In: Proc. 17th IEEE International Conf. on Advanced Robotics (ICAR 2015), Istanbul, Turkey (2015).

Real-Time Motion Adaptation using Relative Distance Space Representation

Yiming Yang, Vladimir Ivan and Sethu Vijayakumar
School of Informatics, University of Edinburgh, Edinburgh, UK
Email: {yiming.yang, v.ivan, sethu.vijayakumar}@ed.ac.uk

***Abstract*—Reacting to environment changes is a big challenge for real world robot applications. This paper presents a novel approach that allows the robot to quickly adapt to changes, particularly in the presence of moving targets and dynamic obstacles. Typically, a configuration space replanning or adaptation is required if the environment is changed. Rather, our method aims to maintain a plan, in a relative distance space rather than configuration space, that can be valid in different environments. In addition, we introduce an incremental planning structure that allows us to handle unexpected obstacles that may appear during execution. The main contribution is that the relative distance space representation encodes pose re-targeting, reaching and avoiding tasks within one unified cost term that can be solved in real-time to achieve a fast implementation for high degree of freedom (DOF) robots. We evaluate our method on a 7 DOF LWR robot arm, and a 14 DOF dual-arm Baxter robot.**

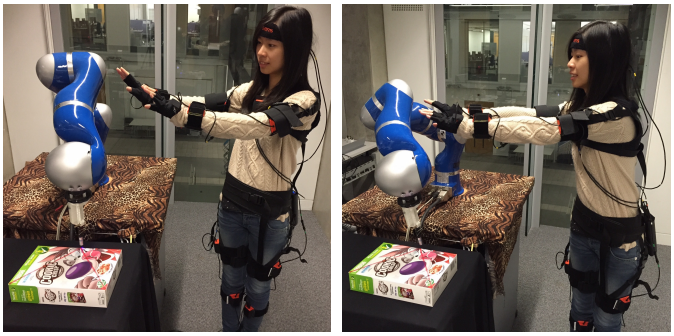


Figure 1: Robot-human close interaction. Left figure shows the robot's original motion, the right figure shows the adapted motion when human subject pushes her hands to block the original trajectory.

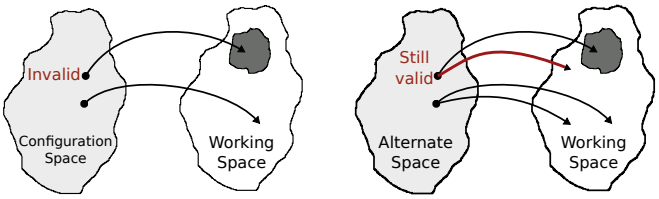
I. Introduction

In the field of robotics, motion planners usually take a snapshot of the environment, construct a model of the environment and, using this mode, computes a trajectory consisting of a series of robot configurations [1]. Each configuration typically has one corresponding pose in working space (Figure 2a), meaning that the robot trajectory in the world is fixed. Such an approach is sufficient if the environment is known and static. However, it is not suitable for real applications, where the environment is changing and new objects may appear and disappear. Controlling robots in dynamic environments is one of the most difficult problems in robotics. It arises in tasks such as manipulating moving objects, or interacting with humans, as shown in Figure 1. The trajectory can be invalidated due to various reasons, e.g. the trajectory is blocked by obstacles, the target moves outside of the working envelope of the robot, etc. In these cases, replanning is typically required to calculate a new feasible plan. Replanning is, however, an expensive process that causes delay, which makes real-time implementation of fast, dynamic motion a significant challenge.

There are different ways to allow robots to operate in dynamic environments. One of them is to keep replanning during execution based on the sensory information [2]. This is a robust but also an expensive approach that normally can not be used for tasks that require both accuracy and efficiency. Recently, real time replanning following this approach has been achieved by using many-core GPUs [3], where multiple processors are created simultaneously to speed up the computation. Online replanning cost can be reduced by interleaving planning with execution [4], where they split a whole trajectory into multiple sub-trajectories and only plan for one of them at each step. One can also apply motion adaptation methods such as Dynamic Movement Primitives (DMP) [5] when one has access to a demonstrated trajectory, where any captured motion gets encoded into a set of differential equations. These methods can be used to handle perturbations during execution [6] [7]. The Artificial Potential Field (APF [8]) method and its derivatives have gained popularity in the field of mobile robots to solve problems involving online collision avoidance. APFs use the idea of imaginary forces acting on the robot, where the obstacles have repulsive forces and target has an attractive force. The robot is driven by the sum of all these forces, calculated based on the minimum distance between robot and obstacles/target. From this point of view, APFs can be considered as a relative distance based approach. Park et al. [6] introduced a dynamic potential field where the potential field takes obstacles' velocities into account to provide more robust plans. Similar to global methods, the performance of local planners such as APF can also be improved with the aid of parallel computing [9]. A dynamical system (DS) based approach is introduced in [10], where an original motion can be modified on-the-fly to avoid convex obstacles. However, it only considers the end-effector trajectory while there are situations where the trajectories of other links are in collision as well.

While these aforementioned methods aim to find valid configuration space plans (Figure 2a), some approaches encode the plans in alternate spaces [11] [12] (Figure 2b). Relationship based representations have been studied in computer graphics [13] [14] [15] for motion re-targeting problems and they have been applied to robotics in [16], [15] and [17]. Rather than using configuration space, these methods represent the problems in some alternate spaces in which the relationships between

978-1-4673-7509-2/15/$31.00 ©2015 IEEE



(a) One-to-one mapping from configuration space to working space.

