2011 IEEE International Conference on Robotics and Automation
Shanghai International Conference Center
May 9-13, 2011, Shanghai, China

Reactive Robot Motion using Path Replanning and Deformation

Eiichi Yoshida and Fumio Kanehiro.

Abstract— **We present a reactive method for online robot motion replanning in dynamically changing environments by combining path replanning and deformation. Path deformation is newly integrated in our replanning method featured by efficient roadmap reuse and parallel planning and execution. This enhancement allows the planner to deal with more dynamic environments including continuously moving obstacles, by smoothly deforming the path during execution. Simulation results are shown to validate the effectiveness of the proposed method.**

I. INTRODUCTION

Thanks to recent progress, probabilistic sampling-based motion planning methods have gained strong attention in many application areas. These methods are designed to build a graph called a roadmap composed of nodes and edges that represent collision-free configurations and local paths, which expresses the network of free local paths in the given environment. Based on this roadmap, a collision-free path that connects the initial and goal configurations is computed using graph search. Two roadmap building mechanisms are identified as mainstream of sampling-based method: diffusion (e.g. Rapidly-exploring random tree, RRT) and sampling (e.g. Probabilistic RoadMap, PRM) [1], [2].

Those sampling-based planners were originally developed as off-line planning methods in static environments. In accordance with the increasing computational power, now there have been many research efforts on reactive motion planning in dynamic environments where the positions of obstacle changes over time.

The improvement of reactiveness of sampling-based motion planner has been addressed over the last decade. The first approach is the "replanning" of the path when the environment changes. In [3], an obstacle-free roadmap encoded to cell-decomposed workspace is modified according to the environmental changes. Ferguson et al. proposed a replanning method based on RRT for dynamic environments by updating the tree when certain parts become invalid due to the changes [4] and an "anytime" RRT that provides a solution with guaranteed improvement to queries at an arbitrary moment [5]. Online replanning based on RRT has also been applied to urban vehicle driving by improving the efficiency by introducing biased sampling, special heuristics for distance computation and lazy check [6]. Jaillet and Simeon proposed a method for computing several alternative paths to be switched when one path becomes impossible due to environmental changes like door-closing or moving obstacles [7]. Roadmap update based on voxel information of the environment acquired based on vision system is presented in [8]. We have also proposed a practical online replanning method based on roadmap reuse and parallel planning and execution [9]. Those "replanning" methods update the global path and switch between alternative paths by running search in the updated roadmap. Typically, these methods can cope with sudden emergence or discrete or slow displacement of obstacles. Even though the planning efficiency has been improved, the robot should stop frequently if the planning does not finish before the obstacles enters the safety zone. For this reason, it is generally difficult for those "replanning" methods to handle continuously moving obstacles due to repeated queries that are computationally expensive.

In contrast, "deformation" (or "reshaping") method aims at real-time path adaptation by modifying it locally without further queries on the roadmaps. For vehicle navigation, Fraichard et al. proposed a reactive navigation method for car-like vehicles [10] and navigation with guaranteed convergence in dynamic environments [11], [12] have also been reported. The "elastic band" [13], [14] is also categorized in this type that allows the planned path to be adapted to moving obstacles through energy minimization. We have also introduced "path reshaping" for dynamic robot motion [15] but it remains off-line planning. Since drastic changes of the path topology in the roadmap are not assumed while deforming, these are suitable for navigation path adaptation of a vehicle for instance. If the path being executed gets infeasible due to the obstacles come into its way, the "reshaping" method cannot be applied any more.

In this research, therefore, we propose a reactive motion method that combines the replanning and deformation methods. Once a collision-free path is planned and starts being executed, the planner attempts to apply the deformation first when obstacles approach to the robot. The robot can keep executing the path as long as the path remains feasible with necessary deformation according to the motion of obstacles. If the executed path becomes infeasible even after deformation, the replanning is activated to find an alternative path through queries on the updated roadmap. The contribution of the paper is efficient unification of the replanning and deformation by benefiting from the advantages of both the reactivity of deformation and the replanning capacity of globally feasible paths. The computation of path deformation is fast enough because it is done without using inverse of Jacobian. In addition, since the roadmap-based planning is not called unless it is necessary, the roadmap remains compact to

Eiichi Yoshida and Fumio Kanehiro are with CNRS-AIST JRL (Joint Robotics Laboratory), UMI3218/CRT, and F. Kanehiro is also with Humanoid Research Group, both belonging to Intelligent Systems Research Institute, National Institute of Advanced Industrial Science and Technology (AIST), 1-1-1 Umezono, Tsukuba, Ibaraki 305-8568 Japan (e.yoshida@aist.go.jp)

maintain the efficiency of the planning framework in terms of computational time. Reactive motion planning taking into account global connectivity has been proposed as "elastic roadmap" where the global roadmap whose nodes are moved according to environmental changes in order to derive a path that achieves task-level goal [16]. Vannoy and Xiao proposed a framework called "Real-Time Adaptive Motion Planning (RAMP)" that preserves population of feasible paths that are ranked by fitness and switched according to the situation [17]. Since longer control cycle time is set than planning cycle here, drastic path switches may occur. In our research, the path deformation is applied locally to the executed path, and the global connectivity is efficiently managed by learning and working roadmaps for the robot to replan alternative collision-free paths quickly when necessary.

