Path Planning with General End-effector Constraints: Using Task space to Guide Configuration Space Search

Zhenwang Yao
School of Engineering Science
Simon Fraser University
8888 University Dr, Burnaby, BC, Canada
zyao@cs.sfu.ca

Kamal Gupta
School of Engineering Science
Simon Fraser University
8888 University Dr, Burnaby, BC, Canada
kamal@cs.sfu.ca

***Abstract*— In this paper, we address the *path planning problem with general end-effector constraints* (PPGEC) for robot manipulators. Two approaches are proposed. The first approach is adapted from an existing randomized gradient descent (RGD) method for closed-chain robots. The second approach is radically different. We call it ATACE *Alternate Task-space And Configuration-space Exploration*. Unlike the first approach which searches purely in C-space, ATACE works in both task space and C-space. It explores the task space for end-effector paths satisfying given constraints, and utilizes trajectory tracking technique(s) as a local planner(s) to track these paths in the configuration space. We have implemented both approaches and compare their relative performances in different scenarios. ATACE outperforms RGD in majority (but not all) of the scenarios. We outline intuitive explanations for the relative performances of these two approaches.**

***Index Terms*— Path planning, end-effector constraints, ATACE.**

I. Introduction

In this paper, we address the *path planning problem with general end-effector constraints* (PPGEC) for robot manipulators. It is an extension of the basic motion planning problem [11], and is particularly appropriate for redundant robots, where extra degrees of freedom can be exploited to pursue additional objectives. These additional objectives often take form of constraints on the end-effector pose (position and orientation). For instance, the Space Station Remote Manipulator System (SSRMS, also called the Canadarm2) [9], may need to move a satellite payload to a desired position, while keeping it in a certain orientation. Another example is that a manipulator arm moving/transporting a glass of water should keep the glass vertically up all the time. In some other applications, the end-effector may be constrained to move in a plane.

While several works have considered specific forms of constraints, the problem with general end-effector constraints has not been addressed in previous works. For example, [1] considered *point-to-point inverse kinematics* where only the end-effector pose is specified at the goal, rather than the entire robot configuration. Applications such as welding require the end-effector to move along a given end-effector path (or trajectory), and it is required to determine a joint-space path (or trajectory) that tracks the given end-effector path, hence the name *trajectory tracking problem* [2], [6], [8], [14], [15], [16], [17], a well studied problem. In PPGEC, rather than the entire end-effector trajectory being specified, the end-effector pose is constrained to an allowable subset. For these constraints, re-parameterizing constrained C-space into lower dimension is in general difficult and tedious, although it may be done in case-by-case fashion (w.r.t different robots and constraints). We try to explore more general approaches.

This paper proposes two global planning algorithms for PPGEC. The first approach is a pure configuration space search based approach and is adapted from a *randomized gradient descent* (RGD) method originally proposed for closed-chain robots [12]. This method for closed kinematics chains essentially breaks a closed chain and then imposes a closure constraint at the "break point", which is essentially similar to a pure positional end-effector constraint. Please note that an alternate path planning method for closed chains [3], [7], although more efficient, explicitly uses the closed-chain mechanism structure, and is therefore not directly extended to our case of general end-effector constraints, although it can be used to efficiently generate configurations for a given end-effector pose. We discuss this in more detail in Section III-A.

The second approach is radically different, and explicitly explores the task space (T-space), the space of end-effector poses ($SE(3) = R^3 \times SO(3)$), for feasible end-effector poses and end-effector paths connecting them; and then with these end-effector paths, the corresponding C-space paths are generated by invoking a "local" trajectory tracking planner. Essentially, it uses T-space search to guide the C-space search. We call this approach ATACE, *Alternate Task-space And C-space Exploration.* Intuitively, searching for feasible end-effector paths is relatively easier than searching directly in C-space, since T-space normally has lower dimensionality (than C-space) and only considering the end-effector reduces the initial search cost. Having found feasible end-effector paths, *ATACE* avoids searching many infeasible regions of C-space. Our empirical evidence (as shown in later sections) suggests that in most scenarios, this strategy yields superior performance (faster run time and "better quality" end-effector paths).

There is a vast literature on "local" methods for trajectory tracking problem [6], [14], [15]. These approaches generally work at instantaneous velocity level, using the robot manipulator's Jacobian matrix. [17] extends Jacobian-based technique to a global approach for offline applications. [16] presents a probabilistic approach to the trajectory tracking problem. Any of these planners could be used as a trajectory tracking local planner within ATACE.

