Path Planning with Collision Avoidance for Free-Floating Manipulators: A RRT-Based Approach

João R.S. Benevides and Valdir Grassi Jr.(✉)

Department of Electrical and Computer Engineering,
São Carlos School of Engineering (EESC), University of São Paulo (USP),
São Carlos, Brazil
{jrsbenevides,vgrassi}@usp.br

Abstract. The difficulty of creating a path planner with collision avoidance for Space Manipulators (SMs) is well known due to the presence of dynamic singularities and because of its non-holonomic behaviour. Furthermore, the main contributions in the field of motion planning of SMs are often concentrated in the point-to-point strategy, with special interest in the complex dynamics of such systems. In fact, planners for space manipulators generally count on a previously computed path in order to modify it to avoid collisions. Nonetheless, the computing of the previous path still lacks robust formulations, specially in the case of free-floating manipulators. Our goal consists in creating a path planner with collision avoidance for a free-floating planar manipulator. The dynamic model is based on the Dynamically Equivalent Manipulator and the concept of Rapidly-Exploring Random Trees serves as a framework for the developed algorithm. A combination of a method that reduces the metric sensitivity with a bidirectional approach is proposed in order to achieve a solution convergence. Details of the collision checking algorithm are provided. The system is validated by simulating the path planning task for a three-link planar free-floating manipulator, while considering the presence of an obstacle. The results are then discussed and promising directions for future works are presented.

1 Introduction

Space missions are often related to hostile environments, which are also connected to extreme temperatures, radiation and lack of gravity. These factors endanger and complicate human mobility in Extra-Vehicular Activities (EVAs). In order to assist in assembly services, space manipulators (SMs) are playing a key role in this matter. Activities like substitution of components, satellite repair and refueling are fundamental in the sense of making more flexible on-orbit operations and increasing the overall mission lifespan [1].

Path planning is known to be a major challenge in the field of general robotics. In the case of space robots, this difficulty is magnified by the dynamic coupling and the non-holonomic behavior of such systems, due to the nonintegrability of

F. Santos Osório and R. Sales Gonçalves (Eds.): LARS 2015/SBR 2015, CCIS 619, pp. 103–119, 2016.
DOI: 10.1007/978-3-319-47247-8_7

the angular momentum [2]. Another challenging task is the handling of dynamic singularities (DS), which had their existence proven in [3]. These differ from singularities of fixed-based manipulator because their location cannot be simply predicted from the kinematic manipulator structure. In fact, dynamic singularities are product of the dynamic properties of space robots and depend on the path taken.

Space manipulators are normally classified into two major categories. First, free-flying manipulators count on an active position and attitude control. This compensates the displacement generated by the joint motions in order to maintain a stable basis. Therefore, most of the control laws for fixed-base manipulators also apply. However, excessive fuel consumption compromises the duration of on-orbit missions. On the second group are the free-floating manipulators, which allow the satellite to freely move in response to the arm's motion. In that case, no reaction wheels or propulsion jets are used. Thus, the system can save fuel and energy. Nonetheless, this advantage comes with an extra challenge regarding the description of its dynamics and behavior.

In spite of so many difficulties, researchers gave valuable contributions in the matter of motion planning of space robots, specially in the sense of avoiding DSs and dealing with their special nature. The work presented in [4] adopts unit quartenions in order to represent dynamic singularities and avoid them. Using this representation, inverse kinematics algorithms are formulated based on geometric variables. An analytical path planning method for free-floating manipulators is presented in [5]. The cartesian control of the end-effector is achieved along with the system's attitude control. Nevertheless, trajectory points are supposed known and collision avoidance is not considered in this planning. [6] proposes a path planning technique that yields the appropriate initial system configurations to avoid DSs. However, the approach is based on a reference path prior to the application of the proposed technique, specifically, a straight line is considered in this evaluation. As one can notice, path-planning of free-floating robots in the presence of obstacles reveals a vast scenario to be exploited.

With the goal of evaluating space systems on earthly environments, a platform for assessing different control approaches for a free-floating planar manipulators was built in [7]. The UnderActuated Robot Arm-E (UARM-E) consists of a mechanical-electronical system that floats over an Ealing-like table. This manipulator is remotely connected to a simulation and control environment, also developed in this work. Moreover, this platform can be configured with up to six active joints with one or two arms. A typical configuration of the UARM-E is shown in Fig. 1.

The path planning of free-floating manipulators does not count on solid formulations when random obstacles are considered. This encouraged us to pursue a methodology of a planner that autonomously computes a path between two configurations of a free-floating manipulator. The Rapidly-Exploring Random Tree (RRT) algorithm, widely known as a powerful path-planning tool, acts as a framework for the proposed architecture. Two other major RRT-based approaches are considered in order to solve convergency problems and

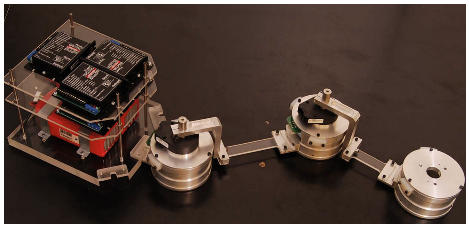


Fig. 1. UARM-E configured with a single arm and two active links.

core details about the implementation are given. These concepts were blended and modified to finally form the structure of the final planner.