The proposed method is developed based on the online replanning method we have proposed [9] to take advantage of the efficient roadmap reuse and parallel planning and execution. In this paper, after introducing the planning framework in Section II, the deformation method is presented in Section III. Several simulation results are presented in Section IV before concluding the paper.

II. Replanning framework

In this section we present briefly our reactive planning framework composed of parallel execution and planning [9] and also describe how the deformation is integrated in the framework.

A. Planning framework with parallel planning and execution

Figure 1 illustrates how the reactive planner works with different states. It has the feature of having two "threads", Execution and Planning, running simultaneously. In Fig. 1, the texts in box correspond to the "states" of the planner. State transition occurs when a "signal" is received by the thread solid arrows in Fig. 1) or when the internal status of the thread changes (dotted arrows in Fig. 1). The signals and internal status changes causing state transition are indicated by italic and underlined texts in Fig. 1. There are two types of signal: the one is the signal exchanged between the threads, and the other is that received from the outside. The former includes those signals such as "Query", "Finished", "Canceled", "Path found". The signals of the latter type from outside are "Start", "Stop" and "Geo. change" (Detection of environment changes). We assume that the environmental changes can be detected by sensing mechanisms in Fig. 2 in an appropriate manner, to send "Geo. change" signals to the threads as soon as those changes are observed.

The Execution thread is at "Execute" state during path execution, and if "Geo. Change" signal is received, it goes to "Update Problem" without stopping and verifies if replanning is necessary. If no collisions are anticipated, it goes back immediately to "Execute" state and keep executing the path. To avoid repetitive activation of replanning each time changes occur that do not affect the path execution, "Geo. Change" signal is ignored as long as the executed path is collision-free. If collisions are expected, the Execution thread checks the

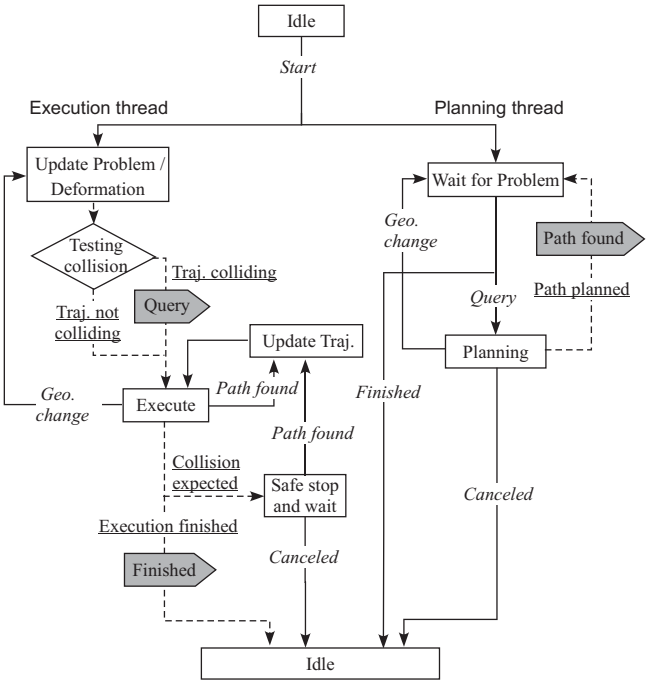


Fig. 1. State transition diagram of replanning. The replanning is performed by parallel threads of Execution and Planning that exchange signals. The states are shown in the boxes and the signal emissions are indicated by shadowed boxes. The italic and underlined texts depict the signals and internal status changes that bring about state transitions respectively.

remaining portion of the path executed to compute a safely stopping path that decelerates and stops at the safety distance away from the obstacle. This trajectory is continuous path execution followed by a safely stopping trajectory, which can be regarded as a contingency plan [18]. Then Execution thread sends "Query" signal to the Planning thread to start the replanning.

If the replanning is successful, the Execution thread goes to the state "Update Traj." to update and execute the replanned path without stopping. If the replanning does not finish before the robot reaches the end of the safely stopping path, the robot stops along this path by decelerating and wait for the planner to return another collision-free path. The planning fails if a feasible path is not found within the specified time. Note that this framework allows Execution thread to go back to the originally planned path by canceling replanning if the obstacles with which collisions are anticipated are removed meanwhile.

Planning thread starts replanning by receiving the "Query" signal from the execution thread at the idle state "Wait for Problem". The planning ends when the planning thread receives "Canceled" or "Geo. change" signal, or when the replanning succeeds.

