The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Motion Planning of a Dual-Arm Mobile Robot in the Configuration-Time Space

Yi-Chih Tsai, Han-Pang Huang, *Member, IEEE*

Abstract—**The main objective of this paper is to develop an autonomous navigation system for robot arms so that they can operate in complex environments, such as kitchen, factory, etc. In order to solve the problem of planning in the dynamic environments, we proposed a CT-RRTs (Bi-direction RRTs in Configuration Time space) planner, which can make the robot arm reach goal successfully in the partially known dynamic environment while remains the greedy heuristics of the RRT-Connect. It is planned in the augmented state-time space of the robot arm configuration and the positions of the moving objects. Various techniques are used to accelerate planning and enhance its safety. We used the new planner to develop the dual-arm planner, and it can be used as the basis to solve higher level problems in motion planning. The experiments show that the proposed method can effectively resolve the dual-arm motion planning problem.**

I. Introduction

In recent years, the humanoid robot arm has been widely developed. The purpose of designing and making a humanoid robot is to help people in operations, or even to substitute for people in some dangerous jobs such as repairing machines, experimenting chemical treatment, achieving precision in work, etc. The robot with its arms can help us do many things in conjugation with its arms. For example, the fixed-base robot arms perform a variety of tasks, including assembling, welding, and painting. In particular, the multi-arm robot has been gradually applied to the production line or service robots [4, 5, 6, 14, 21]. However, the multi-arm robot is unlike a common humanoid robot arm, which is usually composed of six or seven degrees of freedom joints. It is even more complicated.

For robot applications in industry, e.g. automobile factory and production line, most trajectories for moving objects is near the workspace of the robot arm. A service robot usually consists of a mobile platform, two arms, head, and trunk. The mobile robot executes a given task in the human-robot co-existence environment. The robot may use its arms to grasp a book and return it to the bookshelf. This is a very difficult challenge because there are many kinds of uncertainty to be considered, such as sensor measurement errors, command input noise, timing errors, etc. Furthermore, the robot and the robot arms need to avoid obstacles while the robot is moving to execute the task. If the time information and the trajectory of moving objects are added to the planner, it can find the better path than planning in the state space.

In this paper, the motion planning of a dual-arm mobile robot in the configuration-time space will be proposed. Each arm is seven degrees of freedom, and is developed by our laboratory [22]. It can be shown that the proposed planner, CT-RRTs (Bi-direction RRTs in Configuration Time space), which takes the time information into account and remains the greedy heuristics of multi-RRTs, can find the better path in partially known dynamic environments.

II. Related Works

A Rapidly-exploring Random Tree (RRT) [2,13] is a data structure and algorithm that is designed for efficiently searching in non-convex high-dimensional spaces. The RRT-Connect [11], which is often used in the high dimension C-space (configuration space), has greedy heuristics that aggressively tries to connect two trees in the connect function, one from the initial state and the other from the goal. The configuration space concept can be extended to dynamic environments by incorporating an absolute notion of time as an additional dimension [1]. The resultant space is called the configuration-time space (CT-space). In dynamic motion planning, it is useful for the arms to avoid colliding with moving obstacles by making some predictions of their positions.

Many methods discussed above can be applied to CT-space, but seldom planners can work in the high dimensional *CT* space. In particular, as time goes forward, additional constraints are imposed on the validity of paths through the configuration-time space. Obviously, paths should be monotonically increased in the time dimension in order to prevent unrealistic time travel. Thus, the sampling based approaches are very popular in the high dimensional CT-space [9, 10].

This work is partially supported by the Department of Technology, Ministry of Economic Affairs, Taiwan R.O.C. under grants 96-EC-17-A-04-S1-054.

Yi-Chih Tsai is with National Taiwan University, Taiwan, (e-mail: ezi0613@gmail.com).

Han Pang Huang is a professor of Department of Mechanical Engineering, National Taiwan University, Taipei, Taiwan (e-mail: hanpang@ntu.edu.tw). He is currently the Director of Robotics Laboratory.

III. The Framework of CT-RRTs

In this paper, the CT-RRTs (Bi-direction RRTs in Configuration Time space) is proposed for planning a robot path in the dynamic environments. The planner is composed of RRTs Growth and Prune Trees. It is based on the bi-directions RRTs and improves its strength by modifying its function. It is performed in the CT space, and will not only preserve the RRT features but also plan a safer path, rapidly prune trees, and reduce the detour. The time information, cost, and risk estimates are added and new data structure is derived.