This paper is organized as follows. Section 2 covers the key concepts about the dynamically equivalent manipulator. Interesting properties, which are later exploited, are derived here. Section 3 introduces the basic RRT algorithm along with two modifications. This section also proposes and discusses some of the adaptation to the problem. Section 4 describes how all the elements are integrated, provides details of the collision detection method we used and presents the architecture of the proposed planner. Section 5 shows the results of planning tasks in a simulated environment. Finally, Sect. 6 gives the proposed method a general overview and discusses promising directions for future works.

2 Dynamically Equivalent Manipulator

The concept of DEM was introduced in [8] as an alternative to the complex kinematic-dynamic approaches when modelling space robots. This method maps a free-floating manipulator into a conventional fixed-base manipulator, preserving both its kinematic and dynamic properties. This equivalence not only allows the modelling of free-floating arms through traditional methods, but also enables the experimental study of space platforms in more feasible environments, without the need for complex structures that emulate space conditions.

The DEM is originally based on the concept of Virtual Manipulator (VM), presented in [9], in which a kinematically equivalent manipulator is proposed. However, the VM equivalent model is considered to be an ideal kinematic chain with null mass. Therefore, practical experiments are unfeasible for this approach. The DEM exploits that fact and proposes an equivalent model that can be physically built and adopted in experimental studies regarding the dynamic behavior of space manipulators. Besides, it may be used as a tool for developing the dynamic model itself. The DEM is applicable for both free-floating and free-flying manipulators.

Consider a n-links rigid manipulator mounted over a free-floating base. Let C_i be the center of mass of link i and the base from the space manipulator named as link 1. Furthermore, the following links are named from 2 until $n+1$. Assuming that forces and external torques are non-existent, the center of mass

C_o remains fixed in inertial space and is also chosen as main frame's origin (depicted as frame 0, in Fig. 2).

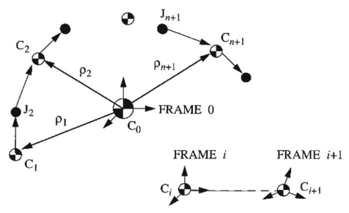


Fig. 2. Frame fixed to SM links.

Briefly, the model derivation uses Lagrange equations in the space manipulator to obtain the dynamic model of a fixed-based manipulator. In this case, the fixed base is replaced by a passive spherical joint. Considering that the DEM works in the absence of gravity and that its base is located at the center of mass of the space robot, the conditions under which both models are equivalent satisfy the following algebraic equations:

$$\begin{aligned}
m_i^{'} &= \frac{M_t^2 m_i}{\sum_{k=1}^{i-1} m_k \sum_{k=1}^{i} m_k}, && i = 2, \ldots, n+1 \\
I_i^{'} &= I_i, && i = 1, \ldots, n+1 \\
W_1 &= r_1, && \\
W_i &= r_i + l_i, && i = 2, \ldots, n+1 \\
l_{c1} &= 0, && \\
l_{ci} &= \frac{\sum_{k=1}^{i-1} m_k}{M_t} L_i, && i = 2, \ldots, n+1
\end{aligned} \tag{1}$$

In (1), W vectors represent the DEM link lengths and their inertial orientations with respect to the space manipulator (SM) inertial frame; $m_i^{'}$ corresponds to the mass of DEM's i-th link; $I_i^{'} \in \mathbb{R}^{3\times 3}$ denotes the inertial tensor corresponding to DEM's i-th link; l_{ci} represents the vector from DEM's i-th joint to the center of mass of i-th link. Additionally, [8] demonstrates that the value of $m_1^{'}$ does not influence in dynamic equivalency. Thus the mass of the DEM's first link might be arbitrarily assigned as a positive non-null value.

Let the generalized coordinates vector be $q = \left[\phi\ \theta\ \psi\ \theta_2 \cdots \theta_{n+1}\right]^T \in \mathbb{R}^{(n+3)}$ decomposed as $q = \left[q_b^T\ q_m^T\right]^T$, where b and m represent the base and manipulator components, respectively. Similarly to classic Euler-Lagrange equations, the system model assume the special form:

$$M(q_m)\ddot{q} + C(q_m, \dot{q})\dot{q} = \tau \tag{2}$$

As the gravity effects are neglected in a spatial environment, so is the gravity vector. In (2), $M(q_m) \in \mathbb{R}^{(n+3)\times(n+3)}$ denotes the symmetric and positive definite inertia matrix, which is dependent exclusively of manipulator coordinates; $C(q_m, \dot{q}) \in \mathbb{R}^{(n+3)\times(n+3)}$ represents the matrix of Coriolis and centrifugal forces. Finally, $\tau = \left[0\ 0\ 0\ \tau_2 \cdots \tau_{n+1}\right]^T \in \mathbb{R}^{(n+3)}$ denotes the vector of applied torques over DEM joints.

Because the DEM coordinate frames are parallel to the SM corresponding frames, and its base is located at the center of mass of the SM, the DEM is identical in geometry to the VM. Therefore, it inherits the fundamental properties of the VM, which are:

– The DEM end-effector coincides with the SM's end-effector.
– The axis of DEM's i-th joint is parallel to the axis of the i-th SM joint.
– During motion, the displacement of each of the DEM's joints is identical to the displacement of the corresponding SM joint.

