Online Replanning for Reactive Robot Motion: Practical Aspects

Eiichi Yoshida, Kazuhito Yokoi and Pierre Gergondet.

***Abstract*— We address practical issues to develop reactive motion planning method capable of replanning the path online when the environment changes during the execution. By introducing planning and execution threads running in parallel, the robot can keep moving even during the replanning process as long as the motion is safe. The proposed method can be applied to discontinuous input of environmental changes that may occur due to incomplete perception. We then address a roadmap reuse method for efficient replanning to make use of the increasing knowledge about the environment, by introducing working and learning roadmaps. A general interface with robot motion controller is also defined so that the method can be applied to various types of robots. The proposed method is validated through planning simulations with moving obstacles.**

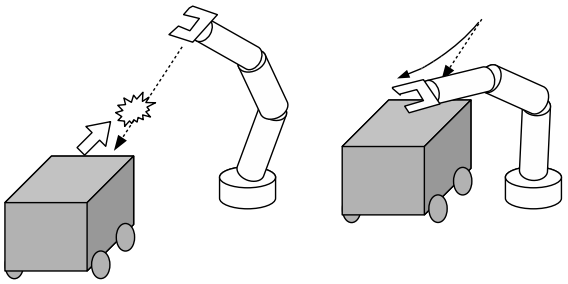


Fig. 1. Reactive path replanning

I. INTRODUCTION

Motion planning techniques for robotic systems have become more and more efficient recent years. Especially, probabilistic sampling-based motion planning methods have had great success [1], [2] . These methods consist in building a graph called roadmap whose nodes are collision-free configurations. There are mainly two roadmap building method, diffusion (e.g. Rapidly-exploring random tree, RRT) and sampling (e.g. Probabilistic RoadMap, PRM) [1], [2]. Based on this roadmap, a collision-free path from initial to goal configurations are sought through graph search. Those methods are now applied to complex robots like humanoids [3], [4].

Initially those motion planning methods assume off-line planning with static environments. There have been quite a few efforts to improve the reactiveness of the method to face dynamically changing environment (Fig. 1).

Fraichard et al. proposed a reactive navigation method for car-like vehicles [5]. Quinlan and Brock et al. proposed "elastic band" so that the planned path can reactively adapt to moving obstacles [6], [7]. Those methods are classified as "reshaping" that deforms the planned path rather than doing further queries on the roadmaps. 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.

On the other hand, there have been increasing research efforts on reactive "replanning" to cope with the cases where the planned path is blocked due to dynamic changes of the environment. Leven and Hutchinson proposed a method for path planning in changing environments [8]. An obstacle-free roadmap is constructed and encoded to workspace decomposed to cells, which is modified according to the environmental changes. Sampling-based planning started being used for online replanning in real-time applications. Ferguson et al. proposed [9] replanning using RRT for dynamic environments by updating the tree when certain parts become invalid due to the changes. They also introduced an "anytime" RRT that rapidly provides initial solution and proposes plans with guaranteed improvement [10]. Kuwata et al. applied RRT to online replanning for urban vehicle driving. They improved planning efficiency required for the real-time application by introducing biased sampling, special heuristics for distance computation and lazy check [11].

As other research efforts to cope with dynamic environments, Jaillet and Simeon proposed a method for computing several alternative paths to be switched when one path becomes impossible without replanning [12]. Van den Berg et al. [13] proposed a method for "anytime" path replanning, where planned paths are continuously modified to avoid collisions with movable obstacles by modeling their future configurations as a cylinder in state-time space. An efficient method for extension is presented to deal with movable obstacle with uncertainty with growing disks [14]. Recently Nakhaei et al. proposed a navigation method for humanoid robot by updating the roadmap in dynamic environment by voxels measured by a vision system [15].