A. Inserting Time Information

In [19], the author adopted the distance measured from the leaf node to its parent to calculate time. If we know the node belongs to which layer, we can reckon the time of this node. While using RRT, only reckoning the time to check collision without adding time into configuration space is enough. However, it will become very complicated since we do not know time relationship between the nodes of each tree with its parent except the root of tree. It is impossible to reckon time of each node. The time of the root of each tree is very hard to define when the root is not at start or goal. Therefore, it is desired to simplify this problem by reducing the forest based planner to bi-direction trees.

We are inspired by [18] about estimating the time of goal. The minimum possible time $h_t(n_{start}, n)$ for traversing from the current start position n_{start} to any particular position n_{goal} can be computed by measuring the distance between n_{start} and n_{goal}. Since the length of each RRT step is set to the same value, we use the number of steps from current node to start to record the path length will not cause significant errors. The time of goal can be estimated by adjusting some parameters as

$$\text{Time of Goal} = w_d \cdot h_t(n_{start}, n_{goal}) + w_e \cdot (TimeError / num) \quad (1)$$

where w_d and w_e are the weight, and the *TimeError* is generated by RRT-Connect when failure is caused by time. It is very difficult to estimate time accurately since the period of reaching the goal depends on the complexity of the environment. Therefore, we must let the time of goal adjust by itself in each planning.

However, each RRT-Connect failure caused by time still has much useful information. If we can effectively use this information, the planner can rapidly find a collision-free path. In the CT-space, only the dynamic objects will change the position along the time axis. In other words, static objects are time-invariant. Therefore, if we can keep the path away from the dynamic objects, their influences will be greatly reduced and the RRT-Connect success ratio will be greatly enhanced.

For each RRT-Connect failure caused by the time, we will record the root of the path as n_{goal}. We add nodes from the path into start tree bit by bit and check if it is collision-free at the same time. These processes will stop until collision is detected or the node is n_{goal}. In this way, the CT-RRTs can maintain the RRT-Connect greedy heuristics since the start tree can rapidly close to n_{goal}. Table I is the pseudo code of this process, where the "timing" is the time of RRT extending one step from the line 4 in the function ReusePath.

TABLE I THE REUSEPATH ALGORITHM.

```
1.  STnode = Connected node from Start Tree
2.  GTnode = Connected node from Goal Tree
3.  while GTnode ≠ qgoal do
4.     GTnode.Time = STnode.Time + Timing;
5.     Predict_Dynamic_Obstacle(GTnode.Time);
6.     Update_Safety(GTnode);
7.     if Collision(GTnode) || GTnode.Safety > Threshold then
8.        return false;
9.     end
10.    Start Tree.addNode(GTnode);
11.    Start Tree.addLink(GTnode , STnode);
12.    STnode = GTnode;
13.    GTnode = GTnode.Parent;
14. end
15. return true;
```

B. Inserting Cost function

In order to make the trajectory smoother and safer, some heuristic functions or cost functions are added. In other words, the cost function will make the tree grow toward a better area (i.e. less cost area). The search function has not only the distance factor but also the cost factor as

$$f(q) = dist(q)^2 + \cos t(q)^2 \quad (2)$$

where q is sampling node. Then, it is desired to find the minimum *f*(q) from the tree rather than finding the minimum *dist*(q). Furthermore, the cost function can consider both distance function and safety function.

$$\cos t = w_d \cdot PathLength(n) + w_s \cdot safety \quad (3)$$

1) Distance cost

The distance cost can make the path smoother because the tree will grow toward the less cost area. In other words, when the tree is expanding, the nearest-neighbor search function will choose the node which has the shorter path. When the weight of the distance cost is larger, the path will become more linear because the redundant twists and turns will cause higher cost. In order to reduce the computation time, we simply record the path length by the number of steps from current node to the start. Although increasing the weight of distance cost can make the path smoother, if it is too large, it will make RRT hardly find a path.