These properties have their importance later demonstrated in the task of path-planning of free-floating manipulators.

3 Rapidly-Exploring Random Trees

Aiming to contribute in the field of sampling-based algorithms, the basic RRT algorithm, introduced in [10], stands out for its natural support to non-holonomic systems and several DOFs. The algorithm provides simple, but powerful search concepts that are explained as follows:

Let T be a tree rooted at its initial state, x_{init}. In each iteration, a random state x_{rand} is uniformly sampled in the free workspace, X_{free}. The algorithm then applies a search method in order to find the state in T that is closest to x_{rand}, based in a certain metric ρ. This state is now called x_{near}. Then, a valid random input vector u is sampled among all set of possible inputs U. Each of the inputs in u is applied in x_{near} and integrated over a certain time interval. After evaluating all the expansions from x_{near} that are collision-free, the one with lowest cost to x_{rand} is chosen as x_{new}. This new state is then added to the nodes of tree T. Likewise, the edge connecting the node x_{near} to x_{new} is added to the edges of T. This process is repeated until the state x_{new} is close enough to x_{goal}, meaning that the algorithm successfully found a solution for the path planning problem. In that case, the sequence of branches that reaches x_{goal} with minimum total cost according to the metric ρ is returned by the algorithm together with the sequence of inputs used to create those branch sequence. However, a solution may not exist or it is extremely hard to find. In order to avoid endless searches for a solution, a limit on the size of T is imposed to restrain the search time. So the algorithm stops when T achieves a user-defined maximum number of branches N. If that limit is achieved, the algorithm does not return a valid solution for the path planning problem. The pseudo-algorithm is shown as Algorithm 1.

Algorithm 1. Basic RRT Planner

```
BuildRRT($N$, $\Delta T$,$x_{init}$)
 1: Add_State_To_Tree($x_{init}$,$Tree$);
 2: for $n = 1$ to $N$ do
 3:     $x_{rand} \leftarrow$ Sample_State();
 4:     $x_{near} \leftarrow$ Find_Closest_Neighbor($Tree$,$x_{rand}$);
 5:     $u \leftarrow$ Sample_Input($x_{near}$,$x_{rand}$,$U$) ;
 6:     $x_{new} \leftarrow$ New_State($x_{near}$,$\Delta T$,$u$) ;
 7:     Add_State_To_Tree($x_{new}$,$Tree$);
 8:     Add_Edge_To_Tree($x_{near}$,$x_{new}$,$Tree$);
 9: end for
10: return $Tree$
```

In Algorithm 1, the function *New_State* is responsible for finding the next state of the system x_{new} by integrating the dynamic model of the robot, represented by a state-pace transition equation $\dot{x} = f(x, u)$, for fixed time ΔT applying input u from state x_{near}. Techniques with higher degree of integration, such as Runge-Kutta or the Euler method, are preferred for solving that integration [10].

The sequence of inputs applied to the dynamic model used for generating the path from x_{ini} to x_{goal} is also returned. In an ideal case, if the dynamic model used for path planning would be a perfect representation of the real system, the application of these sequence of inputs would make the robot to describe the desired path. However, due to modelling uncertainties and unpredictable disturbance, a path-following feedback controller must be used to follow the planned path. Nonetheless, the planned path is feasible because the dynamic model of the system was used for path planning.

Some of the main aspects and advantages of using the RRT algorithm, according to [10,11], are: (1) Strong bias to not yet explored regions; (2) Probabilistic completeness, this means that the probability of finding a solution tends to 1 as time tends to infinity; (3) States always remain connected; (4) Independence of explicit description of X_{free}; (5) Few heuristics or random parameters are required.

The RRT planner, seems to present better convergence ratio when a subtle bias p is applied to the final configuration. It means that the algorithm has probability p to choose final configuration over a random state. However, if p is set too large, it is likely that the search gets trapped in local minima [12].

3.1 Reduction of Metric Sensitivity

As a criterion, the chosen metric ρ is based on a cost function that should translate the cost of bringing one state to another. The ideal metric is represented by the optimal displacement cost between two states. It is known that performance degrades substantially when the chosen metric does not reflect the real cost of motion between two configurations. In fact, computing the ideal metric was proven to be as hard as solving the trajectory problem itself [11].

In order to remedy this situation, a modification in Algorithm 1 is proposed in [13]. The main goal of this method is to refine the exploration strategy, even in the absence of a proper metric. The contribution of this approach does not lie in developing a specific metric, but collecting information along the exploration and growth of the tree.

Basically, a record of all set of inputs is kept for each node. This allows the discard of inputs that were already evaluated. Also, the constraint violation frequency (CVF) is collected for each state. The CVF estimates the probability of a node expansion to result in a collision. Initially, every node has CVF equals to zero. Once a state is selected as x_{near}, its CVF is increased a quantity c/N, where c stands for the number of inputs that result in collision or movement restriction and N the number of inputs. Furthermore, when a $CVF > 0$ is computed, all father-nodes that lead to that state have their CVF incremented accordingly. That is, its m-th father will be added a CVF of c/N^{m+1}. The CVF is then used as probability of not choosing some state when selecting the closest neighbor. By punishing regions that lead to nodes that are likely to collide, the exploration information helps the planner to pick a better x_{near}.