The replanning is performed based on an incrementally updated roadmap to benefit from the knowledge acquired during the previous exploration of the environment. Two kinds of roadmap, working and learning roadmaps are utilized for this replanning. The learning roadmap stores the information about the environment over the whole planning

time, whereas the working roadmap is continuously updated so that it includes only the valid part involved in the current replanning problem [9]. As a result, the working roadmap remains compact but reflects the most recent changes in the environment. With this replanning framework, the appropriate alternative path can be replanned in case of sudden emergence or discrete displacement of obstacles.

B. Integrating path deformation

Path deformation can be easily integrated in this scheme. Since it does not imply replanning, it is included in Execution thread. When "Geo. change" signal is received with a environmental change, if the robot is executing a path, Execution thread goes to "Update Problem" state and checks whether the path can be improved by applying deformation (Fig. 1). If deformation succeeds without inducing collisions, the path being executed is replaced by the deformed one and goes to "Execution" state to continue the execution as if it was executing the same path. As stated later in III-C, the improvement of the path is evaluated by reduction rate of the path length within a fixed computation timeout. If deformation fails with persistent collisions, replanning is launched by using the updated working roadmap.

C. Interface with motion controller

Since the proposed planning scheme is designed to be used for motion planning and execution online, it is important to think about the interface with robot controllers. We define a general interface in Table I so that the proposed replanning method can be generally applied to various robots. Their corresponding functionalities in the robotic system are illustrated in Fig. 2. The interface includes fundamental functions of robot controller, like starting and stopping path execution, getting the current execution state and configuration. Traveled distance computation based on the sensor information such as localization and odometry is used to estimate the anticipated colliding point. The interface for the detection of environmental changes is also defined for internal or external perception systems.

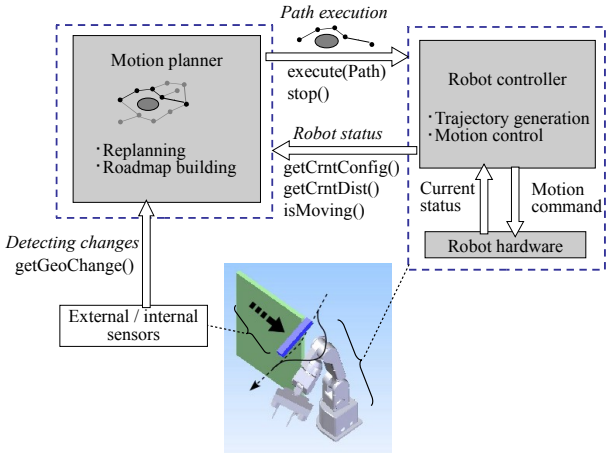


Fig. 2. The functionality of the interface defined in Table I in the robotic systems including motion controller.

TABLE I

THE INTERFACE OF REPLANNING METHOD WITH MOTION CONTROLLER

`bool execute(Path)`	Execute the path
`void stop()`	Stop the execution safely with deceleration
`bool isMoving()`	Return true if moving
`config getCrntConfig()`	Get the current config.
`double getCrntDist()`	Get the traveled distance on the path
`bool getGeoChange()`	Return true in case of environmental changes

III. Replanning with Path Deformation

A. Continuous path deformation

In our previous work [9], a new path search is always performed when replanning is necessary. However, if the interference of the obstacle in the robot motion is not significant, repetitive update of the roadmap and new graph search leads to inefficiency in terms of computational cost.

Especially when the obstacles are moving continuously, in the worst case the replanning process is called every time environmental changes are detected. The robot may stop too frequently if the replanning process does not finish before the obstacles approach within the specified safety distance.

To solve this problem and improve the planning efficiency and robustness in those cases, we introduce a scheme that reshapes continuously the path by moving its arbitrary waypoint instead of repeating roadmap-based path replanning. For this purpose, we adopt the mechanism of path deformation, whose representative method is known as "Elastic Band" [13], [14]. As explained in the following, deformation is performed after estimating deformation direction with respect to the obstacle.

B. Computing deformation direction

As shown in Fig. 3, the distance d_i is defined as repulsion distance. The deformation starts when the minimum distance between the robot and the obstacle becomes smaller than d_i. Let P and Q denote the closest points on the robot and the obstacle. The vector $\overrightarrow{QP}$ can be written as $\overrightarrow{QP} = d\boldsymbol{n}$ by using the distance d between P and Q and the unit vector $\boldsymbol{n}$. In order to "push out" the point P towards the direction that makes d greater than d_i, the displacement ΔP should satisfy the following:

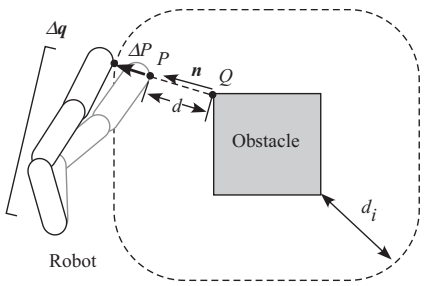


Fig. 3. Direction of path deformation through small displacement for obstacle avoidance

$$\Delta P \cdot \boldsymbol{n} \geq d_i - d.$$

The relationship between the small motion in configuration space $\Delta \boldsymbol{q}$ and ΔP described by using the Jacobian matrix $\boldsymbol{J}_P$ at P as

$$\Delta P = \boldsymbol{J}_P \Delta \boldsymbol{q}.$$

Therefore, the following holds:

