Getting There: Using Environment Objects To Facilitate Task Completion In Unknown Environments

Martin Levihn Henrik Christensen

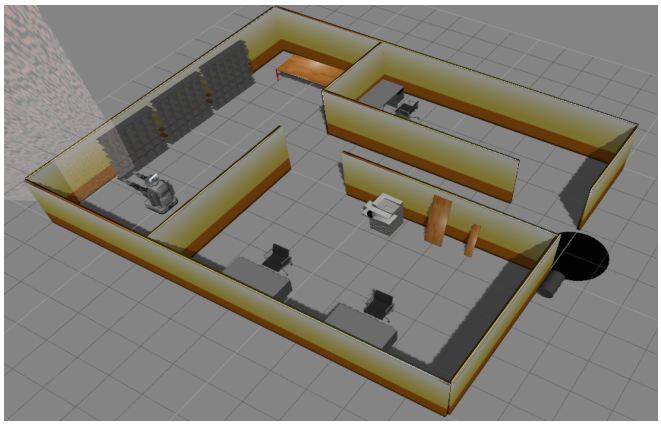
***Abstract*— Robots should be able to reason about using environment objects to assist task completion. If faced with an unexpected situation, such as a liquid spill obstructing the only path to the goal, robots should be able to reason about using environment objects to overcome the obstruction rather than simply declaring failure. In this paper we present our preliminary results for a framework that enables robots to make progress towards task goals even in the presence of obstructions. The framework enables robots to determine online if an obstruction can be circumvented or needs to be overcome. For the latter case, the framework guides the robot to search for and utilize an environment object to overcome the obstruction.**

I. Introduction

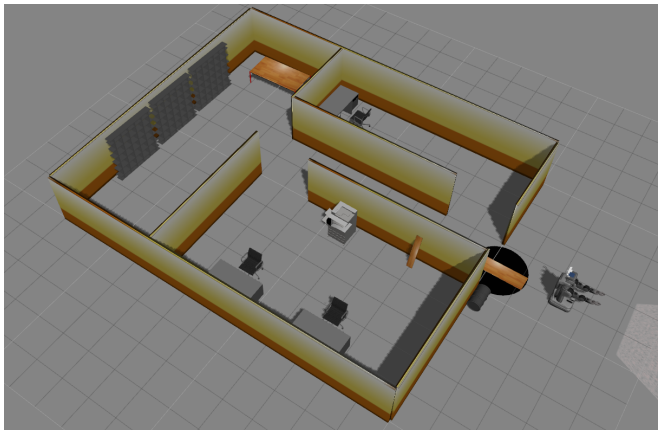
Experienced humans do not hesitate to *use their environments*. Robots shouldn't either. Suppose one is trapped in a room with burning gasoline. The human searches the environment for something that can be placed over the gasoline, finds a long enough board, places it over the fire and escapes. Typically, such situations can not be anticipated a-priori and no predetermined action plan exists. Rather, the encountered situation and present objects have to guide the actions *online*. Possessing such capabilities becomes essential for robots that are expected to operate autonomously in complex, unstructured environments such as disaster areas.

In this paper we present the first framework that allows a robot to reason *online* about using an environment object to facilitate its task completion. To understand the value of such a framework, consider the example visualized in Figure 1(a) in which the robot is tasked with escaping the building. As typical for such cases, we assume that the robot has access to a map containing the static environment properties, but does not know the existence, size or location of non-static objects. Not knowing that an oil barrel fell over, blocking the only exit, the robot computes a motion plan for escaping the building using the exit. As the robot executes the motion plan it obtains sensor readings about the environment and eventually detects the oil spill. Realizing that there is no other way to exit the building, the robot searches its environment. Finding a long enough board in a neighboring room, the robot grasp a board, places it over the oil spill and escapes. Figure 1(b) visualizes the final configuration of the environment.

While full autonomous execution of this example is work in progress, we present the overall framework and subroutines necessary to achieve such behavior. Our framework builds upon execution monitoring and constraint relaxed planning to determine if the current motion plan is obstructed and, if that is the case, whether the obstruction can be avoided or needs to be resolved. To resolve an obstruction, the framework determines necessary object properties for the constraint at hand (e.g. dimensions) and searches for such an object in the environment. If such an object is found, the algorithm guides the robot to pick it up and use it to overcome the obstruction.