In order to implement this method and consequently assign a CVF to a node, it is necessary to keep a record of all the possible inputs that can be applied to that node for its expansion. This ca only be done if instead of a continuous input we restrict the choices of inputs to a discrete.

Following we present a method of organizing a set of discrete inputs for the problem of a generic free-floating manipulator, when torques are the only input to be considered. Let $x \in X$ a node, whose father is x_{father}. Consider the set of inputs necessary to take x_{father} to x, over a certain time interval Δt, represented by a vector $\tau = \begin{bmatrix}\tau_1 \; \tau_2 \; \cdots \; \tau_n\end{bmatrix}^T$, where n denotes the number of links of the space manipulator. Let us impose $+\Delta_i$ and $-\Delta_i$ as a threshold in the increase and decrease of τ_i, respectively. Also, as the torque τ_1 is null due to DEM equivalence conditions, the resulting set of inputs is organized as shown in Table 1, where each row comprises the set of possible torques to be applied.

Table 1. Organization of the discrete inputs

0	$\cdots$	0	$\cdots$	0
$\tau_2 - \Delta_2$	$\cdots$	τ_2	$\cdots$	$\tau_2 + \Delta_2$
$\tau_3 - \Delta_3$	$\cdots$	τ_3	$\cdots$	$\tau_3 + \Delta_3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\tau_{n-1} - \Delta_{n-1}$	$\cdots$	τ_{n-1}	$\cdots$	$\tau_{n-1} + \Delta_{n-1}$
$\tau_n - \Delta_n$	$\cdots$	τ_n	$\cdots$	$\tau_n + \Delta_n$

$\underbrace{\hspace{10em}}_{k \text{ columns}}$

As k stands for the degree of discretization of each input in τ, the amount of possible combinations equals k^{n-1}. The algorithm will then need at least k^{n-1} *bits* to keep track of the expansion of each set of inputs. A naive approach would also consider to keep record of the set of inputs that lead to that expansion. However, for several degrees of freedom and many iterations this may represent an unnecessary allocation of memory, degrading the overall computational performance.

As a workaround, we created for each state a single vector V, with size k^{n-1}, built as follows: The indices $\left(i_2\ i_3\ \cdots\ i_n\right)$, relative to the columns in the torques matrix presented in Table 1, are stored. This set of indices (that correspond to inputs) is associated to element L from vector V as computed in (3).

$$L = 1 + \sum_{j=2}^{n} (i_j - 1) k^{n-j} \tag{3}$$

Another adaptation was made to the method for reduction of metric sensitivity. An heuristic was adopted in order to eliminate, from the closest neighbour search, nodes with high probability of producing child nodes with collision. Consider a set of all possible inputs U for a given state x, and a set of inputs that was already sampled and verified, $U_s \in U$. If U_s reaches a considerable size with respect to the size of the set U, and all input in U_s resulted in collision, then the node x is not selected for further expansion. This is done because is highly probable that other inputs in U that were still not tested would also result in collision, meaning that this node is not able to produce valid children.

The reduction of metric sensibility, combined with the modifications above described, collaborated to find a better expansion during the initial experiments. However, the task of reaching a goal position while avoiding obstacles could not find a single solution, even though 50000 iterations were run in each trial. The unidirectional RRT seems to have special sensibility to local minimas. Depending on a single tree to proper explore the environment and converge to a solution has been verified to be a hard task, specially in the presence of obstacles. This motivated us to also incorporate the bidirectional approach.

3.2 Bidirectional RRTs

Using a single RRT from x_{init} to connect with x_{goal} works well for state spaces of low dimension. The bidirectional RRT, proposed originally by [11], grows two RRTs independently. This approach improves the efficiency for state spaces of high dimension at the expense of having to connect a pair of nodes between two trees. The main aspects are described as follows:

Like the basic algorithm, the first tree has its root at the initial state x_{init}. A second tree is then grown from final state x_{goal}. At each iteration, a random state x_{rand} is sampled and an expansion attempt is made from the first tree. After that, the second tree takes the same x_{rand} as parameter and tries to expand in that direction. The process is repeated until two states x_1 and x_2 are

Algorithm 2. Dual RRT Planner

```
BuildBiRRT()
1: Add_State_To_Tree($x_{init}$,$Tree_1$);
2: Add_State_To_Tree($x_{goal}$,$Tree_2$);
3: while not Connected do
4:     $x_{rand} \leftarrow$ Sample_State();
5:     $x_i \leftarrow$ Create_State($Tree_1$,$x_{rand}$,$Forward$);
6:     if $x_i \in X_{free}$ and close enough from $Tree_2$ then
7:         connect $Tree_1$ to $Tree_2$ through $x_i$;
8:     end if
9:     $x_g \leftarrow$ Create_State($Tree_2$,$x_{rand}$,$Backward$);
10:    if $x_g \in X_{free}$ and close enough from $Tree_1$ then
11:        connect $Tree_2$ to $Tree_1$ through $x_g$;
12:    end if
13: end while
```

close enough. This proximity is verified if $\rho(x_1, x_2) < \delta$, for a small $\delta > 0$. The pseudo-algorithm can be seen in Algorithm 2.