Algorighm 1. Path deformation procedure

Deform(*path*, timeLimit, improveThresh)

1: *timeout* ← false, *changed* ← false
2: **while** !*timeout* AND !*changed* **do**
3: *lenOld* ← *path*.length()
4: **for** *i*=1 to *path*.numConfigs() **do**
5: LP_1 ← localPath($\boldsymbol{q}_{i\text{-}1}$, $\boldsymbol{q}_i$)
6: LP_2 ← localPath($\boldsymbol{q}_i$, $\boldsymbol{q}_{i+1}$)
7: $r \leftarrow \dfrac{LP_1\text{.length()}}{LP_1\text{.length()} + LP_2\text{.length()}}$
8: LP← localPath($\boldsymbol{q}_{i\text{-}1}$, $\boldsymbol{q}_{i+1}$)
9: $\boldsymbol{q}_0$ ← LP.getConfigAt(r)
10: $\boldsymbol{q}_{\text{new}}$ ← tighten($\boldsymbol{q}_i$, $\boldsymbol{q}_0$)
11: // local path is "tightened" towards $\boldsymbol{q}_0$
12: *newSeg* ← ∅
13: LP_1^{new} ← localPath($\boldsymbol{q}_{i\text{-}1}$, $\boldsymbol{q}_{\text{new}}$)
14: LP_2^{new} ← localPath($\boldsymbol{q}_{\text{new}}$, $\boldsymbol{q}_{i\text{-}1}$)
15: **if** LP_1^{new}.valid() AND LP_2^{new}.valid() **then**
16: *newSeg* ← makePath($\boldsymbol{q}_{i\text{-}1}$, $\boldsymbol{q}_{\text{new}}$, $\boldsymbol{q}_{i+1}$)
17: *path*.replace(*i*-1, *i*+1, newSeg)
18: *changed* ← true
19: **else**
20: **if** LP_1.valid() **then**
21: *newSeg*.add(LP_1)
22: **else**
23: // cut the local path into two
24: $\boldsymbol{q}_{\text{half}}$ ← LP_1.getConfigAt(0.5)
25: *newSeg*.newPath($\boldsymbol{q}_{i\text{-}1}$, $\boldsymbol{q}_{\text{half}}$, $\boldsymbol{q}_i$)
26: **end if**
27: // do the same thing for LP_2 and update *newSeg*
28: //...
29: *path*.replace(*i*-1, *i*+1, newSeg)
30: *changed* ← true
31: **end if**
32: *timeout* ← checkOutOfTime(timeLimit)
33: **end for**
34: **if** *changed* = true **then**
35: **if** $\dfrac{lenOld - path\text{.length()}}{path\text{.length()}} <$ improveThresh **then**
36: *changed* ← false
37: **end if**
38: **end if**
39: **end while**

$$\boldsymbol{J}_P \Delta \boldsymbol{q} \cdot \boldsymbol{n} = \Delta \boldsymbol{q} \cdot \boldsymbol{J}_P^T \boldsymbol{n} \geq d_i - d$$

By using the lower bound of robot displacement $d_i - d$ the minimum-norm value of $\Delta \boldsymbol{q}$ can be derived without inversing $\boldsymbol{J}_P$ as follows:

$$\Delta \boldsymbol{q} = (d_i - d) \frac{\boldsymbol{J}_P^T \boldsymbol{n}}{||\boldsymbol{J}_P^T \boldsymbol{n}||^2} \tag{1}$$

If the robot makes a movement larger than $||\Delta \boldsymbol{q}||$ in the direction of $\Delta \boldsymbol{q}$, it goes safely out of the bound area defined by d_i.

C. Deformation algorithm

Algorithm 1 describes the deformation algorithm. The executed path is composed of collision-free local paths connecting configurations by the specified steering method. The procedure Deform(*path*, timeLimit, improveThresh) scans the configurations in the path and applies the deformation LP_1 and LP_2, which are before and after the *i*-th configuration $\boldsymbol{q}_i$. This is done by the function tighten() that moves $\boldsymbol{q}_i$ toward the interpolated point $\boldsymbol{q}_0$ on the local path directly connecting $\boldsymbol{q}_{i\text{-}1}$ and $\boldsymbol{q}_{i+1}$. If no obstacles are near the robot, this operation results in simple shortcut and adds a new interpolated configuration $\boldsymbol{q}_{\text{new}}$ in the path.

