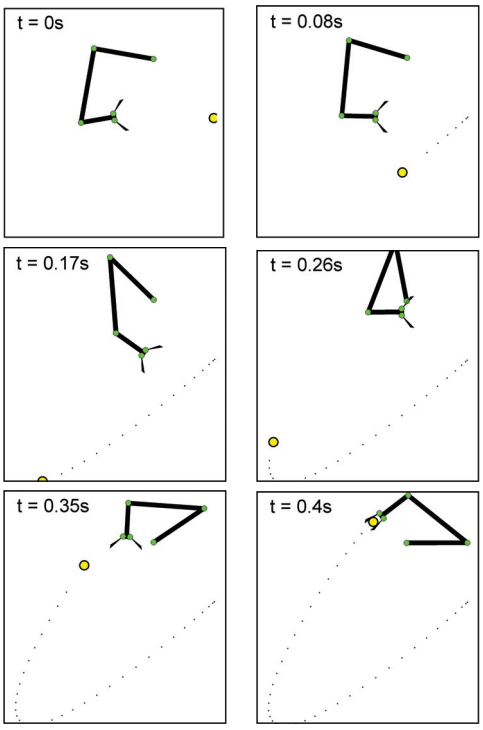

Fig. 9. The motion of the manipulator to catch the object.

increases. The simulation for the bat was done with time tolerance of ± 0.04 seconds. Figure 10 demonstrates the sensitivity of the algorithm with reference to time tolerance increase. Here, decrease in the runtime and number of iterations can be seen as the time tolerance grow. As expected, with larger tolerances more solutions could be found, however, with less accuracy.

VI. Conclusions

In this paper we have presented an approach for trajectory planning under specific time constraint. Moreover, we have extended the approach for rendezvous planning between a controlled dynamic system and an uncontrolled one. The

TABLE I

System	Iterations		Runtime	
	min	max	min	max
Bat	1,287	19,344	1.17s	193.5s
Manipulator	11,945	36,365	31.5s	1633s

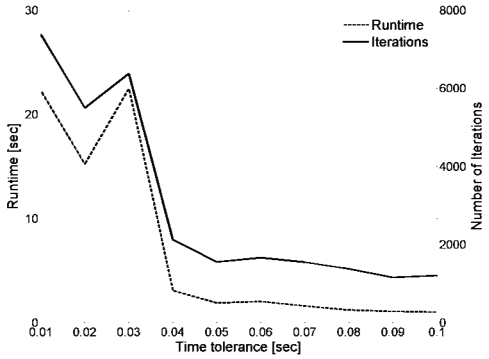

Fig. 10. Sensitivity of the algorithm in terms of runtime and iteration number as the time tolerance increases.

TB-RRT approach was presented building a probabilistic tree in the defined state-time space, taking the dynamics of the systems into account. Simulations of typical applications were shown to demonstrate and validate the method.

Further work on this approach should deal with computation time of expanding the tree to enable real-time planning. Such work should address decreasing the complexity of nearest neighbor search algorithms. Another approach could be real-time expansion of the existing TB-RRT based on feedback data only in regions which are relevant to the systems motion.

References

[1] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Trajectory planning in robotics," *Mathematics in Computer Science*, vol. 6, no. 3, pp. 269–279, 2012.

[2] S. Ge and Y. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonomous Robots*, vol. 13, no. 3, pp. 207–222, 2002.

[3] A. Piazzi and A. Visioli, "Global minimum-jerk trajectory planning of robot manipulators," *IEEE Transactions on Industrial Electronics*, vol. 47, no. 1, pp. 140–149, 2000.

[4] Z. Bien and J. Lee, "A minimum-time trajectory planning method for two robots," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 414–418, 1992.

[5] S. LaValle and J. Kuffner, J.J., "Randomized kinodynamic planning," in *Proceedings. IEEE International Conference on Robotics and Automation*, vol. 1, 1999, pp. 473–479.

[6] K. S. Erer and O. Merttopçuoglu, "Indirect Impact-Angle-Control Against Stationary Targets Using Biased Pure Proportional Navigation," *Journal of Guidance Control Dynamics*, vol. 35, pp. 700–704, Mar. 2012.

[7] M. Mehrandezh, N. Sela, R. Fenton, and B. Benhabib, "Robotic interception of moving objects using an augmented ideal proportional navigation guidance technique," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 30, no. 3, pp. 238–250, 2000.

[8] J. Michael, K. Chudej, M. Gerdts, and J. Pannek, "Optimal rendezvous path planning to an uncontrolled tumbling target," *eprint: arXiv:1307.1327*, 2013.

[9] T. Rybus and K. Seweryn, "Trajectory planning and simulations of the manipulator mounted on a free-floating satellite," in *Aerospace Robotics*, J. Ssiadek, Ed. Springer, 2013, pp. 61–73.

[10] R. Sedgewick and K. Wayne, *Algorithms, 4th Edition*. Addison-Wesley, 2011.