In this research, we address practical issues of a reactive replanning method that can cope with frequent environmental changes including discrete ones like sudden appearance of obstacles. Once a path is planned, it is executed in one thread. Future collisions are checked based on the planned path when environmental changes are detected during the path execution. If any collision is anticipated, then the replanning process is activated in parallel as another thread, while the execution is still in progress. As soon as a collision-

Eiichi Yoshida and Pierre Gergondet are with CNRS-AIST JRL (Joint Robotics Laboratory), UMI3218/CRT and Kazuhito Yokoi is 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)

free path is replanned, the updated path is passed to the execution thread to execute it immediately. By allowing the motion execution during replanning, unnecessary stop can be avoided: if the colliding obstacles are removed during the replanning, the robot can keep moving by discarding the replanning. In the worst case where the replanning does not finish before the robot arrives at the anticipated colliding point, the execution thread makes the robot stop safely and wait for the replanning to finish. The planning scheme is also flexible enough to allow the goal configuration to be changed at any time. Petti and Fraichard [16] proposed partial motion planning for simultaneous planning and execution divided into synchronous processes. On the other hand, in this paper we first obtain the path to the goal and deal with environmental changes by replanning and safe execution in an asynchronous way.

The replanning is performed by using incremental roadmap update to reuse those previously constructed. We introduce two kinds of roadmap, working and learning roadmaps. The learning roadmap stores the information about the environment over the whole planning time, whereas the working roadmap is kept compact by adding only the part involved in the current replanning problem.

The contribution of the paper consists of the practical implementation issues on reactive planning: continuous planning and execution in parallel, replanning based on roadmap reuse, and a general interface with robot motion controller are integrated into a reactive planning system.

In this paper, first the reactive planning framework is described including execution and planning threads that can run in parallel to meet the reactiveness requirements for the desired planner in Section II. Then in Section III we present a method for roadmap reuse for efficient replanning. Valid collision-free edges in the roadmap are stored and updated continuously throughout the planning process. After addressing a general interface to the robot controller and the perception in Section IV, we validate the proposed method by planning simulations in Section V.

II. Structure of replanning system

In this section we present the specifications for the planner we aim to devise for the reactive replanning and introduce a framework including two parallel threads for path execution and planning.

A. Specifications

By assuming that changes have been detected during the robot motion, the following specifications are required so that the planner can replan the path.

- In the event of environmental changes, if collisions are anticipated on the planned path being executed, the planner starts replanning immediately. As soon as another collision-free path is obtained again, it is executed.
- During the path replanning, the robot continues its motion unless it approaches obstacles within the specified safety distance. The path is executed in such a way that it decelerates when approaching the obstacle and makes a complete stop at a safety distance.
- There may be a case where the anticipated collision on the path is removed during the replanning. In this case, the replanning is canceled and the robot continues executing the original path without stopping.

B. Planner structure

Figure 2 illustrates the state transition diagram of a planner structure that satisfies the aforementioned specifications. It has the feature of having two "threads" that are running simultaneously in order to allow the robot to do planning during its execution of the previously planned motion.

In Fig. 2, the texts in box correspond to the "states" of the planner. The states make transition from one to another when "signals" are received by the thread or when some internal states change. The shadowed texts show the signal emission. 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).

The planner should cope with unexpected environmental changes, such as displacement of obstacles after starting the motion or appearance of newly detected obstacles. We assume that those environmental changes can be detected by visual or range sensors equipped to the robot or those embedded in the environment. The location and geometry of the obstacles detected from those sensors will be utilized for replanning. Modification of the goal configuration during path execution can also be handled as an environmental