On the other hand, if the closest obstacle is found within the repulsion area, the function tighten() returns the configuration $\boldsymbol{q}_{\text{new}}$ projected onto the hyperplane of repulsion boundary with d_i in the direction $\boldsymbol{n}$ in Fig. 3 in configuration space, as shown in Fig. 4. If the new local paths LP_1^{new} connecting $\boldsymbol{q}_{i\text{-}1}$ and $\boldsymbol{q}_{\text{new}}$ and LP_2^{new} connecting $\boldsymbol{q}_{\text{new}}$ and $\boldsymbol{q}_{i+1}$ are both valid without collision, then the function Deform() replaces the local paths LP_1 and LP_2 by LP_1^{new} and LP_2^{new}. Otherwise, at least one of the new local paths is not valid.

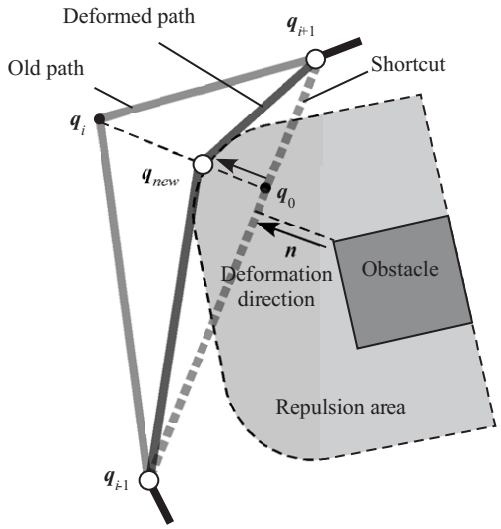


Fig. 4. Local deformation of the path by the function Deform(). The local paths connecting the configurations $\boldsymbol{q}_{i\text{-}1}$, $\boldsymbol{q}_i$, $\boldsymbol{q}_{i+1}$ is deformed using $\boldsymbol{q}_{\text{new}}$ that is the projection of the shortcut configuration $\boldsymbol{q}_0$ onto the hyperplane representing the boundary of repulsion area.

In this case, if LP_n^{new} (n=1, 2) is not valid, the old local path LP_n is divided into two at the middlepoint so that this local path is deformed in the next visit of **for** loop.

The function Deform(*path*, timeLimit, improveThresh) repeats this procedure as long as the timeout and improvement conditions, respectively specified by the parameters "timeLimit" and "improveThresh", are satisfied. The improvement condition is evaluated by the length of the deformed path. The deformation finishes if the deformation procedure in Algorithm 1 does not improve the path length any more than the threshold "improveThresh".

As stated earlier, the deformation procedure is inserted in Execution thread in Fig. 1. When environmental changes are detected during path execution, the thread goes to "Update Problem" state and the deformation is applied to the currently executed path. If the deformation is successful, including the obstacle-free case without any path improvement, then the executed path is replaced by the deformed one. Otherwise, the replanning process is started.

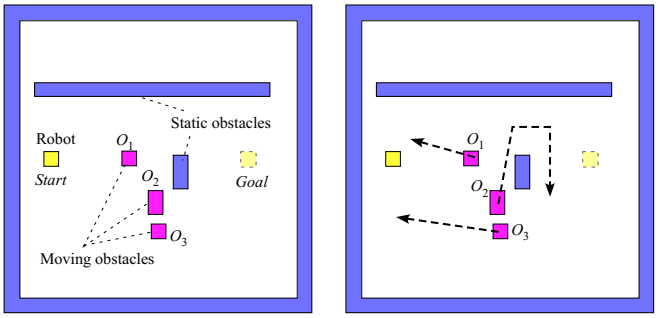
In this way, the deformation is naturally integrated in the framework of parallel planning and execution.

IV. Planning Examples

In this section we show some results of the proposed planning method. As a case that our previous method [9] has difficulty in handling, we address planning with continuously moving obstacles.