The organization of this paper is as follows: Section II presents the problem definition, and Section III presents the two approaches. Experimental results are shown in Section IV, followed by conclusion in Section V.

II. Problem Formulation

The end-effector (we use it in the generic sense, it could be a payload or a tool carried by the robot in its end-effector) constraints are task-related and denoted in terms of the end-effector pose, p, represented as a pair $(\mathcal{P}, \mathcal{O}) \in SE(N)$, where $\mathcal{P} \in R^N$ is the position of the origin of the end-effector frame, and $\mathcal{O} \in SO(N)$, an $N \times N$ rotation matrix, is the orientation of the end-effector frame with respect to a universal frame. $N = 2$ for planar applications and $N = 3$ for spatial applications. $\mathcal{P}$ can be parameterized by a triple (x, y, z), and $\mathcal{O}$ can be parameterized by another triple (α, β, γ), say with fixed-angle representation [4], hence p can be viewed as a six dimensional vector (locally). We denote a general end-effector constraint as

$$G_i(p(\tau)) = 0 \qquad \forall \tau \in [0, 1], \quad i = 1, 2, \cdots \tag{1}$$

where $p(\tau)$ is the end-effector pose along the path, and G is a continuous function in task space, which corresponds to a sub-manifold in C-space. For example, if the end-effector is required to move in a plane, the constraint will be:

$$G(p(\tau)) = ax(\tau) + by(\tau) + cz(\tau) - d = 0.$$

For orientation constraints, the constraint expression may be complex in general, but could be relatively simple. For instance, for a robot trying to keep a glass of water strictly vertical, the constraint will be:

$$G(p(\tau)) = \begin{cases} \beta(\tau) = 0 \\ \gamma(\tau) = 0 \end{cases}$$

Given this notation, we formulate the *PPGEC* problem as:

Given a start robot configuration q_s, and desired end-effector pose p_g, determine a reachable configuration q_g and a collision free path $q(\tau)$ to it, $\tau \in [0, 1]$, such that $q(0) = q_s$, $q(1) = q_g$, $F(q_g) = p_g$, and end-effector pose $p(\tau) = F(q(\tau))$ satisfies constraints (1) for all $\tau \in [0, 1]$, where $F(q)$ is the forward kinematics function of the robot.

Unlike the trajectory tracking problem where all feasible poses are given, in *PPGEC* problem, the pose is not explicitly given; instead it is implicitly described by constraints Eq (1), and the constraints normally correspond to a sub-manifold of feasible poses.

III. Algorithms for PPGEC

A. Adapted Randomized Gradient Descent (RGD)

Our first approach to deal with the PPGEC problem is to adapt the *randomized gradient descent* (RGD) method for closed-chain robots [12]. This method takes the closure constraint as a special type of end-effector constraint, and hence, it can be adapted for more general end-effector constraints. The RGD uses a PRM-based framework, and constructs a constrained roadmap whose nodes and edges satisfy the closure constraint. For PPGEC problem, we make the following three key modifications.

1) RGD for closed chains finds a feasible configuration by executing a randomized search for the root of a cost function that is defined as the distance between "broken" joints. For PPGEC problem, to satisfy more general end-effector constraints, the defined cost function depends on the constraints. It essentially represents the distance (from the current end-effector pose) to the constraint surface. For example, if the end-effector is required to move in a plane, then the cost function is defined as the distance between the current end-effector and the constraint plane.
2) RGD for closed chains assumes that the start and goal configurations are given. For PPGEC problem, the goal is defined as an end-effector pose. For a redundant robot, it normally corresponds to a sub-manifold of possible goal configurations. Therefore, roadmap query procedure needs modification. We use the configuration generation techniques for closed-chain robots [3], [7] to generate different configurations for the goal pose, and these different goal configurations are then used for roadmap query as in the original algorithm.
3) RGD for closed chains uses a multi-query PRM [10] based scheme. However, in some applications a single-query scheme may be preferred. For example, if the environment keeps changing and we use the roadmap just once, the multi-query planner may waste time exploring unreachable areas. In these cases, an RRT [13] like single-query method can avoid unnecessary exploration by limiting the search space in the reachable space from the start configuration. We have implemented both schemes, and name them *PRM-RGD* and *RRT-RGD* respectively, although in this paper, we report mainly on RRT-RGD.