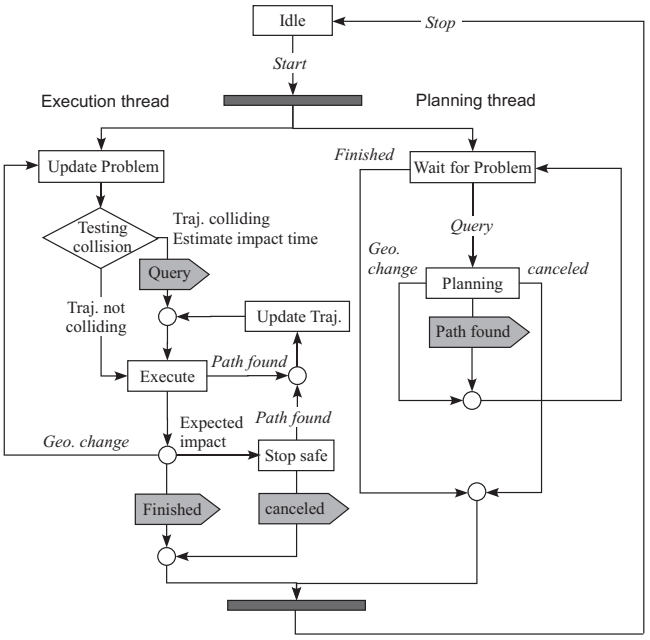


Fig. 2. State transition diagram of replanning. The replanning is performed by parallel threads of Execution and Planning that exchange signals. The states are shown in the boxes and the signal emissions are indicated by shadowed boxes. The italic texts depict the signal reception and corresponding state transition.

change in a broad sense. We can therefore apply this framework to the cases where the goal configuration is frequently updated based on perception.

The initial state is "Idle" before starting by receiving the signal "Start".

C. Execution Thread

At the "Update Problem" state, the Execution thread keeps track of the current status of the robot, the environment and the collision with the obstacle for the planned path if any. If no collisions are detected, the states transit to "Execute" to execute the path. Even if "Geo. Change" is received due to environmental changes, it is ignored and the execution continues as long as the executed path is collision-free. This avoids repetitive activation of replanning each time changes occur that do not affect the path execution.

If collisions are expected, the Execution thread computes the remaining distance along the path until the collision and sends "Query" signal to the Planning thread. Let us denote the path by $P(s)$ which is parameterized between an interval $[0, S]$. When a collision is anticipated along the current path at s_2 and the robot is currently at $P(s_0)(s_0 < s_2)$, the Execution thread chooses a point $s_1 \in [s_0, s_2]$ of starting deceleration such that the robot has sufficient time to stop before s_2. Then it executes a trajectory until s_1 normally and starts the stopping motion if replanning is not finished before arriving at s_1. This trajectory is continuous path execution followed by a safe stopping trajectory, which can be regarded as a contingency plan [17].

If the replanning is successful, the Execution thread goes to the state "Update Traj." to update the path to be executed as described later in Section III. Then the robot does not stop at s_1 to execute the updated collision-free path.

By allowing the robot to execute the path until the anticipated collision, unnecessary suspension of the execution can be avoided. Thanks to this mechanism, if the obstacles with which collisions are anticipated are removed during the motion, the replanning is canceled to go on the originally planned path.

D. Planning thread

Initially, the Planning thread stays at the state "Wait for Problem". Then when it receives the "Query" signal from the execution thread, it starts replanning and goes to "Planning" state. The state "Planning" ends when the planning thread receives "Canceled" or "Geo. change" signal, or when the replanning succeeds, as explained later in Section V. Then the thread goes back to "Wait for problem" state except the first case where the replanning itself is canceled.

III. Replanning by reusing roadmap

A. Continuous roadmap update

For reactive replanning, it is important to manage the roadmap efficiently to reflect the dynamic environmental changes. In previous researches also, the static roadmap are updated online [8], [13], [10], [11].

In our case, we allow the roadmap have several connective components. In other words, the roadmap can include some isolated connective components which will be eventually connected to other connected components as the roadmap building progresses. When the roadmap should be updated due to motion of obstacles, managing connectivity property when deleting the nodes becomes very cumbersome. Therefore, rather than making this difficult operation, a long-term learning roadmap to be reused, denoted Rdm_{learn} here, is maintained throughout the planning process including the roadmap building. This roadmap Rdm_{learn} is created once at the initialization stage before starting whole planning and execution processes.

Each time the replanning is requested, the working roadmap Rdm_{crnt} for replanning is cleared. Before starting the planning, the function PrefillRoadmap() is called to extract the valid collision-free edges as the list *EnrichEdges* from the learning roadmap Rdm_{learn} as shown in Fig. 3. This function runs rapidly because there is no search.

The planning thus starts with this list *EnrichEdges* and the empty working roadmap Rdm_{crnt}. In sampling-based planning methods like PRM and RRT, the roadmap is grown by adding nodes through sampling in the configuration space. The planning proceeds by repeating an incremental procedure that executes a stepwise basic roadmap construction operation, e.g. find one valid edge to add the roadmap as described later in Section V.