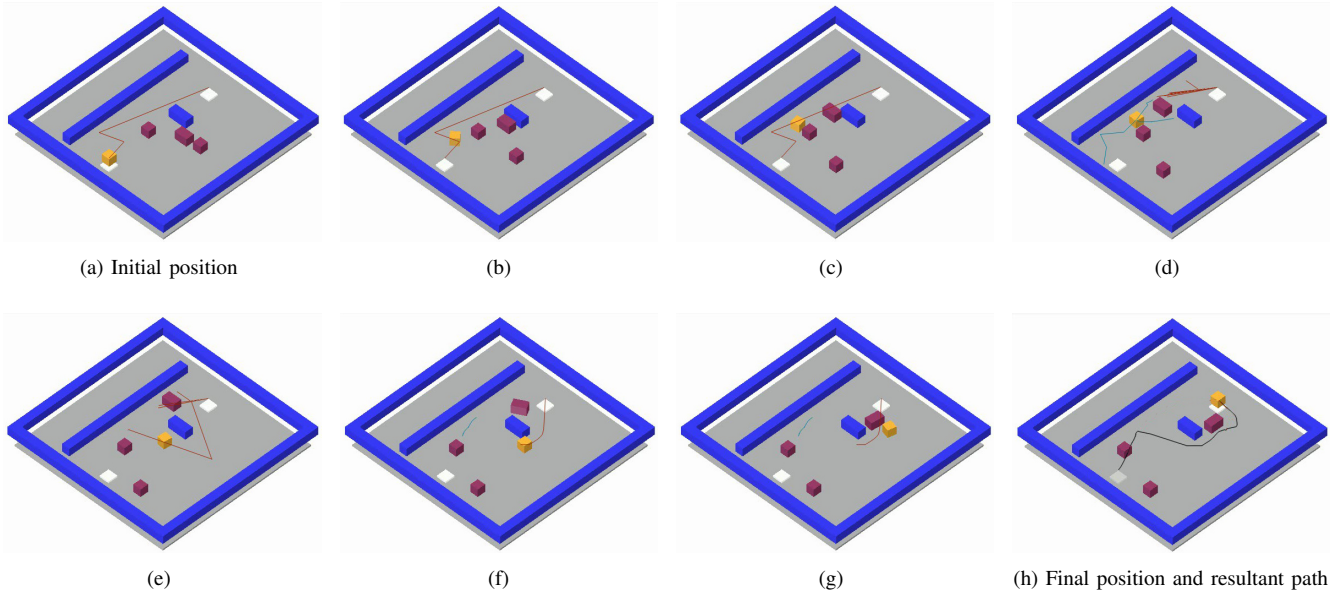
Figure 5 shows the environment where the robot moves linearly on a plane with static and moving obstacles. The robot is required to arrive the goal position avoiding the three moving obstacles $O_1 \sim O_3$ making continuous displacement with different velocity. A simple robot controller is implemented according to the general interface defined in Table I. This controller follows the given command as much as possible with a specified velocity limit. The motion of the obstacles is set with velocities below this limit.

A planning result is shown in Fig. 6. Since the goal cannot be directly reached from the start position, a collision-free path is first planned on the left side of the environment and the robot starts executing it. We can observe that the path is kept deformed as O_1 moves continuously towards the robot. As long as the originally planned path can be deformed to avoid obstacle, roadmap-based replanning is not performed (Fig. 6b-c). Then another obstacle O_3 gets in the robot's way to fill the passage the robot is supposed to pass through. This situation cannot be resolved with deformation and a path passing on the other side of the obstacle O_2 should be replanned using the work roadmap continuously updated during the execution (Fig. 6d). As can be seen



(a) Simulation environment (b) Movement of obstacles

Fig. 5. Simulation environment with continuously moving obstacles



(a) Initial position (b) (c) (d)

(e) (f) (g) (h) Final position and resultant path

Fig. 6. Results of reactive motion with deformation and replanning. The initial path is deformed at (b-d). Alternative path is searched when obstacle fills the gap (d) and the replanned path is executed (e-g) which is also deformed at (g). The resultant path is shown in (h). The lines in the figures represent connective components of roadmaps used for the (re)planning. As the path is smoothed later and is also deformed, it does not necessarily correspond to the consequent robot motions.

in Fig. 6e-f, the robot changes the moving direction as soon as the alternative path is replanned. This obstacle O_3 keeps the motion by rotating and encounters the robot again near the goal position. Here also, the path is deformed to avoid the collision (Fig. 6g) before arriving at the goal. The improvement threshold and timeout are set to 1% and 0.1s respectively in this planning simulation. The deformation can be observed more clearly in the accompanied movie.

We have also tested the planner without deformation in the same environment and obstacle motions. As a result of ten simulation runs with different random number seeds, as much as 70% of execution failed by ending up with collision, whereas the success ratio was 100% with the proposed method with deformation. Since the roadmap-based replanning is activated when the obstacle motions are detected at every sampling time, the replanning sometimes does not finish even though the replanning is efficient enough with roadmap reuse. In this case the robot stops at the safety distance while the obstacles are still moving. This often results in collisions when the obstacles pass near the robot.

In addition to execution success ratio, the efficiency of the search is also improved. Since many cases of continuous obstacle movements can be handled by deformation without replanning, we can expect that reduced replanning calls help the planner keep the roadmaps compact during the planning and execution. Figure 7 shows the size of learning roadmap in successful cases of planning with and without deformation. We can observe that the size of roadmap becomes approximately four times smaller by using deformation, which leads to efficient planning with roadmaps.

Figure 8 shows the comparison of cumulative planning time. As expected, the time consumed for replanning is significantly reduced if deformation is applied. On the other hand, cumulative computation time for deformation is comparable with the replanning time of "replanning only" case. We can remark that deformation time is increasing during replanning is not performed from around 20s to 60s. This is because the deformation is integrated in Execution thread and called every time obstacle motions are detected. In the proposed planning framework in Fig 1, even if the moving obstacles are not currently close to the robot, the deformation procedure is called in a preventive manner to deal with future collision in the remaining portion of the executed path. Considering the great improvement of planning success ratio with continuously moving obstacle, this extra computational cost is reasonable since there is no significant loss of performance.