Notice that, because the second tree is created from final configuration towards the initial one, its integration must be computed backwards in time. Considering Δt the step size used to grow the main tree, this issue can be solved by using another $\Delta t^{'} = -\Delta t$, everytime a backward integration is necessary. The reason for this lies in the fundamental equality $\int_a^b f(x)dx = -\int_b^a f(x)dx$.

Another important aspect to highlight is that the bidirectional RRT also require the solution for the inverse kinematic of the manipulator at the desired final pose of the end-effector in order to integrate and compute new states for the second tree. However, it is difficult to find an analytic solution for the space manipulator. For this reason as both the free-floating manipulator and the DEM have the same end-effector location, the inverse kinematic was handled by iteratively computing the joint positions of the DEM manipulator from the end-effector. After finding the DEM joint positions, the real manipulator is constructed from the DEM through direct kinematics and the collision detection algorithm performs the validation of such configuration.

The knowledge of the joint positions provides a thorough description of every configuration. This is particularly interesting because it allows the program to consider more parameters, thus enabling a more accurate estimation of the cost-to-go.

4 Creating the Path Planner

In order to use consistent data for the simulated environment, the test platform UARM-E had its parameters, shown in Table 2, incorporated into the algorithm.

The DEM parameters are then computed with (1) and shown in Table 3:

In order to detect collision of an obstacle with the space manipulator at a given configuration, we decided to use the Separating Axis Theorem (SAT)

Table 2. SM parameters of UARM-E robot

	$m_i(kg)$	$I_i(kgm^2)$	$L_i(m)$	$R_i(m)$
Base	4.780	0.0404	0	0.150
Link 2	1.380	0.0182	0.144	0.111
Link 3	1.011	0.0115	0.103	0.112

Table 3. DEM parameters of UARM-E robot

	$m_i(kg)$	$I_i(kgm^2)$	$W_i(m)$	$l_{ci}(m)$
Base	4.780	0.0404	0.100	0
Link 2	2.410	0.0182	0.191	0.096
Link 3	1.177	0.0115	0.201	0.089

Algorithm [14]. The main goal was to perform a rapid collision check among the manipulator and the obstacles. Additionally, a self-collision check is performed using the same approach. As the SAT applicability is restricted to convex forms, we chose to treat each robot link as a convex polygon and to represent the space manipulators as an open kinematic chain. An illustration of the SM and its DEM is depicted in Fig. 3.

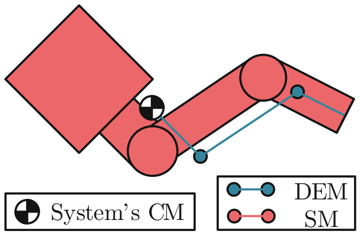


Fig. 3. SM constructed from the DEM.

The SAT is a special case of Minkowski's separating hyperplan theorem applied to solve a collision detection problem. Essentially, the theorem states that two convex objects do not collide if there is at least one line (called here as separating axis), upon which the objects projections do not overlap (see Fig. 4). Algorithm 3 presents the pseudocode of the Separating Axis Theorem regarding two bidimensional objects O_1 and O_2. On this algorithm, a sweep is done around the β angle of the separating axis over a step size ξ between each collision check.

Regarding the execution of SAT algorithm described in Algorithm 3, some particularities were observed and then modifications were introduced in the sense of reducing computational cost. First, the axis is no longer sampled. Instead, it is always perpendicular to a line containing an object's side. The reason for

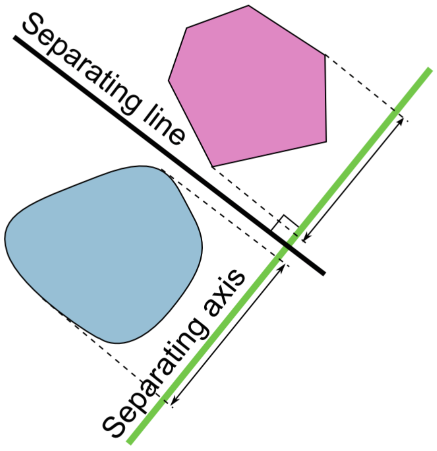


Fig. 4. Illustration of the Separating Axis Theorem.

Algorithm 3. SAT Naive Approach

SAT(O_1,O_2)

```
 1: $state \leftarrow COLLISION$;
 2: $\beta \leftarrow 0$;
 3: while ($\beta < 360$) AND ($STATE = COLLISION$) do
 4:     $axis \leftarrow$ line with $\beta$ angle;
 5:     if Projection of $O_1$ and $O_2$ over $axis$ = OVERLAP then
 6:         $state \leftarrow NO\_COLLISION$;
 7:         $break$;
 8:     else
 9:         $\beta \leftarrow \beta + \xi$;
10:     end if
11: end while
```

this modification is straightforward: assuming that two polygonal and convex objects do not collide, there is at least one line that passes between them without touching. Therefore, there is at least one line, parallel to the side of one of the objects, that also freely passes between them. Second, only non-parallel sides are considered when building the separating axis. As parallel sides result in the same separating axis, we shorten the number of searches for all of the parallel sides. Finally, because the algorithm verifies whether joint angles are feasible prior to the execution of the SAT, it is not necessary to check for self-collision for two subsequent links. Algorithm 4 summarizes the main modifications for a more efficient collision check among two generic, convex and bidimensional objects O_1 and O_2.