We aim to enrich the working roadmap Rdm_{crnt} for the new environment by adding the edges in *EnrichEdges*. However, addition of all the collision-free edges to the roadmap at the beginning is not practical because applying collision checking to all the edges at a time is a relatively costly operation. Thus this roadmap enrichment operation is implemented as a step function EnrichRoadmapStep() that is called at each call of the incremental planning process as described in Fig. 4. This helps the planner have a well-explored working roadmap rapidly by keeping it compact at the same time.

PrefillRoadmap(Rdm_{learn}, *EnrichEdges*)

```
1: for i = 0 to Rdm_learn.numNodes() do
2:    N_start ← Rdm_learn.node(i)
3:    for j = 0 to N_start.numOutGoingEdge() do
4:       E ← N_start.outGoingEdge(j)
5:       N_end ← E.endNode()
6:       if ¬N_start.isDeadEnd() AND ¬N_end.isDeadEnd()
         then
7:          EnrichEdges.add(E)
8:       end if
9:    end for
10: end for
```

Fig. 3. A procedure extracting interested edges from learning roadmap

EnrichRoadmapStep(Rdm_{crnt}, *EnrichEdges*)

```
1: addFlag ← false
2: for i = 0 to EnrichEdges.size() do
3:    E^r ← EnrichEdge[i]
4:    N^r_start ← E^r.startNode(i)
5:    N^r_end ← E^r.endNode(i)
6:    if N^r_start ∈ Rdm_crnt OR N^r_end ∈ Rdm_crnt then
7:       if Rdm_crnt.addEdge(E^r) == success then
8:          addFlag ← true
9:          EnrichEdges.erase(E^r)
10:         // The edge was successfully added
11:         break
12:      end if
13:   end if
14: end for
15: if addFlag == false then
16:   // Add an edge anyway even if isolated
17:   for i = 0 to EnrichEdges.size() do
18:      E^r ← EnrichEdge[i]
19:      if Rdm_crnt.addEdge(E^r) == success then
20:         addFlag ← true
21:         EnrichEdges.erase(E^r)
22:         break
23:      end if
24:   end for
25: end if
```

Fig. 4. Function for one-step enrichment of the roadmap during planning

B. Online replanning using the roadmap

As we mentioned in Section II, the planning thread runs during the motion execution and then the replanned path is executed as soon as a solution is found.

Here we explain how the updated roadmap is utilized for replanning. Figure 5 shows how actually it works. When environmental changes are detected and the "Geo. change" signals is received, the execution thread estimates the impact point s_2 where the collision will occur along the path $P(s)$ being executed and s_1 to start safe stopping motion (Fig. 5a).

Thus the replanning is made to find a collision-free path from the configuration $P(s_2)$ to the goal configuration by using the quickly enriched working roadmap (Fig. 5b). If a collision-free path $P'(s)$ is found before reaching s_1, then the Planning thread sends "Path found" signal to the Execution thread that goes to "Update Traj." state. The Execution thread then adds a short collision-free connecting edge between a configuration before s_1 on $P(s)$ and another on $P'(s)$. To smooth transition from the current path to the replanned one, a path optimization is applied (for example, adaptive shortcut optimization [18]) to the whole replanned path (Fig. 5c). As soon as the optimization finishes, the Execution thread start executing the resulting replanned path (Fig. 5d).

If the replanning does not finish before reaching s_1, the robot makes a stop at s_2 safely and the execution is

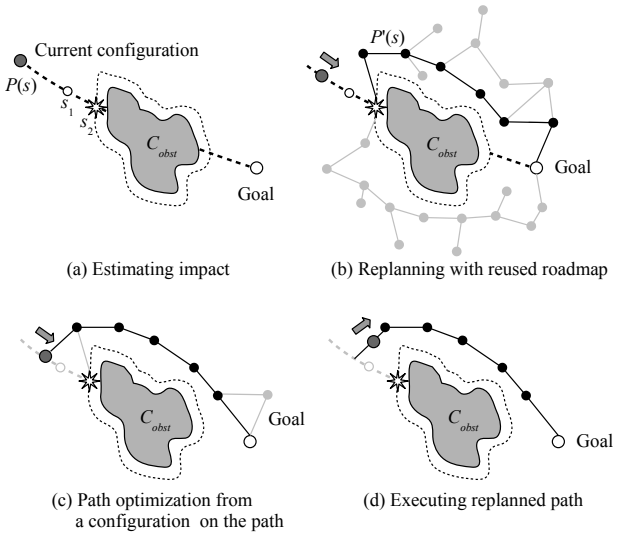


Fig. 5. Replanning the path with the anticipated collision.

suspended until the replanning is completed. If a timeout is defined for replanning, the planning is canceled and waits for another request.