For more detail about the original randomized gradient descent method, please refer to [12], and for more details on adaptations, see [18].

B. Alternate T-space and C-space Exploration (ATACE)

The second approach to solve the PPGEC problem is ATACE. Unlike the adapted RGD method, which works purely in C-space, *ATACE* works in both C-space and task space (T-space). It explores T-space first for feasible end-effector paths, and then tracks these end-effector paths in C-space with a trajectory tracking planner.

ATACE explores the task space with a number of end-effector paths and tackles general end-effector constraints by transforming these constraints into end-effector velocity constraints. To make sure that every end-effector path satisfies the constraints, the path is extended from a feasible pose, and the next pose in the path is computed by choosing a velocity that will not violate the constraints, i.e., choosing a velocity that is tangent to the constraint surface. For instance, if an end-effector is required to move in a plane, and the first pose is given in this plane, then obviously, as long as the end-effector moves with a velocity tangent to this plane, it will not violate this plane constraint. So, *ATACE* first determines the connectivity in the task space by choosing proper velocities and moving along with these velocities to come to a new node. In this way, both the connecting path and the new node are guaranteed to satisfy the constraints.

After getting a feasible end-effector path in task space, *ATACE* utilizes a local planner to track the end-effector path and avoid obstacles, joint limits and singularities. We now describe the ATACE planner in detail.

1) The Algorithm: *ATACE* constructs a search tree, $\mathcal{T}$, in T-space, and tracks the search tree in C-space. Every node in the search tree corresponds to a pair (q_k, p_k). q_k is a configuration, and p_k is the end-effector pose under this configuration, i.e., $p_k = F(q_k)$. Every edge is represented by an end-effector sub-path (in the task space) and a joint sub-path (in the configuration space) to track the end-effector sub-path. If there is a sequence of edges (sub-paths) in the tree joining the start pose and the goal pose, a feasible joint path is extracted to solve the problem. A schematic of the search tree is shown in Figure 1. The root, (q_s, p_s), is the start configuration and the corresponding pose. Planning terminates when a node with the goal end-effector pose, p_g, becomes a leaf of $\mathcal{T}$. Every node in the tree has additional information associated with it, contained in the following parameters:

1) (q, p), configuration q, and pose $p = F(q)$;
2) $parent$, parent node in the tree;
3) $pPath$, an end-effector sub-path connecting p to its parent pose;
4) $cPath$, a joint sub-path (to track $pPath$) connecting q to its parent configuration.

Note that although sub-paths of the tree are represented as straight lines in the figure, the actual path in the task space, $pPath$, or the corresponding path in the C-space, $cPath$, is not necessarily a straight line due to constraints.

ATACE_Plan() iteratively picks a random direction and grows the tree in this direction before trying to connect the tree to the goal. The *ATACE* algorithm is summarized below.

```
ATACE_Plan($q_s$, $p_g$)
  $p_s \leftarrow F(q_s)$;
  $\mathcal{T} \leftarrow$ Initialize_Tree($q_s$, $p_s$);
  REPEAT
    $q_d \leftarrow$ Random_Config(); $p_d \leftarrow F(q_d)$;
    $N_c \leftarrow$ Nearest_Node( $p_d$ );
    ($s$, $N_k$) $\leftarrow$ Extend_With_Constraint($N_c$, $p_d$, FALSE);
    IF ($s \neq Trapped$)
      IF (Connect_To_Goal($N_k$)=$Reached$)
        RETURN success;
  UNTIL (time out)
  RETURN failure;
```

Initialize_Tree() initializes the search tree $\mathcal{T}$ with node (q_s, p_s) as its root. *Random_Config()* randomly generates a configuration q_d. *Nearest_Node()* looks up $\mathcal{T}$, and finds the node $N_c = (q_c, p_c)$ which is the closest to $p_d = F(q_d)$. The metric(s) for finding N_c is described in Section III-B.4. *Extend_With_Constraint()* places a new node $N_k = (q_k, p_k)$ in $\mathcal{T}$ by stepping from N_c in the direction of p_d, more precisely, in the direction of the projection of p_d in the tangent space of the constraints. $Connect\_To\_Goal()$ tries to connect the tree to the goal from the new node N_k. These procedures are described in more detail in the following sections.