Figure 5 gives an overview of the organization of the planner structure. The proposed planner binds the concepts and modifications presented so far to

Algorithm 4. SAT for Space Manipulator Collision Check

```
SAT($O_1$,$O_2$)
 1: $state \leftarrow COLLISION$;
 2: SetOfCheckedAngles $\leftarrow EMPTY$;
 3: $\beta \leftarrow$ angle from line containing one side of $O_1$ or $O_2$;
 4: while ($\exists\beta$ to check in $O_1$ OR $O_2$) do
 5:     $axis \leftarrow$ line with $\beta$ angle;
 6:     if Projection of $O_1$ and $O_2$ over $axis$ = OVERLAP then
 7:         $state \leftarrow NO\_COLLISION$;
 8:         $break$;
 9:     else
10:         SetOfCheckedAngles $\leftarrow$ Add $\beta$;
11:         $\beta \leftarrow$ angle $\in$ ($O_1$ OR $O_2$) AND $\notin$ SetOfCheckedAngles;
12:     end if
13: end while
```

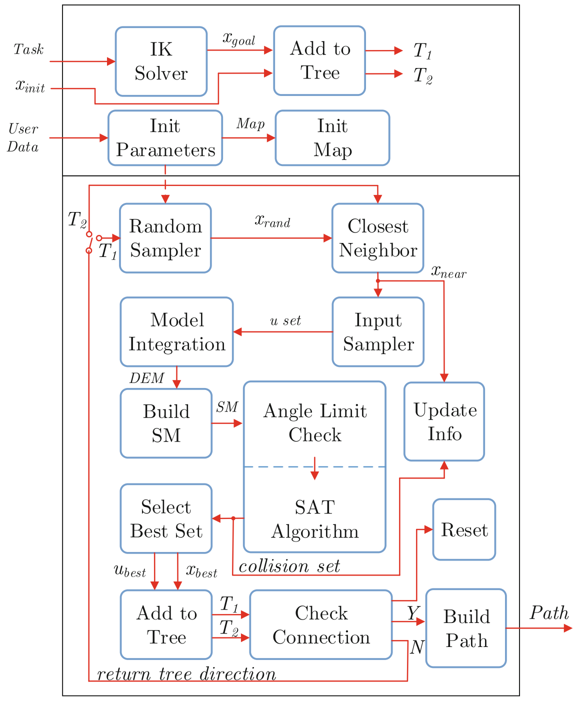


Fig. 5. Overview of the planner

successfully find a path between two configurations. The presented architecture has its main components and characteristics described as follows.

– **IK Solver:** Computes the inverse kinematic based on the end-effector goal position. Basically, the position of each DEM joint is computed iteratively, from last to first, based on the joint limitations and lengths of previous links. As infinite solutions may appear, one is randomly picked and converted back to the space manipulator (SM) model. If this configuration is assessed as collision-free, the IK solver has a valid solution.

- **Random Sampler:** Picks a valid random configuration. For that purpose, a sequence of angles is randomly chosen in order to build the DEM model. The SM is then build after the DEM. In the case some collision or joint limitation is detected, a new sample is computed. The same sampled configuration is used for the tree to grow in the opposite way.
- **Closest Neighbor:** Finds the closest neighbor from random configuration. [13] provides the pseudocode for this matter.
- **Input Sampler:** Samples an input set from the total input set (U_s from U). Consider Table 1 representing the discrete inputs. A number $m < k$ of samples is randomly chosen for each joint. Thus m^{n-1} set of inputs are sampled.
- **Model Integration:** Integrates the dynamic DEM model for every input in U_s.
- **Build SM:** Builds the free-floating manipulator from the DEM model. This is done by iteratively applying direct kinematics from end-effector to the base.
- **Collision Detection:** Checks every expansion made and identify the ones that are collision-free based on the SAT algorithm. This block also verifies if some joint has reached its opening limits.
- **Update Info:** Updates the constraint violation frequencies (CVFs) and marks inputs already evaluated.
- **Select Best Set:** Checks, among every expansion considered, the one that is collision-free and has the lowest cost of motion to the desired configuration.
- **Add to Tree:** Add new nodes and edges to the tree.
- **Check Connection:** Runs the routine of connection verification between two trees.
- **Build Path:** Builds the path between initial and final configurations after connecting two trees.
- **Init Parameters:** Loads all constant parameters of manipulator and algorithm. Initializes also variable parameters that are user-defined, such as task description and bias to goal.
- **Init Map:** Loads obstacles that compose the map. These must be described as poligonal objects.
- **Reset:** As observation of simulations, the convergence of the algorithm proposed in Fig. 5 is still jeopardized due to local minimas in some cases. It was noticed that the first 1000 samples suffice to provide a fair glimpse of how well the exploration would be conducted. Hence, this block restarts the planner if a reasonable approximation is not achieved during the first 1000 iterations. This enables the algorithm to spare effort in the search for a solution.