(a) Environment Configuration.



(b) Final Environment Configuration.

Fig. 1. Example execution of the proposed framework.

The remainder of the paper is organized as follows: Section II presents related work and Section III provides an overview of the proposed framework. After discussing implementation details in Section IV, the paper concludes with final remarks in Section V.

The authors are affiliated with the Institute for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, Georgia 30332, USA. Email: levihn@gatech.edu, hic@cc.gatech.edu

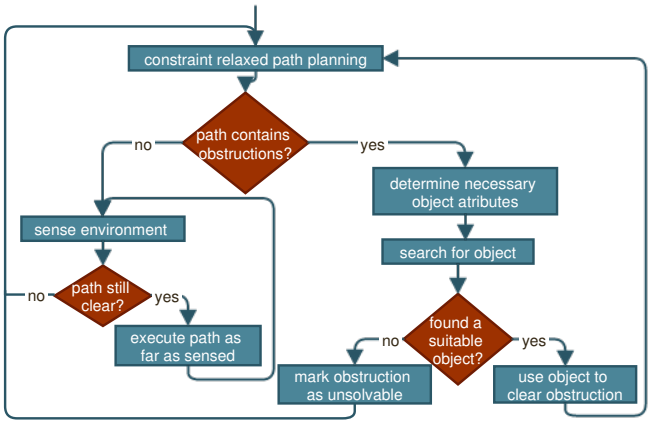


Fig. 2. Flowchart of the proposed framework.

II. Related Work

Most existing forms of robotic object use are focused on accurate positioning and control of specific tools such as welding instruments [5], spray guns [4], drills [7] and surgical instruments [3, 6]. In all of these scenarios the robot performs a well defined task with the tool. While the control methods developed for these scenarios are complementary to the proposed system, we do not assume that the robot is given a specific task to accomplish with a specific object. Instead, the robot has to autonomously determine if and how it needs to use environment objects to accomplish its overall task.

As an initial step towards this goal, we presented a system that allowed a HRP-2 robot to autonomously utilize environment objects to create itself a path in [11]. The robot used a box to create a stair step for itself and placed a board on the ground to cross a gap. Similarly, in [9] we presented a planning system that allowed a robot to reason about force transmission properties of environment objects. However, both methods required full a-priori environment knowledge.

In contrast, the framework presented in this paper is designed for realistic cases in which the robot does not possess full a-priori knowledge but rather needs to obtain the necessary information online.

III. Overview

We now provide an overview of the proposed system before discussing implementation details in the following section. We assume that the world is static and that the robot has access to a map containing immobile environment objects such as walls and stairs. We do not assume that the robot has knowledge of the existence or location of manipulable environment objects prior to any sensing actions.

A. Framework

To obtain an initial motion plan, the framework uses a constraint relaxed path planning system similar to [11]. In contrast to traditional motion planning systems that attempt to find a sound path to the goal [8], constraint relaxed planning systems might return a path that violates robot constraints, such as moving through obstacles. While the resulting path might not directly be executable by the robot, it provides two crucial insights. First, it establishes whether a low cost path to the goal without obstructions exists. Second, if no such paths exists and obstructions need to be overcome, it provides information to subsequent planning steps about where and how exactly the environment needs to be modified.

Dependent on the output of the constraint relaxed path planning system, the framework branches. If the constraint relaxed planning step finds a path to the goal without obstructions, the robot is tasked with moving along the path while continuously sensing the environment for potential obstructions. In case an obstruction interesting the current path is detected, the algorithm re-plans a path to the goal.

In case either the initial path or the updated path contains an obstruction (that is no low cost path circumventing the obstruction could be determined), the framework proceeds by guiding the robot to search the environment for an object that can be used to overcome the obstruction. To achieve this, the framework first determines the necessary properties any suitable object needs to have and then searches the environment for such an object. Note that this process stands in contrast to traditional object recognition problems (*e.g.* [12]) in which the task is to find a specific object. Here, the goal is to find *any* object that is usable by the robot. If a suitable object is found, the frameworks proceeds by guiding the robot to use the object to overcome the obstruction. If this process fails, the obstruction is marked as unresolvable. In either case, the algorithm loops. Figure 2 summarizes the proposed framework.