(b) One-to-many mapping from alternate space to working space

Figure 2: (a): the configuration space plan will be invalidated if the working space state is in collision. (b): a state in alternate space is still valid even if some of the working space states are in collision.

robot and environment are encoded, generating executable plans by capturing relational invariances. The alternate space states typically have multiple corresponding robot poses, as shown in Figure 2b, such that the state can be still valid if some of the corresponding poses are not. Typically these methods need to know the relationship in advance and encode them into the alternate space. In real world scenarios, there are many situations in which one encounters unexpected and unmodelled objects (e.g. moving obstacles, people) which is non-trivial to handle online with these existing alternate space methods.

In order to allow the robots to handle unexpected objects, for problems that involve reaching targets around dynamic obstacles in particular, we present a relative distance based space representation, in which we model the relative distances between robot links, targets and obstacles. In addition, we construct the relative distance space plan in an incremental way, which gives the robot the ability to avoid not only the obstacles which are known apriori during planning phase but also the unexpected obstacles which are detected during execution. We assume that there only exists one global minima, such that we can employ a fast local method to remap from relative distance space to joint space. In contrast to some other end-effector trajectory adaptation methods, e.g. DS approach, our method is able to adapt the trajectories of all robot links simultaneously. We apply our approach on a 7 DOF robot arm with a mock-up welding problem, as illustrated in Figure 1. We also demonstrate the scalability of our method on a 14 DOF dual-arm Baxter robot with a water pouring task. In both experiments, the robots operate in relatively unstructured environments, where the robot needs to accomplish the tasks while avoiding colliding with human.

II. Relative Distance Space

We will now present our method for capturing interactions of the robot with its environment. Our objective is to create a method that will: 1) capture the pose of the robot on its own (for mimicking or pose re-targeting), 2) capture the reaching behaviour (as the task objective), and 3) capture the avoiding behaviour (obstacle avoidance). A representation that simultaneously captures these three kinds of interactions would provide a powerful tool for transferring, adapting, and planning robot motion for a wide range of reaching and manipulation tasks in environments with dynamic obstacles.

Assume a robot has N joints $q_i (i \in N)$ which control the

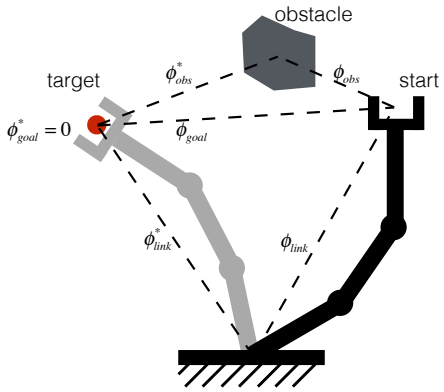


Figure 3: Desired and current relative distances. The target state $\Phi^* = [\phi^*_{link}, \phi^*_{goal}, \phi^*_{obs}]$ is constructed before execution, and start/current state $\Phi = [\phi_{link}, \phi_{goal}, \phi_{obs}]$ is computed during each control iteration. Not all of the relative distances are shown here, we only show one example for each type of distances. Note that ϕ_{obs} is not required if there is no obstacle.

robot's links. The planning problem is formalized as

$$\min_{q_1,\ldots,q_N} \sum_{i=1}^{N} c(q_i) \tag{1}$$

where $c(\cdot)$ is the state-dependent cost function. Typically,

$$c(q_i) = c_{pose}(q_i) + c_{goal}(q_i) + c_{obs}(q_i) + c_{rest}(q_i) \tag{2}$$

where $c_{pose}(\cdot)$ is the cost for maintain particular poses, $c_{goal}(\cdot)$ is the cost for reaching goal, $c_{obs}(\cdot)$ is the cost for collision avoidance and $c_{rest}(\cdot)$ is the cost for other constraints such as joint limits. However, we argue that since the first three costs represent such closely-tied behaviours, i.e. reaching target in a particular way while avoiding obstacles, one can unify them into one cost term in relative distance space. That is

$$c(q_i) = c_{\mathcal{D}}(q_i) + c_{rest}(q_i) \tag{3}$$

where $c_{\mathcal{D}}(\cdot)$ is the cost in relative distance space states that should solve pose re-targeting, reaching and avoiding in a coherent, consistent way. $c^t_{\mathcal{D}}(q_i) = \|\Phi^* - \Phi^t\|$ denotes the actual cost at time t, where Φ^* is the desired state in relative distance space, and Φ^t is the state at time t. The rest of this section explicitly explains how to compute Φ^* and Φ^t.

We attach M number of virtual points p_j $(j \in M)$ to the robot's links. These virtual points are usually joint and end-effector positions. We will also attach E number of virtual points p_e $(e \in E)$ to the centre of the obstacles in the environment. The relative distance space models the edges (ϕ_{jl}) between each p_j and p_l, $l \in M \cup E$ and $j \neq l$. Note that p_j always represents a robot link and p_l represents another object (other robot links, targets, obstacles), meaning that we model three different types of distances (Figure 3):

1) $\phi_{jl} = \phi_{link}$, if p_l is a point on different robot link.
2) $\phi_{jl} = \phi_{goal}$, if p_l is a point on target.
3) $\phi_{jl} = \phi_{obs}$, if p_l is a point on obstacle.

Note that we ignore the fourth type where $j, l \in E$, since the edges between two environmental objects are not controllable.