In this way, reactive replanning is performed in an efficient way by reusing the continuously updated roadmap.

C. Limitations

One of the difficult situations for the proposed method occurs when obstacles move continuously. In this case, a motion history can be used for a conservative collision check. Let us denote the position of obstacle i observed at k th sampling time t_k by $p_i^{\text{obs}}(t_k)$ and the observation sampling period by T. By estimating its velocity $v_i^{\text{obs}}(t_k)$ over past N sampling time, we use the geometry of the obstacle grown by $v_i^{\text{obs}}(t_k)T$. If an obstacle moves continuously, the Execution thread keeps receiving the "Geo. change" signal. As the replanning process is not started as long as the trajectory is collision-free, the path execution is not influenced. If the obstacle motion updates the estimated impact point s_2 continuously, the replanning is triggered frequently. Since the replanning process runs in parallel with execution, the path execution is not influenced either as long as the replanning finishes before arriving at s_2.

However, even though the roadmap reuse accelerates the replanning process, it may happen that the replanning does not finish before reaching the newly estimated impact point due to rapid obstacle motion, which means collision avoidance is not guaranteed. In this sense the proposed method is more suitable for the environmental changes that are discrete and gradual with respect to the robot motion and the planning capacity. To tackle such a case of continuously moving obstacles, we will need to combine the proposed method with reactive path deformation like elastic bands [6], [7].

Another limitation concerns complex dynamic constraints of the robot, including time constraints. If we include exact dynamics in planning, it is too computationally expensive. One of the reasonable solutions is two-stage planning like in [4] where replanning can be performed fast enough with

a simpler geometric model using appropriate tolerance. In this case, estimation of deviation due to the dynamic motion from the nominal path is important for collision avoidance.

IV. Interface with Motion Controller

For the proposed replanning method to be generally applicable to various robots, we define a general interface in Table I. Their corresponding functionalities in the robotic system are illustrated in Fig. 6. The functions of starting and stopping the path execution, and of getting the current execution state and the configuration, are required to perform replanning and path modification according to the situation. The function of traveled distance computation 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. The environmental changes detection is the trigger of the replanning process.

V. Implementation and Planning Example

A. Stepwise Planner Implementation

The proposed replanning method presented in Fig. 2 has been implemented and tested to verify its effectiveness. In order to make the planner thread reactive and to run in parallel with the execution, the planner is implemented in an incremental manner so that using the function StepPlan() shown in Fig. 7.

After initializing by calling PrefillRoadmap() in Fig. 3, the planner repeats this stepwise planning that corresponds to an incremental expansion of the working roadmap Rdm_{crnt} in RRT or PRM. At each of this stepwise planning procedure, the roadmap update EnrichRoadmapStep() in Fig. 4 is called to make use of the learning roadmap Rdm_{learning}. This "atomic" implementation is suitable for the proposed replanning method with parallel threads since the planning can be reactively activated and interrupted through exchanges of signals. For example, if the signal "Geo. Change" is received during planning, the planning is canceled because the roadmap used in the current working roadmap does not reflect the newest environment any more. This stepwise planner has been implemented by using motion planning software library KineoWorks$^{\text{TM}}$ [19].

The proposed planner is implemented on RT (Robot Technology) Middleware which has been proposed as software platform [20] as a software unit called an RT Component. RT Middleware encourages the modularization and the reuse of software in the robotic field. Other software modules communicating with the motion planner, like the robot controllers and sensor systems to detect the environmental changes in Fig. 6, can also be implemented as RT Components.