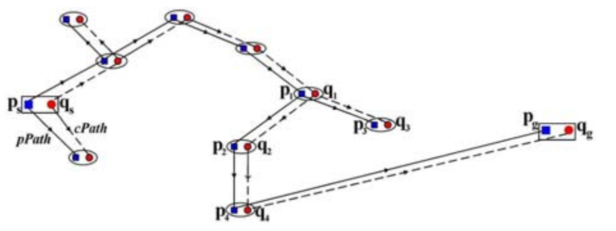


Fig. 1. A search tree constructed by *ATACE*. An oval with ■ and ● in it is a node in the tree; the leftmost and rightmost rectangles stand for the start and goal. (●, ■) represents a configuration-pose pair (q, p). A solid line represents a $pPath$, and a dotted line represents the corresponding $cPath$ that tracks the $pPath$

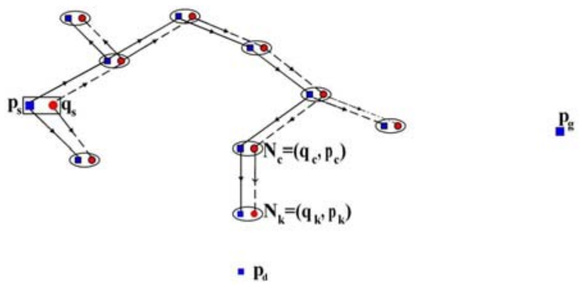


Fig. 2. *Extend_With_Constraint()* grows the tree to a new node N_k in direction of p_d. p_d is randomly generated. N_c is the closest node to p_d, before N_k is added.

2) Extend_With_Constraint(): As shown in Figure 2, given a node, N_c=(q_c, p_c), in $\mathcal{T}$, and a stepping direction in the task space, p_d, *Extend_With_Constraint()* tries to advance toward p_d (more precisely, toward its projection on the current tangent plane) along a path that also lies on the constraint surface (or satisfies the constraints). *Extend_With_Constraint()* returns (s, N), which can be: (a) ($s = Advanced$, $N = N_k$), a feasible path is extended; (b) ($s = Reached$, $N = N_k$), the extended feasible path reaches the projection of p_d; (c) ($s = Trapped$, N =NULL), no feasible path.

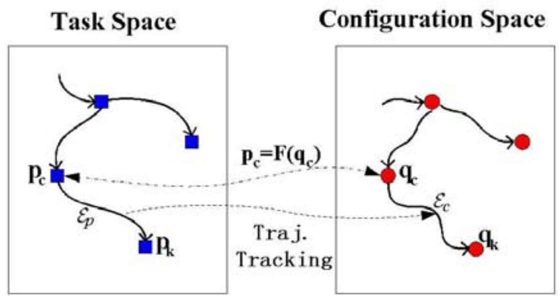


Fig. 3. T-space and C-space extension in *Extend_With_Constraint()*.

More specifically, *Extend_With_Constraint()* first extracts an end-effector sub-path $\mathcal{E}_p$ in the task space that satisfies the constraints, as in Figure 3(a), and then computes a sub-path in the configuration space, $\mathcal{E}_c$, by using the local planner to track $\mathcal{E}_p$, as shown in Figure 3(b). To assure an end-effector sub-path in the task space connecting two poses satisfies constraints, the sub-path is extracted by choosing feasible velocities toward a direction p_d at every step along the path. In the following pseudo-code, when parameter $greedy = FALSE$, the sub-path is extracted for no more than M steps, $\mathcal{E}_p = p^1, p^2, \cdots, p^M$; when $greedy = TRUE$ (this is needed when called by *Connect_To_Goal()*), the sub-path reaches p_d (actually its projection in the tangent plane), and may consist of more than M poses.

```
Extend_With_Constraint($N_c$, $p_d$, $greedy$)
  $\mathcal{E}_p \leftarrow \phi$;   $i \leftarrow 0$;   $p^i \leftarrow p_c$;
  REPEAT
    ($v^i$, $\omega^i$, $\hat{p}^i_d$)$\leftarrow$ Compute_Valid_Velocity($p^i$, $p_d$);
    ($s$, $p^{i+1}$) $\leftarrow$ Compute_Next_Pose($p^i$, $v^i$, $\omega^i$, $\hat{p}^i_d$);
    IF ( Is_Path_Clear($p^i$, $p^{i+1}$) )
      $\mathcal{E}_p \leftarrow \mathcal{E}_p \bigcup \{\overline{p^i p^{i+1}}\}$;
      $i \leftarrow i + 1$ ;
    ELSE
      $s \leftarrow Trapped$;
  UNTIL( ((!$greedy$) AND ($i \geq M$)) OR
            ($s = Reached$) OR ($s = Trapped$) )
  IF ($s \neq Trapped$)
    ($ss$, $\mathcal{E}_c$) $\leftarrow$ Track_EndEffector_Path($q_c$, $\mathcal{E}_p$);
    IF ($ss$ = success)
      $N_k.p \leftarrow$ the last element of $\mathcal{E}_p$;
      $N_k.q \leftarrow$ the last element of $\mathcal{E}_c$;
      $N_k.pPath \leftarrow \mathcal{E}_p$;
      $N_k.cPath \leftarrow \mathcal{E}_c$;
      $N_k.parent \leftarrow N_c$;
      $\mathcal{T} \leftarrow \mathcal{T} \bigcup N_k$;
    ELSE
      $s \leftarrow Trapped$;
  RETURN ($s$, $N_k$);
```