We define the relative distance between robot links as

$$\phi_{link} = w_{jl}\|p_j - p_l\|, \tag{4}$$

where w_{jl} is the weighting factor of the edge between p_j and p_l. For ϕ_{goal} and ϕ_{obs}, the relative distance can be Euclidean, e.g. $\phi_{jl} = w_{jl}d_{jl}, d_{jl} = \|p_j - p_l\|$. However, this will cause a series of problems. For example, distant targets will have a dominant influence on the motion. We can apply a non-linear growth model $\psi(j,l)$ on the distance metric to generate a smoother and more robust behaviour for targets and obstacles:

$$\phi_{jl} = w_{jl}\psi(j,l). \tag{5}$$

Different non-linear models can be applied here. In general, from a reaching and avoiding point of view, distant obstacles should not affect the robot, and distant target should not introduce unacceptable large effort. An inverse exponential model has the property that starts from the origin and quickly converges to a maximum value, based on which here we show one possible model for handling the interactions with targets and obstacles

$$\psi_{jl} = 1 - e^{-kd_{jl}}, \tag{6}$$

where $k > 0$ is a constant. The relative distance increases exponentially with d_{jl}, and converges to a maximum value 1 (ϕ_{jl} converges to w_{jl}). A distant obstacle ($d_{jl} \gg 0$) will not affect the robot if we set $\phi^*_{obs} = w_{jl} = w_{safe}$, i.e. $\phi^*_{obs} \approx \phi_{obs} = w_{safe}$, where w_{safe} is a non-negative constant. For reaching task, we set $\phi^*_{goal} = 0$, meaning that a distant target can only introduce a prescribed maximum effort to the system, i.e. $\|\phi^*_{goal} - \phi_{goal}\| \leq w_{jl}, \forall d \geq 0$.

In order to solve a motion transfer, adaptation and planning problem, we require the current state Φ^t, which we compute using Equations 4-6, and a desired state

$$\Phi^* = [\phi^*_{link}, \phi^*_{goal}, \phi^*_{obs}] \tag{7}$$

ϕ^*_{link} constrains robot poses that can be used for imitation problems, where the reference value ϕ^*_{link} can be computed from demonstration data. Minimizing the difference between the demonstrated and current relative distances will then result in transferring the motion based on the relative distances between the links. However, from target reaching point of view, the robot pose is often used as a secondary task, along side a primary reaching task, or it is not used at all. In this case, the relative link distance term can be ignored entirely. ϕ^*_{goal} is usually set to zero for reaching tasks. One can also set ϕ^*_{goal} to other values, e.g. keeping the end-effector and target with particular distance. $\phi^*_{obs} = w_{safe}$, as discussed earlier.

We construct the desired relative distance space target Φ^* by combining all three distance terms: ϕ^*_{link}, ϕ^*_{goal} and ϕ^*_{obs}. The state is, however, only valid if we keep updating the positions of the links, obstacles and target. We use an operational space controller to track the changes in the environment. For this we require the Jacobian of the relative distance space.

First we compute end-effector Jacobian of the points p_l using standard kinematics tools as

$$\mathbf{J}^{\mathit{eff}} = \frac{\partial \Phi^{\mathit{eff}}(\mathbf{q})}{\partial \mathbf{q}} \in \mathbf{R}^{3M \times N}, \tag{8}$$

where $\Phi^{\mathit{eff}}(\mathbf{q})$ is the joint space to end-effector space forward map. Our goal is to find the Jacobian between relative distance space and joint space

$$\mathbf{J} = \frac{\partial \Phi(\mathbf{q})}{\partial \mathbf{q}} = W\frac{\partial \Psi}{\partial \mathbf{q}} \in \mathbf{R}^{X \times N}, \tag{9}$$

where $X = \frac{(M+E)(M+E-1)}{2}$, W is the weighting matrix and $\Psi = [\psi_{jl}], j \in M, l \in M \cup E$. The distances between two obstacles or obstacle and target are not considered, so the number of controllable distances is X.

$$J_{x,i} = \frac{\partial \phi_{jl}}{\partial q_i} = w_{jl}\frac{\partial \psi_{jl}}{\partial q_i}, \tag{10}$$

where $x \in X$, and $\partial\psi_{jl} \in [\frac{\partial \psi_{link}}{\partial q_i}, \frac{\partial \psi_{goal}}{\partial q_i}, \frac{\partial \psi_{obs}}{\partial q_i}]$ depend on edge types. If pose retargeting is required,

$$\frac{\partial \psi_{link}}{\partial q_i} = \bar{d}_{j_1 j_2} = \frac{(p_{j_1} - p_{j_2}) \cdot (J^{\mathit{eff}}_{j_1,i} - J^{\mathit{eff}}_{j_2,i})}{d_{jl}}, \tag{11}$$

otherwise $\frac{\partial \psi_{link}}{\partial q_i} = 0$. Here, $\cdot$ is the dot product, and $J^{jnt}_{j,i} \in \mathbf{R}^{3\times 1}$ is the position Jacobian of point p_j w.r.t. the joint i.

For target reaching and collision avoidance, $\phi_{jl} \in \{\phi_{goal}, \phi_{obs}\}$, the first derivative of Equation 6 yields

$$\frac{\partial \psi_{jl}}{\partial q_i} = k\bar{d}_{jl}e^{-kd_{jl}}, \tag{12}$$

where $\bar{d_{jl}}$ is the relative distance Jacobian to end-effector space

$$\bar{d}_{jl} = \frac{(p_j - p_l) \cdot J^{\mathit{eff}}_{j,i}}{d_{jl}}. \tag{13}$$

Note that the Jacobian entries for goal and obstacles are the same, however, since they have different desired value, $\phi^*_{obs} \neq \phi^*_{goal}$, the effect of their Jacobian entries are different.