2) Safety cost

We use the constant velocity model with a Gaussian uncertainty to predict the path of the moving obstacles. Here, the Kalman filter is used to predict the path and estimate the uncertainty. For simplicity, we assume that all moving objects are with constant velocity models.

$$\hat{x}(k+1\mid k) = F(k)\hat{x}(k\mid k) + G(k)u(k) + v(k) \tag{4}$$

where

$$x(k\mid k) = \begin{bmatrix} x_X \\ x_Y \\ x_Z \end{bmatrix}, F(k) = I, u(k) = \begin{bmatrix} v_X \\ v_Y \\ v_Z \end{bmatrix}, G(k) = \Delta t \cdot I \tag{5}$$

where the control input u(k) is the velocity of the moving objects, and the vector v(k) is the noise process caused by control input. Since the matrix *F(k)* is an identity matrix, the prediction process, Eq. (6) can be rewritten as Eq. (7).

$$P(k+1\mid k) = F(k)P(k\mid k)F(k)^T + V(k) \tag{6}$$

$$P(k+1\mid k) = P(k\mid k) + V(k) \tag{7}$$

v(k) relates to the control input and uncertainty will be enlarged with time increasing. It is reasonable to assume that the covariance matrix *V*(k) relates to time and velocity.

$$V(k) = \frac{\Delta t}{v}\begin{bmatrix} v_X^2 & v_X v_Y & v_X v_Z \\ v_X v_Y & v_Y^2 & v_Y v_Z \\ v_X v_Z & v_Y v_Z & v_Z^2 \end{bmatrix}, \text{where } v = v_X^2 + v_Y^2 + v_Z^2 \tag{8}$$

This probabilistic model $P(k+1\mid k)$ can be used to represent the uncertainty. Then an uncertain configuration is represented by the estimated position $\overline{x}$ and its covariance matrix $P(k+1\mid k)$ [12]. An uncertain position is shown geometrically as an ellipsoid [16] whose center is $\overline{x}$:

$$(x-\overline{x})^T P(k+1\mid k)^{-1}(x-\overline{x}) \le \sigma^2 \tag{9}$$

where σ is the multiple of standard deviation or the confidence threshold.

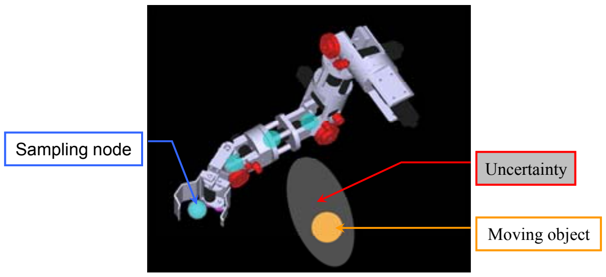


Fig. 1 Sampling nodes for safety cost

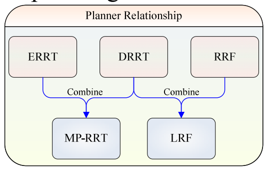
As x closes to the prediction position ($\overline{x}$), we will obtain smaller value from Eq. (9). Therefore, we use this point to define the safety cost. However, how do we make the arm become a point in order to calculate the safety cost? As shown in Fig.1, we choose some nodes to replace the whole arm. But if we choose too many nodes, it will enormously increase the computation time. If the nodes are sampled in the upper arm, it does not make sense because of the small workspace. Therefore, we choose tree nodes in the forearm to define the safety cost:

$$\mathit{Safety}\ \text{Cost} = \sum_i \frac{1}{(x_i - \overline{x})^T P(k+1\mid k)^{-1}(x_i - \overline{x})} \tag{10}$$

3) Re-planning

Generally, dynamic motion planning is similar to static motion planning, except that the planning steps will be changed with system and world update steps. Applications for dynamic planning include navigation or manipulation planning in a world with unknown or moving obstacles [20]. Between planning iterations, several possible events may have occurred simultaneously: (1) the robot may have moved, and have discovered new obstacles to avoid, (2) the goal may have moved, (3) previously sensed obstacles may have moved [20]. In these situations, the planner should plan a different path that can lead a robot or a manipulator to reach the goal safely.

Therefore, if we can efficiently reuse the information of configuration space from the last planning, it will save a lot of computation time and resource in the re-planning process. Especially, the planer is expected to find a path as soon as possible in anytime planning and dynamic environments. In this section, we will address several kinds of re-planning methods, ERRT[3], DRRT[7], RRF[15, 17], MP-RRTs[20], LRF[8], and compare their pruning time and the amount of remained information. The comparison of those algorithms is summarized in Fig.2. Clearly, DRRT is the most suitable one for our planner in that it does not generate other sub-trees and can remain the sub-branches from previous plan. Thus, the DRRT is used in the planner for fast re-planning.