B. Planning results

We here employ an iterative sampling-based motion planning method based on RRT, which is an iterative method based on the automatic tuning of obstacle penetration tolerance [21]. In this method, the thinned obstacles are used

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

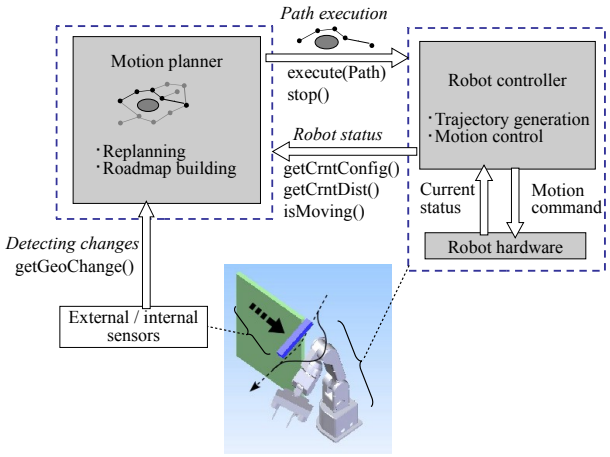


Fig. 6. The functionality of the interface defined in I in the robotic systems including motion controller.

motionPlanner(*startCfg*, *goalCfg*)

```
1:  Rdm_learn ← ∅
2:  while waitingProblem == true do
3:    if received("Query", startCfg, goalCfg) then
4:      // start planning for a problem
5:      initPlan()   // initialization
6:      PrefillRoadmap(Rdm_learn, EnrichEdges)
7:      solved ← false; Rdm_crnt ← ∅
8:      while ¬solved AND ¬received("Geo. Change")
        AND ¬received("Canceled") do
9:        // one step planning
10:       solved ← StepPlan(Rdm_crnt, startCfg, goalCfg,
          solPath)
11:       UpdateRoadmap(Rdm_crnt, Rdm_learn)
12:       EnrichRoadmapStep(Rdm_crnt, EnrichEdges)
13:     end while
14:     endPlan()   // ending operation
15:     if solved == true then
16:       // send the solution path if success
17:       SendToExecThread(solPath)
18:     else if received("Canceled") then
19:       waitingProblem = false
20:     end if
21:   end if
22: end while
```

Fig. 7. Motion planning with incremental planning framework

to make the planning easier. Then the obstacles are grown gradually towards the original size by decreasing the allowed penetration for path deformation to keep the robot can stay away from them. In the course of this iterative method, the working roadmap is sometimes reset when the thinned obstacle is grown.

Figures 8 and 9 show an example for a replanning problem using a robot manipulator arm, with and without environmental changes. In Fig. 8, the arm is required to move from the initial (a) to final (d) configurations by avoiding collisions with fixed (wall) and moving (bar-shape) obstacles.

During the execution of path planned, the position and orientation of the movable bar-shape obstacle are changed from those of Fig. 8, as shown in Fig. 9(a). If the previously planned path is executed, the arm configuration like Fig. 8(c) becomes invalid by the moved object. We assume some sensor components that detect this environmental change and send the "Geo. Change" signal to the execution thread to make the replanning process start. As can be seen, another path avoiding the displaced obstacle is planned as shown in Fig. 9(b),(c) to avoid collisions and to achieve the initially defined goal Fig. 9(d).

In the accompanying movie, the obstacle is manually moved to prevent the robot from executing the path toward the goal configuration. We can observe that the planner is capable of finding alternative paths online.