Now we have the desired relative distance space state Φ^*, current state Φ^t and the Jacobian that are required by the cost function (Equation 1 and 3). In general, this problem can be solved by any optimization based planners. However, from a real-time implementation point of view, we choose a Jacobian-pseudo-inverse IK type controller due to its simplicity and efficiency. We discuss the performance in Section III-B.

III. Incremental Planning Structure

For the obstacles which are known in advance, their relative distances can be encoded during planning phase. However, when we deal with unexpected obstacles, such as humans walking into the workspace of the robot, we have to modify the distance relationship space on the fly in order to avoid the costly replanning. This will involve adding and removing obstacle vertices, as illustrated in Figure 4.

A. Incremental Planning Structure

Assume we have a desired alternate space target

$$\Phi^* = [\phi^*_0, \phi^*_i, \ldots, \phi^*_{M+E}] \in \mathbb{R}^{(M+E)\times(M+E)} \tag{14}$$

where $M + E$ is the number of vertices (links, obstacles and targets combined), and

$$\phi^*_i = [\phi^*_{i0}, \phi^*_{i1}, \ldots, \phi^*_{i,M+E}]^T \tag{15}$$

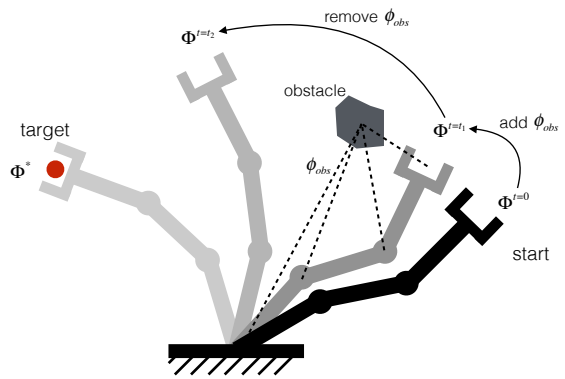


Figure 4: Incremental planning structure, i.e. modifying the relative distance state online. The desired state Φ is computed without obstacle. The robot starts from a state ($\Phi^{t=0}$) with no local obstacle. Obstacle is detected at time t_1, new entries will be added into both Φ^* ($\phi_{obs}^{t=t_1} = w_{obs}$) and $\Phi^{t=t_1}$. At time $t_2 > t_1$, the obstacle is no longer close to the robot, the entries for the obstacle are removed.

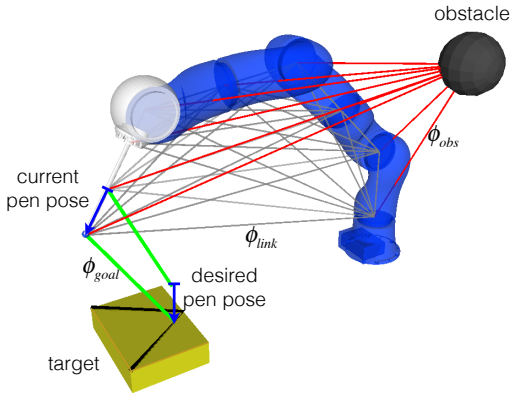


Figure 5: LWR mock-up welding experiment setup. The LWR robot arm is mounted with a laser pen, the task is to use the laser pen to weld along the target surface. We add an additional virtual point along the pen, and set its desired position to be above the real laser tip, such that the robot will keep the pen orthogonal to the surface. The lines represent the current relative distances, ϕ_{link} in grey, ϕ_{obs} in red, and ϕ_{goal} in green.

is the vector that describes the desired distances between vertex i and all other vertices. When a new obstacle k is detected, the original goal Φ^* is no longer valid. We want to have a new target in alternate space in the form of

$$\Phi^*_{new} = \begin{bmatrix} \Phi^* & \phi_k^* \\ \phi_k^{*T} & 0 \end{bmatrix} \in \mathbb{R}^{(M+E+1)\times(M+E+1)}. \quad (16)$$

Note that Φ^* is still valid since it only depends on old vertices, meaning that we only need to compute ϕ_k^* and reuse the old plan as part of the new plan. The key to achieving real-time implementation is to minimise online computation. In our case, it is straight forward to get ϕ_k^* and modify the plan without heavy computation. From Equation 15 we have

$$\phi_k^* = [\phi_{k0}^*, \phi_{k1}^*, \ldots, \phi_{k,M+E}^*]^T \quad (17)$$
$$= [w_{safe}^0, w_{safe}^1, \ldots, w_{safe}^{M+E}]^T \quad (18)$$

where $w_{safe}^m, m \in M \cup E$ are the distances that the obstacle needs to keep from other objects (robot links). In practice these distances may vary based on the shape of the links and obstacle, their velocities, etc. When we add new obstacles, we only need to resize the distance space, keeping the old plan for the existing vertices, and fill in ϕ_k^* to get a new plan Φ^*_{new}. We can continuously add or remove vertices to the distance space during execution without the need to perform replanning.