(a)

	Remain Info.	Pruning time	Forest based
ERRT	Few	Middle	No
DRRT	Middle	Fast	No
RRF	Much	Slow	Yes
MP-RRT	Much	Middle	Yes
LRF	Much	Fast	Yes

(b)

Fig. 2 The comparison table and relation diagram for various algorithm

C. *Anytime Planning*

While navigating among moving obstacles, it is important to know how long a query will take so that it will be possible to expect when a plan may become available for execution. If we want to plan the path that starts at t_{start}, then we cannot start computing the path at t_{start}, because the computation is immediately outdated [1]. Therefore, we must reserve Δt time to plan the path in order to overcome this problem. The path and the estimating of the moving objects should begin at the $t_{start}+\Delta t$.

In such a case, it will be unwise to set the expected time too large, which will render the system unresponsive and waste all the computation resource left when a query finishes early. On the other hand, no query should be allowed to fail if safety is important. Thus, the planning period cannot be set lower than the time consumed of the worst-case, which can be much longer than the average planning time for random path planners.

The "anytime" planning algorithms return a viable but probably highly sub-optimal solution at any time given a relatively short initial planning time, and given more time, a better solution will be planned. For these algorithms, the planning period can be set to any value larger than the longest initial planning time. And as they incrementally improve on their solutions while allowed, no computation resource is wasted when an initial solution is found early [8].

D. Framework of CT-RRTs

In summary, the framework of the proposed CT-RRT consists of two main parts: RRTs Growth and Prune Trees. The RRTs Growth contains the cost computing, and RRT-connect. It generalizes bi-directional RRT by attempting to grow multiple pairs of trees into each other. First, a random sample q_{rand} is generated; see line 2 in Table II.

TABLE II THE RRTS GROWTH ALGORITHM.

```
1.  for Each Tree Ti do
2.      q_rand = RandomSample();
3.      q_near = NearestNeighbor(q_rand , T_i , d_near);
4.    if Extend (T_i, q_near , q_rand ) then
5.      Compute Safety(q_new);
6.      Compute cost(q_new);
7.      Ti.addNode(q_new);
8.      Ti.addLink(q_near , q_new );
9.    end
10.   for Each Tree Tj , such that i ≠ j do
11.     q_near,j = NearestNeighbor(q_new , T_j );
12.     if Connect(T_j , q_near,j , q_new ) then
13.       MergeTrees(T_i, T_j );
14.       return T
15.     else if Time cause fail then
16.         TimeError += (q_new.Time - q_near.Time);
17.         ErrorNum++;
18.         ReusePath(q_near , q_new);
19.     end
20.   end
21. end
```

For each tree T_i (T_{start} or T_{goal}), we try to extend T_i to q_{rand} via some node q_{new}. In the extend function, we will estimate the time of q_{new} and predict the moving objects to check the collision. If the extend process is successful, we will add the q_{new} to the tree and compute its cost in order to search optimal node in the next NearestNeighbor() step. If the extend process is failure which is caused by time, we will reuse its information to extend the start tree toward the goal; see lines 15-19 in Table II. Finally, if some nodes are close enough, the T_{start} and T_{goal} are merged and the path is found.

TABLE III THE PRUNE TREES ALGORITHM.

```
1.  n_start = n_current;
2.  Update(n_start.Time);
3.  estimate time of ngoal;
4.  n_i = n_goal.Child;
5.  while n_i = NULL
6.    Update(n_i.Time);
7.    if n_i.Time - n_start.Time < Threshold then
8.      RemoveNode(n_i);
9.    end
10.   if n_i.Parent = INVALID then
11.     RemoveNode(n_i);
12.   end
13.   if n_i is invalid then
14.     RemoveNode(n_i);
15.   end
16.   n_i = n_i.Child;
17. end
```

The Prune Trees contains TrimRRT() of the DRRT and time estimating. Pruning the tree involves stepping through the RRT in the order in which nodes are added and all child nodes whose parent nodes are invalid are marked as invalid [7]. This effectively breaks off branches where they directly collide with new obstacles and removes all nodes on these branches. In addition to remove the invalid node by moving objects, the Prune Trees also filter the nodes of which time is invalid or unreasonable, see lines 6-8 in Table III. If we want to be careful about collision with moving objects in future, we can set a threshold to filter the nodes with high cost.