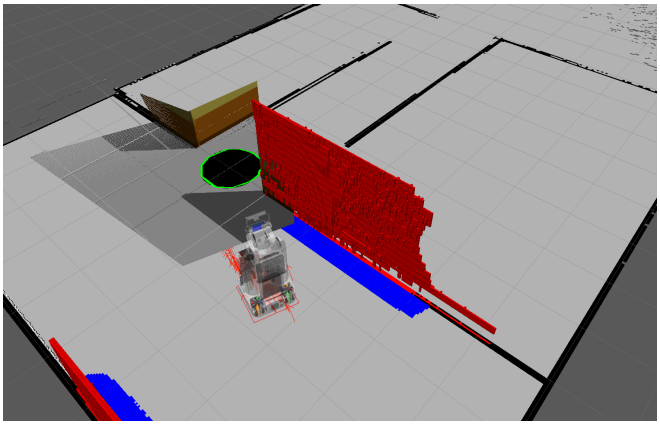
IV. Implementation

While the previous section provided a general overview of the proposed framework, we now describe details of our actual implementation and demonstrate example outputs of our implementation for a simple environment. We implemented the proposed framework in simulation on the PR2 robot using Gazebo and ROS [13]. We focused on cases where liquid obstructions could prevent the robot from reaching its goal.

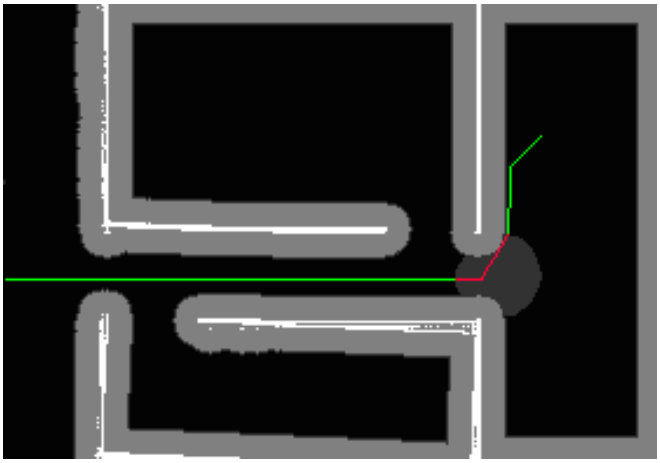
A. Detecting Obstructions

To detect liquid obstructions in the environment, we are utilizing a Kinect sensor mounted to the head of the PR2. First, we are registering the point-cloud information obtained by the Kinect's depth sensor with the color information collected by the Kinect's RGB camera. We then convert the resulting data structure into an image where each pixel is also linked to its depth information. This image is then thresholded according to color differences to the ground. If a significant difference is detected, the depth information associated with the pixels is used to obtain the 3D coordinates of the obstruction's bounding polygon.

While this process is sufficient for cases where the robot either sees the complete obstruction or no obstruction at all, it might cause many unnecessary calls to the path planning system for obstructions that are not being observed in its entirety from the beginning. To minimize the occurrence of partial obstruction reportings, our implementation does not



(a) Obstruction detection. The green polygon indicates the detected obstruction.



(b) Constraint relaxed planning step output. The red portion of the plan indicates a constraint violation, requiring the robot to resolve the constraint prior to following this part of the plan.

Fig. 3. Obstruction detection and constraint relaxed planning step examples.

forward the obstruction information to the planning system until either the obstruction's dimensions do not increase anymore, or the robot is getting too close to the obstruction. Figure 3(a) visualizes an example output of this obstruction detection method for a simplified version of the scenario shown in Figure 1.

B. Constraint Relaxed Planning

If the obstruction detection algorithm reports an obstruction to the planning system, the obstruction is added to the internal cost map and re-planning is triggered. In our implementation, the constraint relaxed planning step is realized as an A* search on a 2D cost map presentation of the environment. As mentioned above, collisions with obstructions are considered as soft constraints rather than hard constraints. To avoid unnecessary constraint violations, a heuristic cost penalty is applied for each initial intersection with an obstruction. Figure 3(b) shows the output of the constraint relaxed planning step following the detection of the oil spill in Figure 3(a). The output indicated that the robot has to cross the oil spill in order to reach the goal.