B. Complexity Analysis

In this section we analyse the computational complexity of our approach. In our experiments we assume that the state is valid when the robot's pose is collision free and the end-effector gets closer to or is at the target position. The computation can be separated into two main steps: *construction phase* and *solving phase*. In the construction phase, we compute the current relative distance space state, Φ, and the relative distance space Jacobian, $\mathbf{J}$. The desired state Φ^* is calculated once before execution, and it will get modified when new obstacles are detected, also, during construction phase. In solving phase, we solve operational space control problem using Equation 1 and 3. The order of complexity of the method is $O(\frac{1}{2}M(M+E))$ in the worst case where we consider edges between all robot links and all obstacles and targets. Furthermore, if we consider a reaching problem, without the pose re-targeting, we can omit the edges between the robot links entirely, which reduces the computational complexity to $O(ME)$.

We analyse the computational time of our method with different total number of edges $X = M + E$ on a reaching problem. The increase of the construction time is negligible compared to the increase of the solving time. An evaluation of maximum controlling speed with different X is illustrated in Table I. The result suggests that the proposed method can solve the adaptation problem in most common scenarios very efficiently. We have used a 3GHz Intel Core 2 Quad CPU.

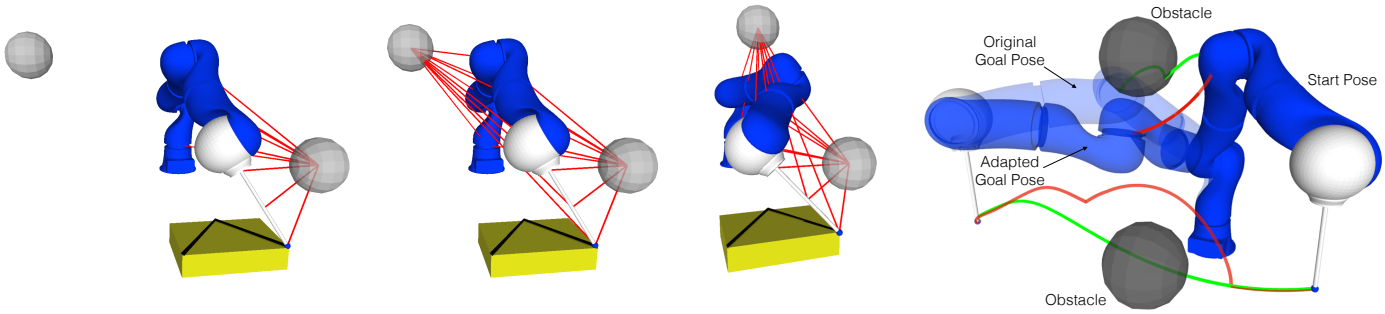
Table I: Maximum controlling frequency with different space size $X = M + E$. For example, $X = 10$ can be used for single arm ($M = 7$) robot in simple environment (1 target and 2 obstacles, i.e. $E = 3$). In contrast, $X = 20$ should be more than enough for single arm robot in most complex environment (e.g. KUKA LWR, Figure 5), $X = 30$ should be sufficient for a dual-arm upper body robot (e.g. Baxter robot, Figure 9), and $X = 50$ for humanoids.

Space Size (X)	10	20	30
Control Speed (Hz)	750 ± 50	630 ± 20	600 ± 30
Space Size (X)	50	70	100
Control Speed (Hz)	490 ± 20	350 ± 10	215 ± 5

IV. Experiments

We evaluate our approach with two different experiments. The first experiment uses a 7 DOF KUKA LWR robot arm to mock-up a dynamic welding task (Section IV-A), and the second experiment is a liquid pouring task in a close robot-human interaction scenario on a 14 DOF dual-arm Baxter robot (Section IV-B). The experiment setup will be detailed in each section accordingly. The tasks are implemented using EXtensible Optimization Toolset (EXOTica)[1], which is a planning

[1]EXtensible Optimization Toolset (EXOTica) planning framework. http://wcms.inf.ed.ac.uk/ipab/slmc/research/EXOTica



(a) Incremental planning, adding new obstacle information to the original plan.

(b) Trajectory Adaptation.

Figure 6: (a): A new obstacle (unconnected one) is detected during execution, then new relative distances will be added into the original state when it gets close. (b) Example of trajectory adaptation, where the green lines are the original (end-effector and elbow) trajectories and the red ones are the adapted trajectories under multiple obstacles constraint.