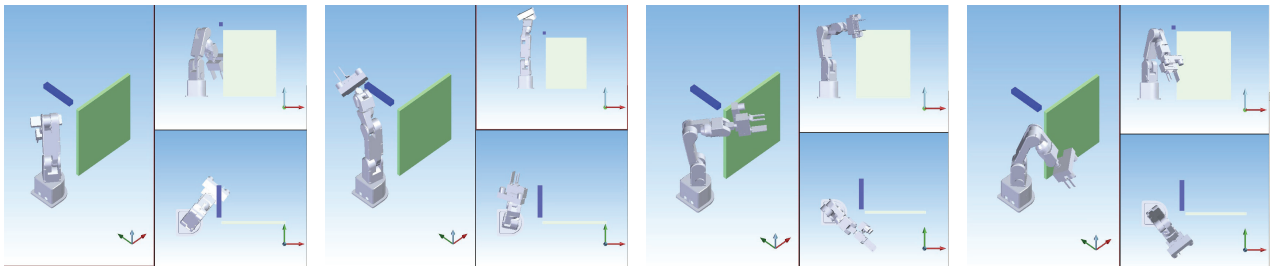
Figures 10 and 11 depict the time evolution of the roadmap size (the number of edges) and the activation of the Execution and Planning threads for abrupt displacement or appearance of the obstacles in the similar environment. In Fig. 10, each time the obstacle is moved discretely, the replanning starts in parallel with the execution that continues until the robot enters in the safety distance. The execution is not suspended in Fig. 11 because the obstacle disappears before the completion of replanning which was canceled right after the disappearance.

We can also observe the increase of the roadmap size of the working and learning roadmaps. The drastic decrease of the size of the working roadmap occurs when the planning finishes with either success or cancellation, and also when the iteration loop of the iterative planning [21] is renewed. After finishing planning, it takes some time to start execution to optimize the connected path of transition and replanned paths in Fig. 5c.

From those planning results, we demonstrated the capability of online replanning and also the planner can cope with sudden displacement and appearance that may happen due to limited capacity of perception system.

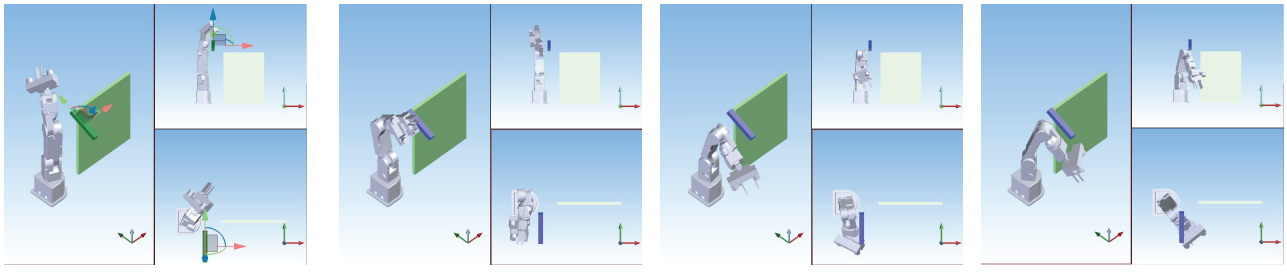
VI. Conclusions

We have proposed an online replanning and execution method for reactive collision avoidance in changing environments. By introducing two threads of execution and planning, the planner can generate alternative paths reactively without unnecessary suspension of path execution. The goal configuration can also be modified during path execution thanks to the reactiveness of the proposed framework. To make use of the knowledge accumulated during the replanning, we employ a mechanism for roadmap reuse. After introducing the general interface with the planner, we have conducted simulations to demonstrate that the planner can replan the motion online and also can deal with discrete environmental



(a) Initial configuration (b) (c) (d) Goal configuration

Fig. 8. Motions without environmental changes



(a) Obstacle moved during execution (b) Executing replanned path (c) (d) Goal configuration

Fig. 9. Planning results with environmental changes

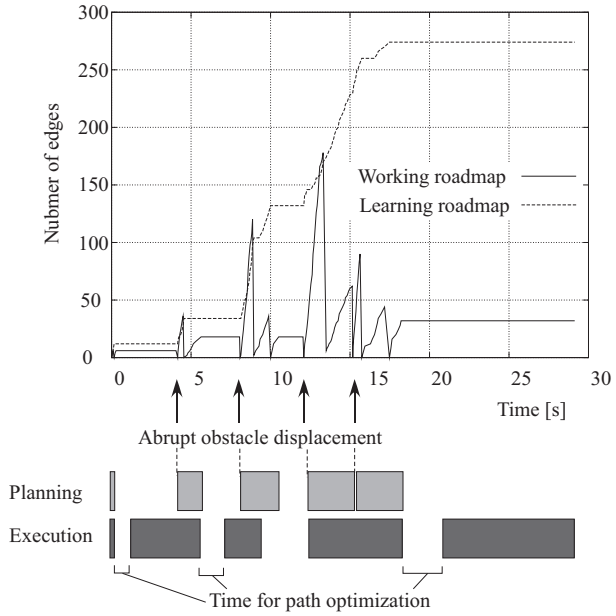


Fig. 10. Time evolution of the number of edges in roadmaps and the time chart of activation of the Execution and Planning threads for abrupt displacement of obstacles.