C. Object Search

Given the output of the constraint relaxed planning step, the system now abandons the process of attempting to reach the goal through pure navigation and the robot is tasked with finding a suitable object in the environment to help it cross the oil spill. As mentioned above, in contrast to most existing research in object detection (*e.g.* [10]), this step does not focus on detecting a known object, but rather on finding *any* object that the robot could utilize to overcome the obstruction. To achieve this, our object detection algorithm is based on the output of the constraint relaxed planning step. Recall that the constraint relaxed planning step returns the exact location and dimensions of the obstruction that is currently blocking the robot. We can now utilize this information to determine the minimum dimensions for a suitable object and use these dimensions to guide the search. For the example visualized in Figure 3, we need to find an object that has at least the length of the oil spill and is at least as wide as the robot base. While reasoning about most likely locations of candidate objects is an interesting research area in itself (*e.g.* [14, 15]), our implementation takes advantage of simple heuristics to explore the environment such as on-the spot rotations and wall following.

Using the just determined dimensionality requirements, the algorithm takes scans of the environment, segments the resulting point-clouds into clusters and rejects all clusters not fulfilling the size requirements. Further, any clusters that are above a size threshold, indicating that the robot would likely not be able to manipulate the corresponding object, are rejected. The remaining clusters, representing potential candidate objects, are then sorted based on a custom cost function. We used a scoring function that attempts to capture the notion of "manipulable" using surface smoothness and object width. In the order defined by the cost function, the robot then attempts to use the objects to overcome the obstruction.

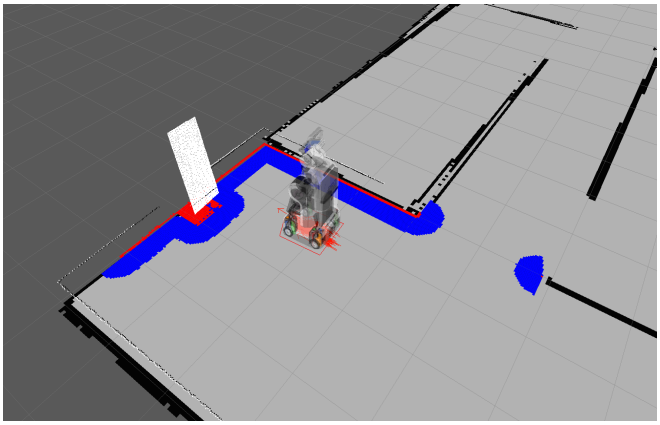
Figure 4(a) visualizes an example output of the obstacle detection method.

D. Grasping and Dropping

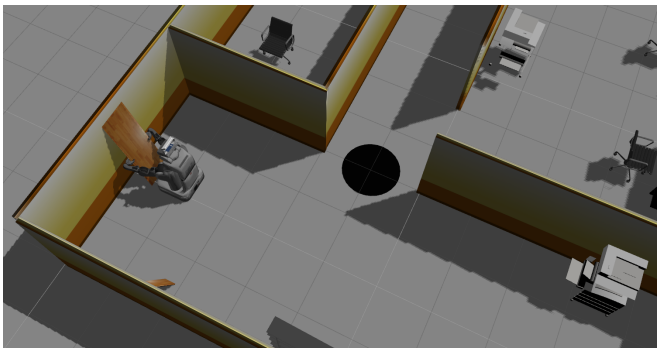
If a candidate object has been found, the robot needs to navigate to it and grasp it. For navigation path planning without the risk of needing to overcome obstructions, we are using the default move_base package provided in the ROS navigation stack [1].

In order to grasp the object, we use the cluster information to determine pre-grasp configurations for the grippers. These configurations are computed to be $5cm$ from the edges of the cluster on each side. Given these pre-grasp configurations we utilize the MoveIt! package [2] to move the grippers into those configurations. The arms are then controlled to move the grippers inwards to establish contact with the object. Upon contact, the grippers are closed and the shoulder joint of the each arm controlled such that the object will be lifted from the ground.