Another planning result with a redundant manipulator is shown in Fig. 9. The problem is to generate collision-free motion of a 7-DOF manipulator arm from the initial (Fig. 9a) to goal (Fig. 9h) configuration in an environment populated with static and moving obstacles. First, when a cubic obstacle approaches to the manipulator going out of the gap between two further static plate obstacles, the path is deformed towards the depth direction (Fig. 9b, c). Then the manipulator keep moving towards the goal configuration at a lower position between the two closer

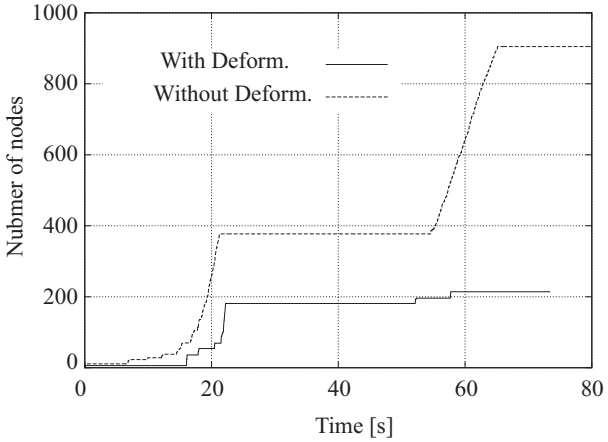


Fig. 7. Development of learning roadmap size during execution with and without deformation

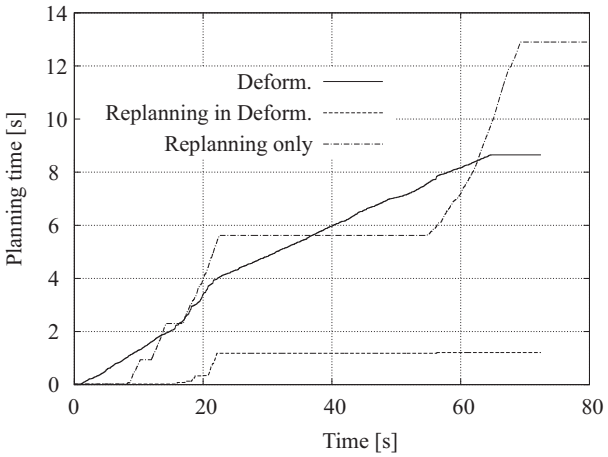


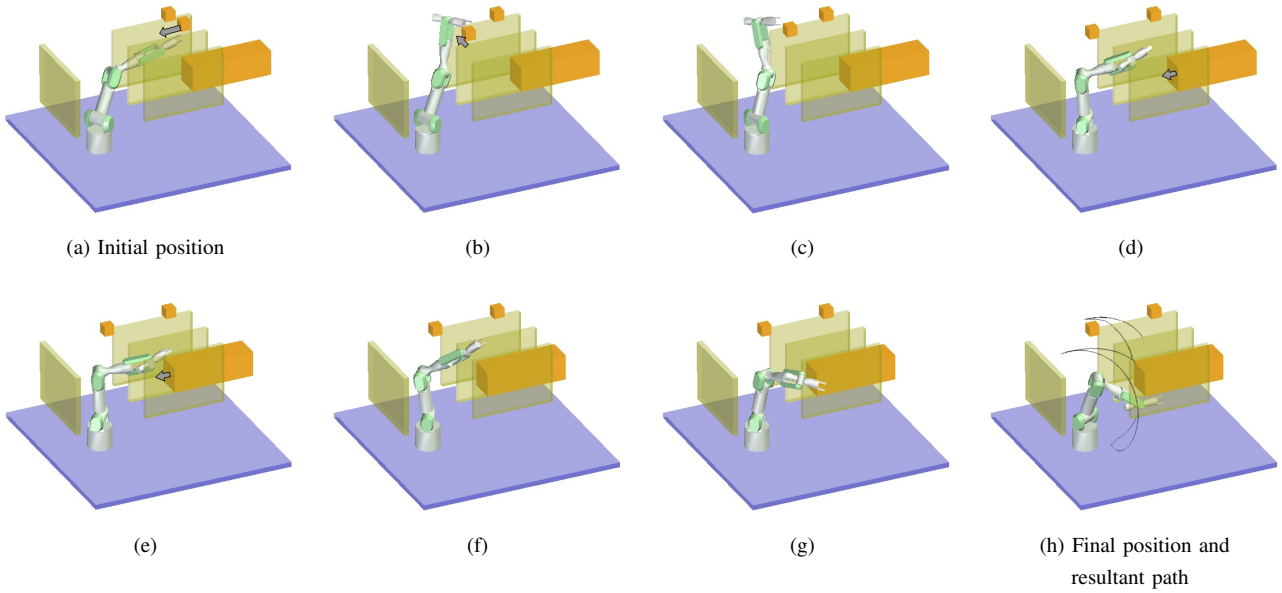
Fig. 8. Cumulative planning time with and without deformation. "Deform." and "Replanning in Deform." mean computation time for deformation and replanning with the proposed planner. "Replanning only" means the planner without deformation.

plate obstacles (Fig. 9d). At this point, another box-shape obstacle approaches and collision becomes inevitable even after applying path deformation (Fig. 9e). An alternative path passing outside of the plate outermost obstacle is replanned based on the roadmap to achieve the goal (Fig. 9f-h).