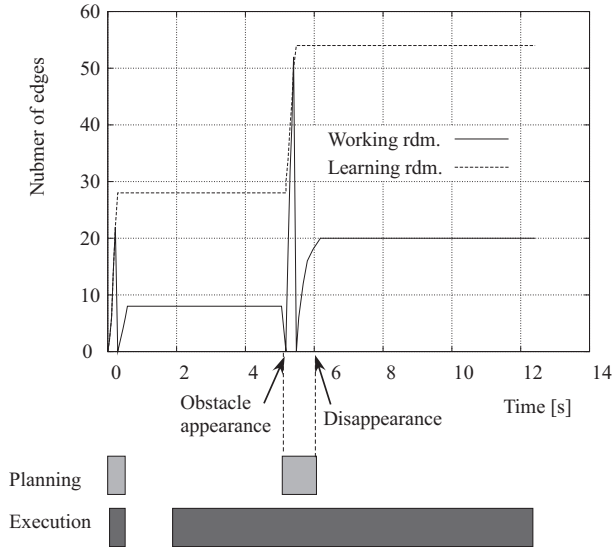


Fig. 11. The case of abrupt appearance and disappearance of an object.

changes.

The proposed method has limitations in continuously moving obstacles and robot motions with high dynamic effects. To make the proposed system more robust against this situation, we will investigate the combination with reactive path deformation and efficient usage of two-stage planning with appropriate estimation of dynamics. We will also study the close and efficient interaction with external perception system and verify the effectiveness of the proposed method with real robot hardware.

ACKNOWLEDGMENTS

The authors thank to Etienne Ferré and Ambroise Confetti of Kineo CAM, and Florent Lamiraux of LAAS-CNRS for their precious support for the software developments. This work has partially been supported by NEDO Project on Development of Software Platform for Robot Intelligence and Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for Scientific Research (B), 21300078, 2009.

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] J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, , and Inoue, "Dynamically-stable motion planning for humanoid robots," *Autonomous Robots*, vol. 12, no. 1, pp. 105–118, 2002.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Proc. 2006 IEEE Int. Conf. on Robotics and Automation*, 2006, pp. 1243 – 1248.

[10] D. Ferguson and A. Stent, "Anytime rrts," in *Proc. 2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006, pp. 5369 – 5375.

[11] 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.

[12] 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.

[13] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proc. 2006 IEEE Int. Conf. on Robotics and Automation*, 2006, pp. 2366–2371.

[14] J. van den Berg and M. Overmars, "Planning time-minimal safe paths amidst unpredictably moving obstacles," *Int. J. of Robotics Research*, vol. 27, no. 11-12, pp. 1274 – 1294, 2008.

[15] 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.

[16] S. PETTI and T. FRAICHARD, "Partial motion planning framework for reactive planning within dynamic environments," in *Proc. of the IFAC/AAAI Int. Conf. on Informatics in Control, Automation and Robotics*, 2005.

[17] 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.

[18] D. Hsu, J.-C. Latombe, and S. Sorkin, "Placing a robot manipulator amid obstacles for optimized execution," in *Proc. 1999 Int. Symp. on Assembly and Task Planning*, 1999, pp. 280–285.

[19] J.-P. Laumond, "Kineo CAM: a success story of motion planning algorithms," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 90–93, 2006.

[20] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W.-K. Yoon, "RT-middleware: Distributed component middleware for RT (robot technology)," in *Proc. 2005 IEEE/RSJ Int. Conf. On Intelligent Robots And Systems (IROS2005)*, 2005, pp. 3555–3560.

[21] E. Ferré and J.-P. Laumond, "An iterative diffusion algorithm for part disassembly," in *Proc. 2004 IEEE Int. Conf. on Robotics and Automation*, 2004, pp. 3149–3154.