In the REPEAT-UNTIL loop, an M-step sub-path is extracted satisfying the constraints. *Compute_Valid_Velocity()* computes and returns a valid end-effector velocity (consisting of linear and angular velocity components) at the current (i^{th}) point, and the projection of p_d in current tangent plane, $\hat{p}^i_d$. Note that at every iteration, $\hat{p}^i_d$ may change and the sub-path is extended toward $\hat{p}^i_d$. If *Compute_Valid_Velocity()* returns $Reached$, it means the sub-path arrived at $\hat{p}^i_d$. With current pose p^i and current velocity v^i, *Compute_Next_Pose()* computes the next feasible end-effector pose over a user-specified fixed sampling time interval Δt. *Is_Path_Clear()* checks the collision for $\overline{p^i p^{i+1}}$. If it is in collision, the path extraction terminates with $s = Trapped$. If $greedy = TRUE$, the sub-path is extended to $\hat{p}^i_d$, even if it takes more than M steps. *Track_EndEffector_Path()* uses a local planner to solve the *trajectory tracking problem* for sub-path $\mathcal{E}_p$. If it succeeds, a joint space sub-path is returned in $\mathcal{E}_c$, and a new node N_k is connected to N_c.

a) Compute_Valid_Velocity(): This function transforms end-effector constraints into end-effector velocity constraints. Assume poses $p^0, p^1, \cdots, p^i$ are already generated, and we want to generate a feasible velocity, v^i (at pose p^i), using which we will compute the next pose satisfying the position constraint $G(p) = 0$.

From p^i, with the assumption that the time interval is small enough, we can go to another feasible pose providing the velocity vector is tangent to the constraint plane,

$$v : < \nabla G(p^i), v > \; = \; 0. \tag{2}$$

As shown in Figure 4, to extract the next pose, p^{i+1}, in the direction of p_d, we compute v^i as follows. Project p_d into the tangent plane. Denote $\hat{p}^i_d$ as its projection. Choose v^i as a unit velocity with the same direction as $\hat{p}^i_d - p^i$.

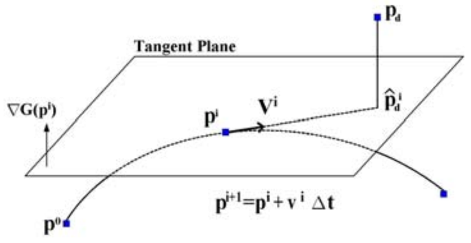


Fig. 4. Extract an end-effector sub-path in task space

For orientation constraints, consider the case of a robot holding a glass of water and trying to keep it strictly vertical. Let's say this requires the z axis of the end-effector frame to remain vertical with respect to the universe frame. The feasible angular velocity is

$$\omega = k \cdot [0 \quad 0 \quad 1]^T$$

where k is an arbitrary scalar value. A more general cases in which a unit vector $\vec{n}$ in the end-effector frame is required to be kept constant with respect to the universe frame has an feasible angular velocity of

$$\omega = k \cdot \vec{n}.$$

b) Compute_Next_Pose(): This function computes the next pose p^{i+1} (including position $\mathcal{P}^{i+1}$ and orientation $\mathcal{O}^{i+1}$), given the current pose p^i, the instantaneous velocity (v^i, ω^i) and the target pose $\hat{p}^i_d$.