In the simulation results, reactive path has been performed in an adaptive manner by switching path replanning and deformation according to the changes in the environment, which demonstrates the effectiveness of the proposed method.

V. Conclusions

In this paper we have presented a method for reactive robot motion planning including replanning and deformation. We have introduced path deformation mechanism based on the observed local distance to obstacles. This is integrated on top of our previous replanning method featured by parallel planning and execution. This enhancement allows the planner to cope with the changing environment including continuous



(a) Initial position (b) (c) (d)

(e) (f) (g) (h) Final position and resultant path

Fig. 9. Planning of a redundant manipulator. The deformation is performed with the small moving obstacle (b-c). When the planned path is obstructed (d-e), a new path is replanned (f-h). The resultant end-effector path is shown in (h).

obstacle movements through progressive deformation of the executed path. The advantage of replanning capacity is always maintained; when this deformation cannot remove the anticipated collisions, an alternative path is efficiently computed by the replanning framework based on the continuously refreshed roadmap. We have shown the effectiveness of the proposed method by planning simulation where there are multiple moving obstacles. The planner handled correctly each situation by applying deformation and replanning in a timely manner.

There is still room for improvement in the proposed method. The path deformation procedure can be made even more efficient by pruning unnecessary computation when there is little risk of anticipated collisions, whereas it is currently preventively called every time obstacle motions are detected. Validation of the proposed method with complex robots through dynamic simulations or hardware experiments is also an issue to be addressed in future work.

ACKNOWLEDGMENTS

This work has partially been supported by Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for Scientific Research (B), 21300078, 2009. The authors thank to Etienne Ferré and Ambroise Confetti of Kineo CAM for their precious support for the software developments.

REFERENCES

[1] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, 2006.
[2] S. LaValle, *Planning Algorithm*. Cambridge University Press, 2006.
[3] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *Int. J. of Robotics Research*, vol. 21, no. 12, pp. 999–1030, 2002.
[4] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *Proc. 2006 IEEE Int. Conf. on Robotics and Automation*, 2006, pp. 1243 – 1248.
[5] D. Ferguson and A. Stent, "Anytime RRTs," in *Proc. 2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006, pp. 5369 – 5375.
[6] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, "Motion planning for urban driving using RRT," in *Proc. 2008 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2008, pp. 1681–1686.
[7] L. Jaillet and T. Simeon, "Path deformation roadmaps: Compact graphs with useful cycles for motion planning," *Int. J. of Robotics Research*, vol. 27, no. 11-12, pp. 1175–1188, 2008.
[8] A. Nakhaei and F. Lamiraux, "Motion planning for humanoid robots in environments modeled by vision," in *Proc. 8th IEEE-RAS Int. Conf. on Humanoid Robots*, 2008, pp. 197–204.
[9] E. Yoshida, K. Yokoi, and P. Gergondet, "Online replanning for reactive robot motion: Practical aspects," in *Proc. 2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2010, pp. 5927–5933.
[10] T. Fraichard, M. Hassoun, and C. Laugier, "Reactive motion planning in a dynamic world," in *Proc. of the IEEE Int. Conf. on Advanced Robotics*. IEEE, 1991, pp. 1028–1032.
[11] J. M. andand Luis Montano, "Nearness diagram (nd) navigation: Collision avoidance in troublesome scenarios," *IEEE Trans. on Robotics*, vol. 20, no. 1, pp. 45–59, 2004.
[12] P. gren and N. E. Leonard, "A convergent dynamic window approach to obstacle avoidance," *IEEE Trans. on Robotics*, vol. 21, no. 2, pp. 188–195, 2005.
[13] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *Proc. 1993 IEEE Int. Conf. on Robotics and Automation*, 1993, pp. 802–807.
[14] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *Int. J. of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.
[15] E. Yoshida, C. Esteves, I. Belousov, J.-P. Laumond, T. Sakaguchi, and K. Yokoi, "Planning 3D collision-free dynamic robotic motion through iterative reshaping," *IEEE Trans. on Robotics*, vol. 24, no. 5, pp. 1186–1198, 2008.
[16] Y. Yang and O. Brock, "Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments," in *Proc. Robotics: Science and Systems,*, 2007.
[17] J. Vannoy and J. Xiao, "Real-time adaptive motion planning (RAMP) of mobile manipulators in dynamic environments with unforeseen changes," *IEEE Trans. on Robotics*, vol. 24, pp. 1199–1212, 2008.
[18] K. E. Bekris and L. E. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *Proc. 2007 IEEE Int. Conf. on Robotics and Automation*, 2007, pp. 704–710.