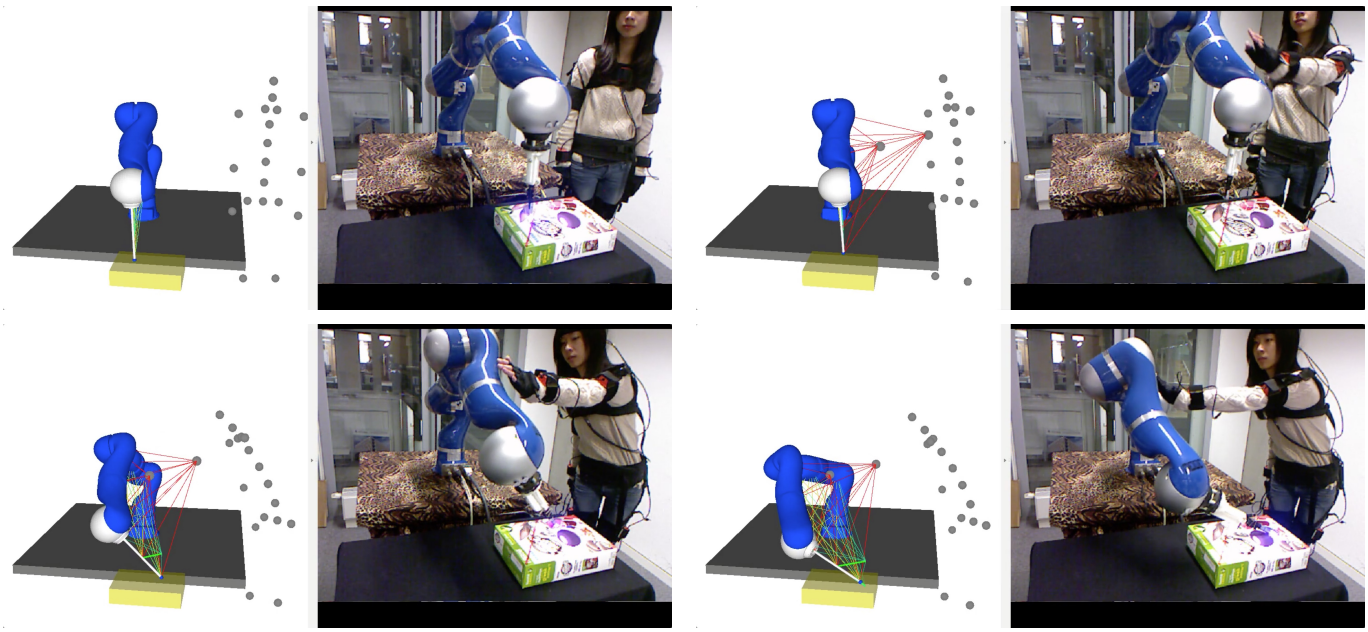


Figure 7: Experiment results on LWR robot hardware. Each figure contains two sub figures, where the right one is the real world environment, and left is the corresponding simulated environment. Top left figure shows the original task without (close) obstacle; top right figure shows that new obstacles are added into the relative distance space state on the fly; bottom figures show the avoiding behaviour when the human gets close.

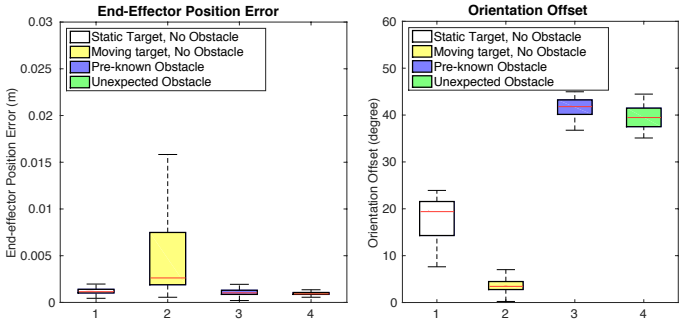
framework for solving robotics motion planning problems.

A. *LWR Mock-up Welding Task*

In the first experiment we aim to show that an accurate manipulation task can be accomplished by only using relative distance based representation. The experiment setup is illustrated in Figure 5. In addition, we add an extra virtual end-effector along the physical end-effector's z axis and a corresponding virtual target to keep the laser pen orthogonal to the target plane. We set different weighting factors for the laser tip, virtual point and obstacles ($w_{tip} > w_{obstacle} \gg w_{virtual}$), so that the pen will be kept orthogonal to the target if there is no other constraints. The orientation will be sacrificed to ensure physical end-effector position and collision free constraints in presence of obstacles. We use a real-time object pose recognition and tracking framework [18] to detect and track the target.

The robot links' collision bodies are represented by a set of spheres with radius of 7cm and the safety threshold $w_{safe} =$ 5cm. The number of ϕ_{obs} can vary based on the number of obstacles. Since this is a reaching and avoiding problem, the robot pose constraint is not considered, i.e. $\phi^*_{link} = \phi_{link} = 0$.

We run the experiment with four different scenarios: 1) static target without obstacles, 2) moving target without obstacles, 3) static/moving target with dynamic obstacles present before planning started, and 4) static/moving target with unexpected obstacles present during execution. We record the laser tip position error and the pose offset over a same time duration across the four scenarios. The position error is illustrated in Figure 8a, where the y axis is the Euclidean error between real laser tip position and desired ones. We can see that during all experiments, the errors of the laser tip are very small (1.1 ± 0.44mm). The error during the second scenario is larger due to the fast movement of the target. In the presence



(a) End-effector position error. (b) Laser pen orientation offset.

Figure 8: (a): In all four scenarios, the laser tip errors are very small (1.1 ± 0.44mm). Note that the error in the second scenario is larger due to the random fast movement of the target. (b): Pen orientation offset that keeps the laser orthogonal to the target surface, it will be sacrificed to maintain the end-effector position in the presence of obstacles. The offsets in scenario 3 and 4 are at the same level, the slight difference is due to random noise.

of obstacles, the orientation is sacrificed to ensure laser tip position and collision free constraints (Figure 8b). Examples of adapted motion in simulation is shown in Figure 6. Note that the robot can adapt not only end-effector trajectory, but also the trajectories of all links. Figure 6b shows an example of adapting end-effector and elbow trajectories simultaneously.

In real world experiments, human subject's motion are tracked in real-time using XSENS motion tracking system. A set of obstacles are created to represent the human subject, as shown in Figure 7. Each figure consists two subfigures, left and right ones, where the right one is the snapshot of the real world environment and left one is the corresponding simulated environment. These results show that we can accomplish accurate manipulation tasks under dynamic obstacle constraints.