- For position
$$\mathcal{P}^{i+1} = \mathcal{P}^i + v^i \cdot \Delta t \tag{3}$$
where v^i is a unit linear velocity ($|v^i| = 1$), and Δt is the sampling interval. If $\mathcal{P}^i$ satisfies the constraints, we can assume over a small time interval, $\mathcal{P}^{i+1}$ also satisfies the constraints with an error $O(\Delta t)$.
- For orientation
$$\begin{aligned} \mathcal{O}^{i+1} &= R(\vec{K}, \theta) \cdot \mathcal{O}^i \\ \theta &= sign(\omega^i) \cdot |\omega^i| \cdot \Delta t \\ \vec{K} &= \tfrac{\omega^i}{|\omega^i|} \end{aligned} \tag{4}$$
ω^i is the angular velocity, $R(\vec{K}, \theta)$ is the matrix of equivalent rotations around axis $\vec{K}$ by angle θ.

Note that the error may accumulate due to discretization and linearized approximation of constraints, so adjustments may need to be done if the error is out of bound.

c) Track_EndEffector_Path(): This function calls a local planner to track the extracted end-effector path. In *ATACE* different local planners can be selected. Any planner for the trajectory tracking problem is suitable to be the local planner. For example, we can use a deterministic local Jacobian-based planner as in [6], [8], [14] which can easily take obstacles into account and avoid them without violating the constraints; or we can use a probabilistic planner as in [16]. The local planner returns success or fail flag to indicate whether a joint path to track the given end-effector path has been found.

3) Connect_To_Goal(): This function tries to connect current tree $\mathcal{T}$ to the goal by extending the tree from the newly-added node N_k to the goal pose p_g. Here, the end-effector path extracted in *Extend_With_Constraint()* is a path from p_k to p_g. Because a greedy algorithm is used to extend the tree to the goal pose p_g, the exploration for the end-effector path succeeds only when p_g is achieved.

```
**Connect_To_Goal**($N_k$)
    ($s$, $N$) ← Extend_With_Constraint($N_k$, $p_g$, TRUE);
    RETURN $s$;
```

4) Nearest_Node(): To find the closest node to p_d in the tree $\mathcal{T}$, we need a metric to measure distance between two nodes. Different metrics can be used. We use a metric in task space, which is defined as:

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \quad (5)$$

where $(x_i,\ y_i,\ z_i)$ is the position component of pose p_i, and the orientation component of pose is simply ignored in this metric. To take orientation into consideration, the metric adopted by [1] can also be used, which is defined as the distance between two frames.

IV. Computer Simulations

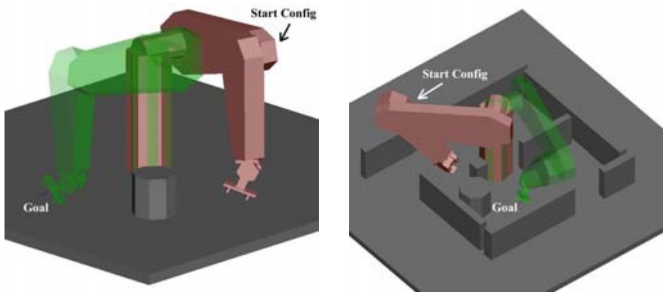
We have implemented both the adapted RGD with an RRT-based scheme (called RRT-RGD) and ATACE with our in-house developed software library MPK, the Motion Planning Kernel [5]. The simulation were carried on a Pentium-4 2.0G PC.

The first two cases are shown in Figure 5. A PUMA-like 6-DOF robot manipulator is moving its end-effector in a plane parallel to the floor. Without considering the orientation part in the end-effector poses, the start and the goal end-effector poses are the position shown in the figure. Case 1 is a fairly simple case, where there is only one small obstacle in the plane. Case 2 is much harder. There is a fence around the robot and several other obstacles in the environment. The start configuration and the goal are in different cells of the fence, and the robot has to move out of a gap in the fence and go through another gap to reach the goal.