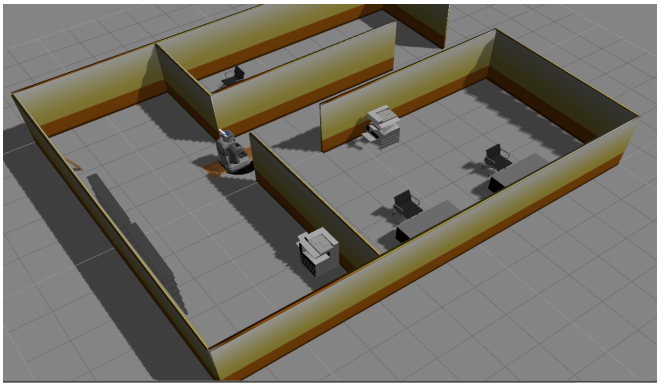
If the object is successfully grasped, the robot moves to the obstruction, aligns itself with the obstruction according to the initial path direction and executes a drop motion.



(a) Robot detects suitable obstacle.



(b) Robot executes grasping subroutine.



(c) After a successful drop of the object the algorithm loops and a path using the object is found.

Fig. 4. Constraint resolution example.

We implemented the drop motion by reverting the grasp motion with the addition of a slight motion of the robot base in the direction of the obstruction. This is done to ensure that the object falls in the correct direction. If the drop was successful, the algorithm marks the new location of the object as traversable space and re-starts the constraint relaxed planning system. Figure 4(c) shows the behavior of the robot after a successful drop of the object.

V. Conclusion

While this paper reflects work-in-progress, to our knowledge, it presents the first framework that allows robots to reason *online* about using environment objects to help them achieve their goal.

We anticipate that the final implementation of the proposed framework will also enable the robot to test relevant object properties, such as applying a certain force to the object prior to deciding to use it as a bridge. We are planning to provide an open-source implementation of our framework to allow other researchers to extended the overall framework or replace individual subroutines with their own.

We expect that future work based on the concepts presented in this paper will allow robots to achieve the intelligent goal-orientated behavior which is characteristic of human beings.

Acknowledgments

This work is dedicated to the memory of Mike Stilman, whose encouragement, support and enthusiasm will never be forgotten.

This work was supported by the ONR under grant N000141210143.

References

[1] move_base package. http://wiki.ros.org/move_base. Accessed: 2014-10-25.

[2] Moveit! package. http://moveit.ros.org. Accessed: 2014-10-25.

[3] Ron Alterovitz, Ken Goldberg, and Allison Okamura. Planning for steerable bevel-tip needle insertion through 2d soft tissue with obstacles. In *IEEE Int. Conf. on Robotics and Automation*, 2005.

[4] Heping Chen, Weihua Sheng, Ning Xi, Mumin Song, and Yifan Chen. Automated robot trajectory planning for spray painting of free-form surfaces in automotive manufacturing. In *IEEE Int. Conf. on Robotics and Automation*, 2002.

[5] George E Cook. Robotic arc welding: research in sensory feedback control. *Industrial Electronics, IEEE Transactions on*, (3), 1983.

[6] BL Davies, SJ Harris, WJ Lin, RD Hibberd, R Middleton, and JC Cobb. Active compliance in robotic surgerythe use of force control as a dynamic constraint. *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, 211(4), 1997.

[7] Myron A Diftler, CJ Culbert, RO Ambrose, R Platt Jr, and WJ Bluethmann. Evolution of the nasa/darpa robonaut control system. In *IEEE Int. Conf. on Robotics and Automation*, 2003.

[8] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[9] M. Levihn and M. Stilman. Using environment objects as tools: Unconventional door opening. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[10] Martin Levihn, Matthew Dutton, Alexander Trevor, and Mike Stilman. Detecting partially occluded objects via segmentation and validation. In *IEEE Workshop on Robot Vision (WoRV)*, 2013.

[11] Martin Levihn, Koichi Nishiwaki, Satoshi Kagami, and Mike Stilman. Autonomous environment manipulation to assist humanoid locomotion. In *IEEE Int. Conf. on Robotics and Automation*, 2014.

[12] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[13] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[14] Lawson LS Wong, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Manipulation-based active search for occluded objects. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2814–2819. IEEE, 2013.

[15] Lawson LS Wong, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Not seeing is also believing: Combining object and metric spatial information. ICRA, 2014.