5 Results and Discussions

For the proposed task, the UARM-E robot was configured with two active joints in one single arm. The obstacle in the environment was considered to be fixed and rectangular. The task tries to find a path between two configurations without hitting the obstacle. For that purpose, a step size $\Delta T = 0.001\,\text{s}$ is used for integrating the DEM, which is performed through Euler method. All of the results were obtained through simulations, runned in Matlab - version R2012a. The computer was powered with an *Intel*® *Core i*7, 3.40 GHz and 12 GB RAM. A tolerance region is created around the goal configuration. Aiming to provide an intuitive idea of the tree growth from initial to final configuration, all the nodes representing end-effector positions are plotted as points for both trees. Examples are given in Figs. 6 and 7.

If we allow the planner to continue the searching, it generally comes up with a better solution as the tree expands in free workspace. Figure 7 shows an expansion after 50000 iterations.

To achieve a convergence between two states, the metric was computed based on: Euclidean distance (x); Torques difference (τ); Velocities difference (υ).

Consider two configurations a and b. In order to relate the parameters, variable x is made $x = -\sum |x_a - x_b|$, where x_a and x_b denote the positions of joints in a and b, respectively. Therefore, $x \to 0$ as the configurations get closer

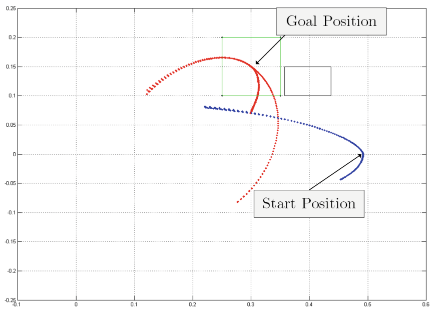


Fig. 6. Expansion after 1000 iterations. The green box stands for the tolerance region. The black box depicts the obstacle. The main tree is presented in blue, while the second tree, rooted at the goal configuration, is depicted in red. (Color figure online)

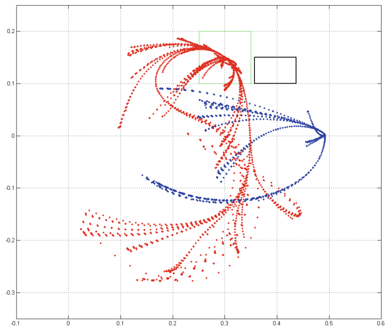


Fig. 7. Expansion after 50000 iterations

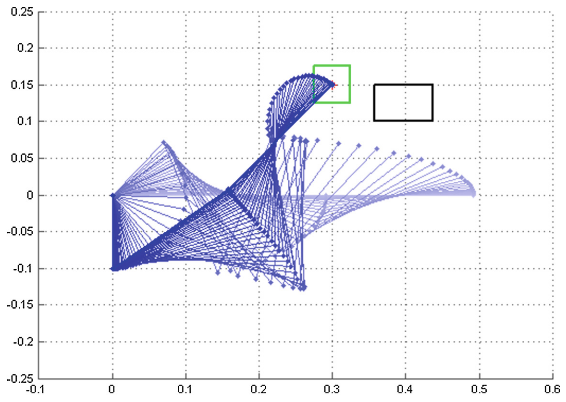


Fig. 8. Example of a computed path. The DEM representation of robot UARM-E gets darker as it approximates from the goal region.

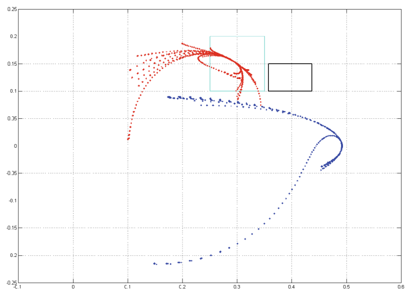
to each other. Equation (4) defines the form of the metric used. Similarly, the difference of torques and velocities between all joints in a and b is computed as $\tau = \sum |\tau_a - \tau_b|$ and $\upsilon = \sum |\upsilon_a - \upsilon_b|$, respectively.

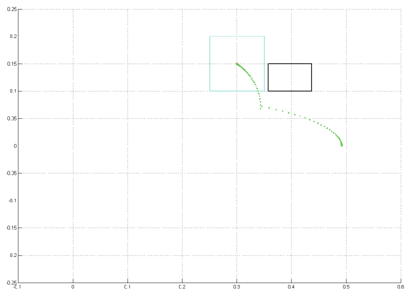
$$\rho(x) = k_1(x)x + k_2(x)\tau + k_3(x)\upsilon \tag{4}$$

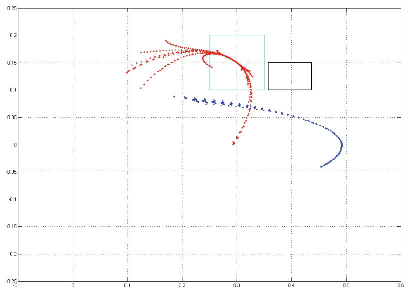
Because variables τ and υ need to influence metric ρ more heavily as configurations get closer, coefficients k_i are represented as sigmoid functions. Equation (5) presents an example of these sigmoid functions after adjustment of the magnitudes of all constants. This function shape has also the advantage of achieving a smooth transition between configurations.