Two other cases for orientation constraints are shown in Figure 6. Case 3 uses a 9-DOF robot manipulator - a 6-DOF PUMA-like manipulator mounted on a 3-DOF platform. The task is to move the end-effector to the goal while maintaining the end-effector pointing horizontally right (the z-axis of end-effector frame matches with the x-axis of the universe frame). In Case 4, the manipulator has similar kinematics structure as the Canadarm2, the Space Station Remote Manipulator System (SSRMS) [9]. It has 7 DOFs and the task in this case is to move a satellite while maintaining the satellite orientation vertically upwards. In the figure the large cylinder at the tip of SSRMS represents a satellite.

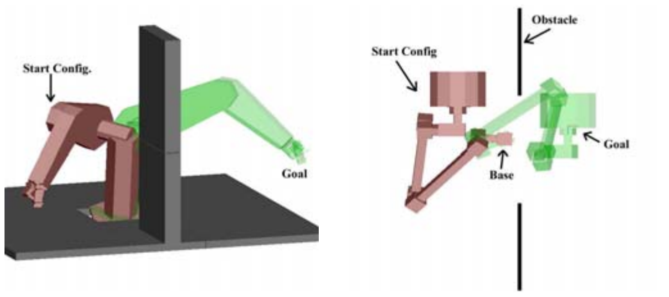
We applied RRT-RGD and ATACE planners to these cases and the results are shown in Table I. With 12 runs for each planner in each case, the column of *Succ/12* shows how many times (out of 12 runs) the planner succeeds to find a path within 1000 seconds, and the column of *Time (s)* shows the average planning time in seconds. If a planner fails to find a path within the limit, the average planning time is computed with 1000.00 for this run (with a ">" sign in the average planning time.)

Our intuition, as outlined in the introduction, is borne out by these experiments. In a simple environment like Case 1, *RRT-RGD* runs faster than *ATACE*. As there is much room for the robot to move around, i.e., most of the C-space is free and virtually any C-space trajectory is easily transformed to a feasible trajectory satisfying the constraints. In more cluttered examples, *ATACE* is faster than RRT-RGD since it first searches in the task space, which has much lower



Case 1 Case 2

Fig. 5. Experiment scenes for planar constraints



Case 3 Case 4

Fig. 6. Experiment scenes for orientation constraints

Case	*Planner*	*Succ/12*	*Time (s)*
1	ATACE	12	12.63
1	RRT-RGD	12	0.68
2	ATACE	12	61.43
2	RRT-RGD	1	>982.38
3	ATACE	12	7.17
3	RRT-RGD	9	>459.89
4	ATACE	12	99.62
4	RRT-RGD	12	292.09

TABLE I
Comparison of different planners in case 1-4. *Succ/12* column indicates how many times (out of 12 runs) the robot is able to find a path within 1000 seconds. *Time(s)* column shows the average planning in seconds over the 12 runs.

dimensionality than the C-space for redundant robots. For example in cases 1 and 2, since only the position is considered the dimensionality of the task space is 3. In addition, it avoids searching useless C-space areas by ruling out infeasible end-effector paths early in the search process, especially when the environment is complex (like case 2) or the end-effector payload is a relatively large object (like case 4). Lastly, the trajectory tracking planner we used in the experiments is a Jacobian-based planner [14], which is more powerful than the local planner used in *RRT-RGD* in the sense that it uses self-motion to avoid obstacles; In more cluttered environments, it tends to be more efficient when it makes connections between nodes.

Another advantage we have noticed in the experiments is that the paths found by *ATACE* (with a Jacobian-based trajectory tracking planner) normally have "better quality" than those found by RGD planner in sense that the robot has less jerky and tortuous movement along the path. To demonstrate this, the joint paths found by different planners are shown in Figure 7. This is because with Jacobian-based planner, ATACE computes the intermediate configurations that are optimal with respect to movement (since it minimizes the joint velocity norm), while in RGD method, the configurations are generated in a probabilistic fashion.

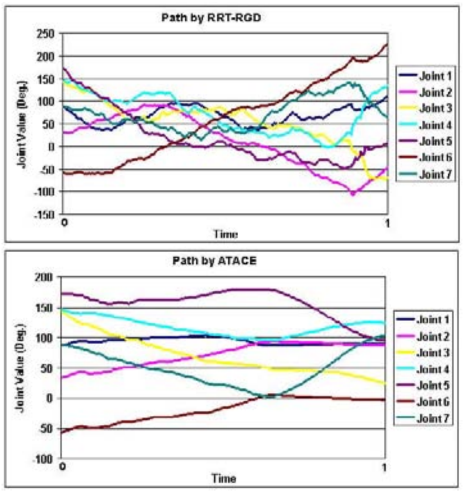


Fig. 7. Comparison of paths (in Case 4) found by RRT-RGD and ATACE.