TABLE IVI THE MASTER ALGORITHM.

```
1.  InitialRRTs();
2.  RRTsGrow();
3.  while Robot has not reached qgoal do
4.    Check path collision and robot localization;
5.    if solution path contain a invalid node then
6.      PruneTrees();
7.      RRTsGrow();
8.      Update task paths;
9.    end;
10.   Send robot next waypoint for active task;
11. end;
```

The robot and planner must work together to accomplish a task when the robot's motion and planning are coupled. This requires an interface much like a feedback loop between the planner and the robot [8]. The robot specifies its current or future location by sensing the environment, and predicts the moving objects to check whether the task path is valid or not. Through growth and pruning steps, the planner attempts to change in the environment or task, and reports an updated command to the robot. Table IV shows the main loop of the planner to execute tasks.

IV. MOTION PLANNING FOR DUAL-ARM

The motion planning for a dual-arm mobile manipulator is a very complicate and difficult problem, because it has to consider the mutual interference of two arms. While the workspace of two arms is highly overlapped, they have to avoid each other and accomplish the task at the same time. Fortunately, the overlapping area of workspace between two arms is often much smaller than the workspace of a single arm.

Two arms should work simultaneously, and should not be broken by each other. The time schedule for dual-arm planner is given in Fig. 3. The single arm planner of the left arm will consider the trajectory of another arm, the right arm, from the time when the left arm starts to move until the right one stops, and vice versa. As for cooperative handling problems, the time schedule should be modified. The arms begin to move simultaneously after the planning of each arm is finished.

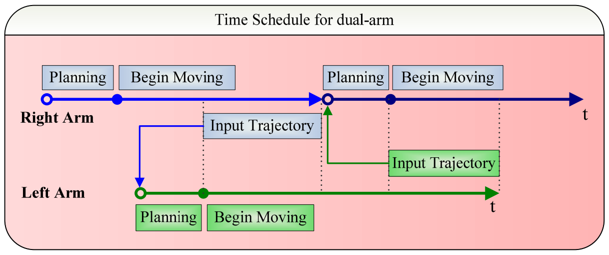


Fig. 3 The time schedule for dual-arm planner

The proposed CT-RRTs planner is very suitable for the applications in the known environments. It can consider the moving objects trajectories and plan a collision-free path, especially when the trajectories are accurately known.

The probability of collision between two arms is very low, when the n_{start} and n_{goal} are not in the overlapping workspace. Therefore, we divide the planning into two parts: two arms paths pass and do not pass the overlapping area. Then the problem is reduced to dependent part and independent part.

In the dependent part, we treat one arm as the moving object, and then the other arm plans the trajectory to avoid collision. According to the distance length from start to goal and the environment complexity, we decide which arm is regarded as the moving object. Then, we use the proposed planner to plan the path in the known dynamic environments. On the other hand, in the independent part, we can use parallel processing to accelerate the planning speed. The flowchart in Fig.4 shows the planning process.

V. Experiment Results

We have implemented the approach on a desktop computer with a 2.8 GHz duo Core processor and 2 GB DDR2-800 RAM. The two seven degrees of freedom robot arms are developed by our laboratory [22]. The scenario is that a mobile robot with two 7-DOF robot arms performs tasks in a complex configuration space. Several scenarios will be conducted to demonstrate the performance of the proposed planner.

A. *"Moving Objects" Scenario*

The first scenario is where three balls are moving with constant velocity in the pane, and a robot arm try to grasp the cup on the table, as shown in Fig. 5. The constant velocities of these three balls are about 0.28~0.43 m/s, and their speeds are faster than the robot arm. In this simulation there are two important assumptions about environment. The first is that we roughly know the velocity of balls. The second is that we have accurate model and position for static obstacles. Fig. 5 indicates that the simulation result is very successful. Practically, sometimes the planning failure will happen in the many times experiments. However, we can use the safety weight to avoid this situation at expense of increasing the process time of the planning.