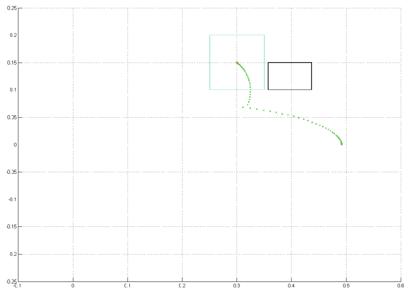
$$\begin{aligned} k_1 &= 0.8 - \frac{0.1}{1 + e^{(-5-100x)}}, \\ k_2 &= \frac{1.5 \cdot 10^{-3}}{1 + e^{(-6-50x)}}, \\ k_3 &= \frac{5 \cdot 10^{-6}}{1 + e^{(-5-120x)}}. \end{aligned} \tag{5}$$

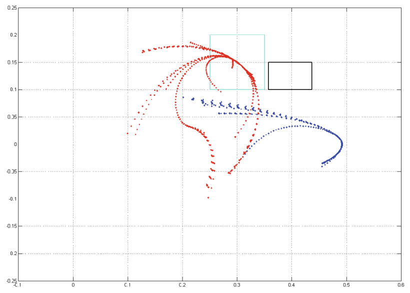
A path was considered found after $|x| < \xi$, with $\xi < 10^{-5}$. With the goal of evaluating the planner performance, 100 evaluation tests were run. On average, the algorithm needed 3827 nodes to find a solution. There were 11 cases where the planner could not find a solution, even after 25000 nodes were grown. Among the results, the solution with faster convergence needed 1604 nodes only. On the other hand, the solution that took longer time was achieved after 16445 nodes. Because of its dependency on the environment's complexity, the computation time was not chosen as a measure of convergence speed. Figure 8 shows a typical solution of the computed path. Only the DEM is shown in this picture for a better visualization. Figure 9 shows other examples of solutions found for the same problem.


(a)


(b)


(c)


(d)


(e)

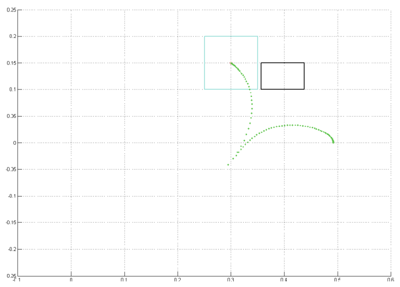

(f)

Fig. 9. (a), (c) and (e) represent expansions with 16445, 7280 e 4916 nodes, respectively. Paths associated with these figures were connected by the planner using 90, 89 and 111 nodes, respectively. Path computed is represented by green nodes in figures (b), (d) (e) and (f) (Color figure online)

6 Conclusion

The main goal of this paper was to present a feasible approach for automatically planning a collision-free path for a free-floating manipulator. Our main contribution was to provide such method through cooperating the RRT with the complex dynamics of free-floating manipulators with the help of the dynamically equivalent manipulator. Furthermore, enhancements like growing bidirectional trees and reducing the metric sensitivity were improved in order to create a robust path planner. A straightforward collision-check for the space manipulator is also shortly presented in order to help for a fast implementation. The proposed methodology proved itself functional after several tests and different conditions. Future works aim to expand the planner to consider trajectories and evaluate the RRT* performance in a different programming environment, like C++ or Python. Finally, we plan to extend the algorithms to operate with two-arm manipulation as well.

Acknowledgment. This research was supported by grants from Fundação de Amparo à Pesquisa do Estado do Amazonas (FAPEAM) and the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

References

1. Yoshida, K.: Achievements in space robotics. IEEE Robot. Autom. Mag. **16**(4), 20–28 (2009)
2. Li, C., Liang, B., Xu, W.: Autonomous trajectory planning of free-floating robot for capturing space target. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1008–1013 (2006)
3. Papadopoulos, E., Dubowsky, S.: Dynamic singularities in free-floating space manipulators. ASME J. Dyn. Syst. Meas. Contr. **115**, 44–52 (1993)
4. Caccavale, F., Siciliano, B.: Quaternion-based kinematic control of redundant spacecraft/manipulator systems. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 435–440 (2001)
5. Tortopidis, I., Papadopoulos, E.: Point-to-point planning: methodologies for underactuated space robots. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 3861–3866 (2006)
6. Nanos, K., Papadopoulos, E.: On cartesian motions with singularities avoidance for free-floating space robots. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 5398–5403 (2012)
7. Pazelli, T.F.P.A.T.: Assembly and nonlinear h infinite control of free-floating base space manipulators. Ph.D. dissertation, EESC-USpP (2011)
8. Liang, B., Xu, Y., Bergerman, M.: Mapping a space manipulator to a dynamically equivalent manipulator. Robotics Institute, Pittsburgh, PA, Technical report, CMU-RI-TR-96-33, September 1996
9. Vafa, Z., Dubowsky, S.: On the dynamics of manipulators in space using the virtual manipulator approach. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 579–585 (1987)
10. LaValle, S.M.: Rapidly-exploring random trees: a new tool for path planning. Computer Science Dept., Lowa State University, Technical report (1998)
11. LaValle, S.M., Kuffner Jr. J.J.: Randomized kinodynamic planning. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 473–479 (1999)
12. Urmson, C., Simmons, R.: Approaches for heuristically biasing RRT growth. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 2, pp. 1178–1183, Outubro (2003)
13. Cheng, P., LaValle, S.: Reducing metric sensitivity in randomized trajectory design. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 43–48 (2001)
14. Gottschalk, S.: Separating axis theorem. UNC Chapel Hill, Chapel Hill, NC, Technical report, TR96-024 (1996)