V. Conclusion

To solve the path planning problem with general end-effector constraints, we proposed two approaches in this paper: adapted-RGD and ATACE. Adapted RGD is adapted from the gradient descent method for closed-chain robots. ATACE is a new planning technique, it explores the task space for end-effector paths satisfying constraints, and tracks these paths in configuration space. Experimental results show that in many complex environments, ATACE achieves significantly better planning time.

A lazy (in the sense that it postpones tracking) version of ATACE has also been investigated, where it first finds an entire end-effector path before tracking all the sub-paths therein, instead of tracking the sub-paths at every iteration. As many of end-effector sub-paths in the tree will not contribute to the final end-effector path, in some cases lazy ATACE has better running time, but not always. For example, in cases (1) and (4) lazy ATACE has better running time, while in (2) and (3) it does not. In the latter two cases we observed that lazy ATACE wasted time in tracking many non-trackable end-effector paths that traverse through the joint limits. For example, in case (2) a path along the upper fence is not trackable due to the joint limits.

Although the main motivation for ATACE came from the fact that end-effector constraints are expressed in the task-space, hence it makes sense to guide the C-space search by first searching the task-space, there is no reason why the ATACE paradigm cannot be extended to the basic path planning problem as well. We are currently exploring it.

References

[1] J. Ahuactzin and K. Gupta. The kinematic roadmap: A motion planning based global approach for inverse kinematics of redundant robots. *IEEE Transactions on Robotics and Automation*, 15:653–669, 1999.

[2] P. Chiacchio, S. Chiaverini, L. Sciavicco, and B. Siciliano. Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy. *International Journal of Robotics Research*, 10:410–425, 1991.

[3] J. Cortes, T. Simeon, and J.P. Laumond. A random loop generator for planning the motions of closed kinematic chains with prm methods. In *IEEE International Conference on Robotics and Automation*, pages 2141–2146, 2002.

[4] J. Craig. *Introduction to Robotics: Mechanics and Control, Second Edition*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[5] I. Gipson, K. Gupta, and M. Greenspan. MPK: An open extensible motion planning kernel. *Journal of Robotic Systems*, 18(8):433–443, Aug. 2001.

[6] Z. Guo and T. Hsia. Joint trajectory generation for redundant robots in an environment with obstacles. In *IEEE International Conference on Robotics and Automation*, pages 157–162, 1990.

[7] L. Han and N. Amato. A kinematics-based probabilistic roadmap method for closed kinematic chains. In B. Donald, K. Lynch, and D. Rus, editors, *Workshop on Algorithmic Foundations of Robotics*, pages 233–246, March 2000.

[8] T. Hsia and Z. Guo. Joint trajectory generation for redundant robot. In *IEEE International Conference on Robotics and Automation*, pages 14–19, 1989.

[9] Internet. http://www.space.gc.ca/asc/eng/iss/mss_ssrms.asp. (Accessed in Jan 2005).

[10] L. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. In *27th Annual ACM Symposium on Theory of Computing*, pages 353–362, Las Vegas, Nevada, United States, 1995.

[11] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[12] S. Lavalle, J. Yakey, and L. Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. In *IEEE International Conference on Robotics and Automation*, pages 1671–1676, 1999.

[13] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Dept. Iowa State University, Oct. 1998.

[14] A. Maciejewski and C. Klein. Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *International Journal of Robotics Research*, 4:109–117, 1985.

[15] Y. Nakamura and H. Hanafusa. Task-priority based redundancy control of robot manipulators. *International Journal of Robotics Research*, 6:3–15, 1987.

[16] G. Oriolo, M. Ottavi, and M. Vendittelli. Probabilistic motion planning for redundant robots along given end-effector paths. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1657–1662, 2002.

[17] S. Seereeram and J.T. Wen. A global approach to path planning for redundant manipulators. *IEEE Transactions on Robotics and Automation*, 11:152–160, 1995.

[18] Zhenwang Yao. Path planning with end-effector constraints. Master's thesis, School of Engineering Science, Simon Fraser University, 2004.