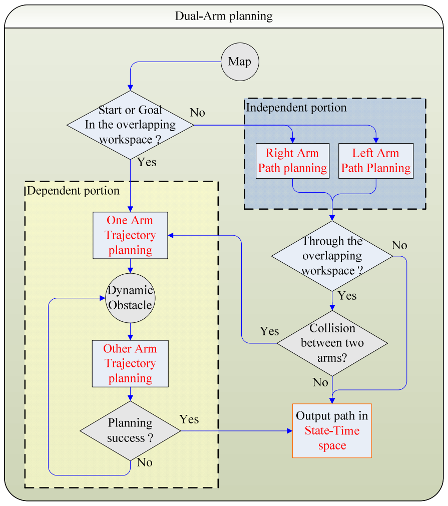

Dual-Arm planning
Map
Start or Goal In the overlapping workspace ?
No
Yes
Independent portion
Right Arm Path planning
Left Arm Path Planning
Dependent portion
One Arm Trajectory planning
Dynamic Obstacle
Other Arm Trajectory planning
Planning success ?
Yes
No
Through the overlapping workspace ?
No
Yes
Collision between two arms?
Yes
No
Output path in State-Time space

Fig. 4 The framework of motion planning for dual arm

B. *"Dual-Arm planning" Scenario*

In this scenario, we mainly test the dual-arm planner in the overlapping workspace. We place six cups whose positions are in the overlapping workspace on the table, and command the two arms to grasp. During the moving process, two arms will interfere with each other.

Fig. 6 shows the simulation for dual-arm planner. Since the CT-RRTs have considered the trajectory of another arm along the time axis, the planner can easily plan a better path. Moreover, because the planner will not stack up the states, it can rapidly check the collision. The robot can rapidly finish many tasks and two arms can work at the same time.

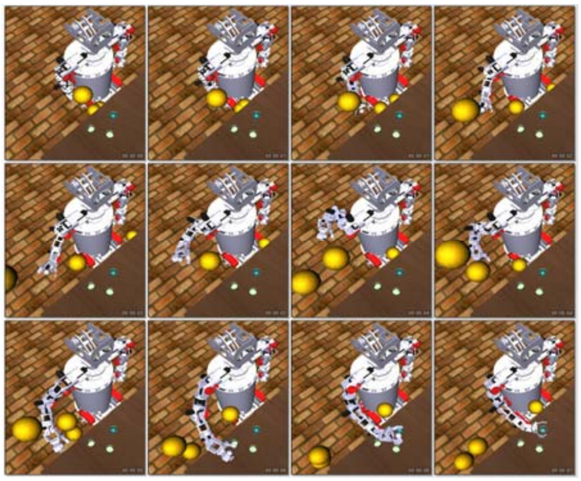


Fig. 5 The simulation for moving objects scenario

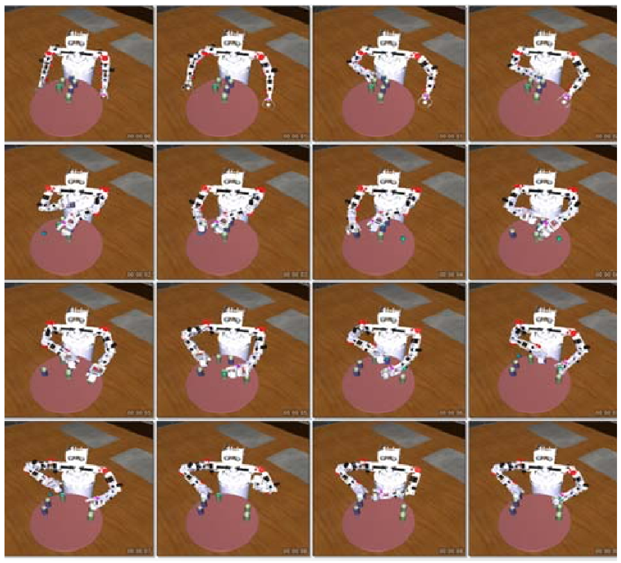


Fig. 6 Dual-arm of scheduling scenario for CT-RRTs

VI. CONCLUSIONS

In this paper, the proposed CT-RRTs is to mainly solve the dynamic motion planning problems. It plans in the augmented state-time space of the robot arm configuration and positions of the moving objects. Various techniques are used to accelerate planning and enhance its safety. When deliberation time is limited, a partial plan can also be returned in an anytime fashion. If the environment is too complex to find a complete solution at first time, the planner can simultaneously plan a path while the robot arm is moving. Due to the advantage of anytime planning, we can save a lot of computation time in the motion process.

The proposed CT-RRTs were used to construct the dual-arm planner, and it can be used as the basis to solve higher level problems in motion planning. From the experiments, the proposed CT-RRTs are very efficient for planning a path in a clutter and dynamic environment.