B. Baxter Liquid Pouring Task

In this experiment, we evaluate the scalability of our method on a 14 DOF Baxter robot (Figure 9). The robot holds a cup with one hand, keeps it horizontal and uses the other hand to grasp a bottle to simulate a water pouring task. We use one big unified relative distance space state to encode the tasks for both hands as well as the possible collision avoidance constraints. Similar to the laser pen orientation in last experiment, here we add extra virtual point (centre of the bottle) to maintain the cup and bottle's orientation. Two goal distances are specified, i.e. $\phi^*_{goal}(tip) = 0$ and $\phi^*_{goal}(centre) = d_c$, where the first one is used to make sure the bottle's tip is at correct position and the later one is used to control the pouring angle. In practice, we set d_c to zero, meaning that the bottle should be kept orthogonal to the cup.

The experiment consists of three different scenarios: 1) perturb the robot from the side on which the hand is holding the bottle; 2) perturb the robot from the side on which the hand is holding the cup, where the cup's desired position is not fixed, meaning the robot can shift both arms to avoid the human; and 3) move the cup randomly by moving the robot's hand. The task here is to keep the bottle tip directly above the cup and keeps the bottle as orthogonal as possible. The result (Figure 10a) shows that in most cases the bottle can

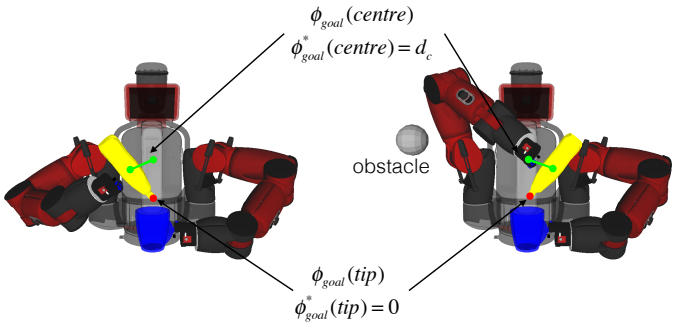
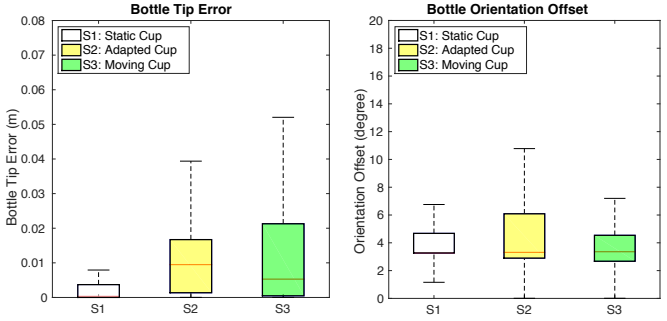


Figure 9: Baxter water pouring experiment. The robot holds a cup with one hand and a bottle with the other hand. The main task is to keep the bottle tip above the cup and avoid obstacles, the secondary task (lower weighted) is to keep a certain orientation between the bottle and the cup. The poses in both figures are equivalent in relative distance space, (i.e. $\phi^*_{goal}(tip) = 0$, $\phi^*_{goal}(centre) = d_c$, where d_c is a constant (the length of the solid green line). In practice we set $d_c = 0$, i.e. keep the bottle orthogonal to the cup.



(a) Bottle tip position error. (b) Bottle orientation offset.

Figure 10: (a): The bottle tip position error. The error in S3 is larger due to the fast random movement of the cup that the robot failed to follow. (b): The orientation offsets from the desired pouring angle, the offsets are at a similar level across different scenarios.

be placed in the correct place with an acceptable mismatch (0.1cm-1cm). The error in the third scenario is larger due to the fact that the robot can not follow the cup when it is moved by human with high velocity. Orientation offsets are similar across three scenarios as shown in Figure 10b, which suggests that the robot is able to "pour" the liquid into the cup with an acceptable pouring angle under dynamic obstacle constraint. Examples of adapted motions under each scenario are illustrated in Figure 11. The video of the experiments is available at https://youtu.be/A1dhiLyLo5U.

V. Discussion

The experiments show that the proposed method can be used for solving accurate manipulation tasks with moving target and dynamic obstacles. The robot can avoid not only the existing obstacles, but also obstacles that are arbitrarily added into the scene during the execution.

Our method has a few limitations, one of which is the local minima problem. Although relative distance space plan can adapt to environmental changes, the robot still fails to converge to the target in some situations (e.g. trapped in large concave or long convex obstacles), where a global replanning is required to find another valid plan. Another limitation is