References

[1] J. V. D. Berg, “Path Planning in Dynamic Environments,” Information and Computing Sciences, Universiteit Utrecht, 2007.

[2] J. Bruce and M. Veloso, “Real-time randomized path planning for robot navigation,” *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 2383-2388 vol.3, 2002.

[3] J. Bruce and M. Veloso, “Real-time randomized path planning for robot navigation,” *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 2383-2388, 2002.

[4] R. Buckingham, “Multi-arm robots,” *Industrial Robots,* vol. 23, pp. 16-20, 1996.

[5] Z. Doulgeri and J. Peltekis, “Modeling and dual arm manipulation of a flexible object,” *2004 IEEE International Conference on Robotics and Automation* (ICRA 2004), vol. 2, pp. 1700-1705, 2004.

[6] Y. Fei, D. Fuqiang, and Z. Xifang, “Collision-free motion planning of dual-arm reconfigurable robots,” *Robotics and Computer-Integrated Manufacturing,* vol. 20, pp. 351-357, 2004.

[7] D. Ferguson, N. Kalra, and A. Stentz, “Replanning with RRTs,” *IEEE International Conference on Robotics and Automation*, pp. 1243-1248, 2006.

[8] R. Gayle, K. R. Klingler, and P. G. Xavier, “Lazy Reconfiguration Forest (LRF) - An Approach for Motion Planning with Multiple Tasks in Dynamic Environments,” *IEEE International Conference on Robotics and Automation*, pp. 1316-1323, 2007.

[9] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, “Randomized kinodynamic motion planning with moving obstacles,” *International Journal of Robotics Research,* vol. 21, pp. 233-255, 2002.

[10] M. Kallman and M. Mataric, “Motion planning using dynamic roadmaps,” *IEEE ICRA*, vol. 5, pp. 4399-4404, 2004.

[11] J. J. Kuffner, Jr. and S. M. LaValle, “RRT-connect: An efficient approach to single-query path planning,” *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 995-1001, 2000.

[12] A. Lambert and D. Gruyer, “Safe path planning in an uncertain-configuration space,” *IEEE International Conference on Robotics and Automation*, vol. 3, pp. 4185-4190, 2003.

[13] S. M. LaValle, “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” *Technical Report*, Computer Science Dept, Iowa State University, 1998.

[14] S. Lee, H. Moradi, and Y. Chunsik, “A real-time dual-arm collision avoidance algorithm for assembly,” *IEEE International Symposium on Assembly and Task Planning (ISATP)*, pp. 7-12, 1997.

[15] T.-Y. Li and Y.-C. Shie, “An incremental learning approach to motion planning with roadmap management,” *Journal of Information Science and Engineering,* vol. 23, pp. 525-538, 2007.

[16] R. C. Smith and P. Cheeseman, “On the representative and estimation of space uncertainty,” *International Journal of Robotics Research,* vol. 5, pp. 56-68, 1986.

[17] L. Tsai-Yen and S. Yang-Chuan, “An incremental learning approach to motion planning with roadmap management,” *IEEE Intl. Conf. on Robotics and Automation*, vol. 4, pp. 3411-3416, 2002.

[18] J. Van Den Berg, D. Ferguson, and J. Kuffner, “Anytime path planning and replanning in dynamic environments,” *IEEE International Conference on Robotics and Automation*, Orlando, FL, United States, vol. 2006, pp. 2366-2371, 2006.

[19] H. C. Yen, “Path Planning for Mobile Robots in Dynamic Environments,” *Master Thesis*, Graduate Institute of Mechanical Engineering, National Taiwan University, 2007.

[20] M. Zucker, J. Kuffner, and M. Branicky, “Multipartite RRTs for Rapid Replanning in Dynamic Environments,” *2007 IEEE International Conference on Robotics and Automation*, pp. 1603-1609, 2007.

[21] M. Saha, P. Isto, and J.-C. Latombe, “Motion planning for robotic manipulation of deformable linear objects,” *Springer Tracts in Advanced Robotics,* vol. 39, pp. 23-32, 2008.

[22] J.Y. Kuan, H.P. Huang, " Independent Joint Dynamic Sliding Mode Control of a Humanoid Robot Arm ," IEEE Intl. Conf. on Robotics and Biomemetics (ROBIO), pp. 202-208, Hainan, China, 2008.