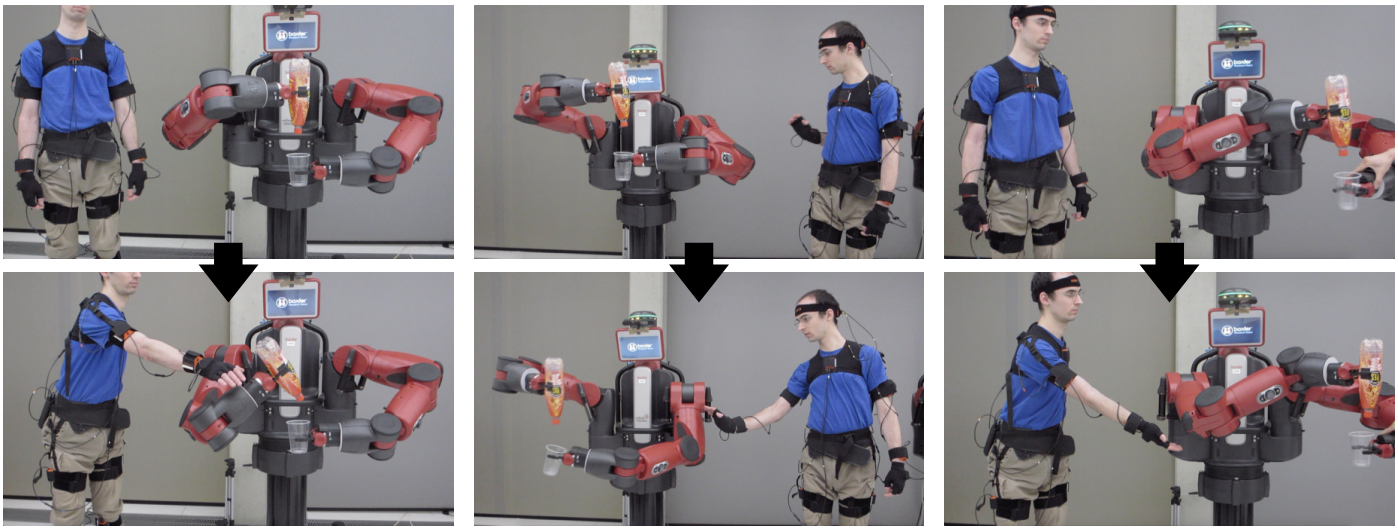


Figure 11: Experiment 2: Baxter robot water pouring task. In the first scenario (column 1), the human subject only disturb the robot from the left, the cup's position is fixed, the robot needs to adapt its right arm (with bottle) to fill the water while avoiding human; in the second scenario (column 2), the robot gets perturbed from the right, which means it needs to move its left arm (with the cup) to another collision free pose, and meanwhile the relationship between the bottle and cup needs to be maintained; in the third scenario (column 3), the cup's position is controlled by another human subject, the robot needs drive its right hand with the bottle to follow the cup while avoiding the human.

that, the method creates relative distance space plans based on end-effector space or configuration space trajectories. One may argue that if there exists a way to provide the original trajectory in relative distance space, then the proposed method can be used directly.

VI. Conclusion and Future Work

This paper presents a novel method to encode pose retargeting, reaching and avoiding problems into an unified term, the relative distance space. By constructing this space, we model the relative distances between robot links, targets and obstacles. An incremental planning structure is also proposed that allows the planner to manage the relative distance space in a dynamic way, which gives us the ability to interact with unexpected obstacles. We evaluated the system on two platforms with different tasks, showing that the proposed method can be applied to various realistic applications.

Future work will be focused on detecting local minima in relative distance space and then finding a new feasible plan, e.g. global replanning or selecting an alternate plan from some pre-calculated dictionary of initial plans.

References

[1] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

[2] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime Motion Planning using the RRT*," in *ICRA*, pp. 1478–1483, May 2011.

[3] J. Pan and D. Manocha, "GPU-based parallel collision detection for real-time motion planning," in *Algorithmic Foundations of Robotics IX*, pp. 211–228, Springer, 2011.

[4] C. Park, J. Pan, and D. Manocha, "High-DOF Robots in Dynamic Environments Using Incremental Trajectory Optimization," *IJHR*, 2014.

[5] S. Schaal, "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics," in *Adaptive Motion of Animals and Machines*, pp. 261–280, Springer, 2006.

[6] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *Humanoids*, pp. 91–98, 2008.

[7] P. Englert and M. Toussaint, "Reactive phase and task space adaptation for robust motion execution," in *IROS*, pp. 109–116, 2014.

[8] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Robotics and Automation*, pp. 500–505, 1985.

[9] K. B. Kaldestad, S. Haddadin, R. Belder, G. Hovland, and D. A. Anisi, "Collision avoidance with potential fields based on parallel processing of 3D-point cloud data on the GPU," in *ICRA*, pp. 3250–3257, IEEE, 2014.

[10] S. M. Khansari-Zadeh and A. Billard, "A dynamical system approach to realtime obstacle avoidance," *Autonomous Robots*, vol. 32, no. 4, pp. 433–454, 2012.

[11] E. S. Ho and T. Komura, "Character motion synthesis by topology coordinates," in *Computer Graphics Forum*, vol. 28, pp. 299–308, 2009.

[12] E. S. Ho, T. Komura, and C.-L. Tai, "Spatial relationship preserving character motion adaptation," *ACM Trans. Graph.*, vol. 29, no. 4, p. 33, 2010.

[13] R. A. Al-Asqhar, T. Komura, and M. G. Choi, "Relationship descriptors for interactive motion adaptation," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 45–53, ACM, 2013.

[14] E. Ho, T. Komura, S. Ramamoorthy, and S. Vijayakumar, "Controlling humanoid robots in topology coordinates," in *IROS*, pp. 178–182, 2010.

[15] V. Ivan, D. Zarubin, M. Toussaint, T. Komura, and S. Vijayakumar, "Topology-based Representations for Motion Planning and Generalisation in Dynamic Environments with Interactions," *IJRR*, vol. 32, pp. 1151–1163, 2013.

[16] E. Ho and H. Shum, "Motion adaptation for humanoid robots in constrained environments," in *ICRA*, pp. 3813–3818, 2013.

[17] T. Nierhoff, S. Hirche, W. Takano, and Y. Nakamura, "Full body motion adaption based on task-space distance meshes," in *ICRA*, pp. 1865–1870, 2014.

[18] K. Pauwels, V. Ivan, E. Ros, and S. Vijayakumar, "Real-time object pose recognition and tracking with an imprecisely calibrated moving RGB-D camera," in *IROS*, pp. 2733–2740, 2014.