2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Combining Planning Techniques for Manipulation Using Realtime Perception

Ioan A. Şucan[1] Mrinal Kalakrishnan[2] Sachin Chitta[3]

***Abstract*— We present a novel combination of motion planning techniques to compute motion plans for robotic arms. We compute plans that move the arm as close as possible to the goal region using sampling-based planning and then switch to a trajectory optimization technique for the last few centimeters necessary to reach the goal region. This combination allows fast computation and safe execution of motion plans even when the goals are very close to objects in the environment. The system incorporates realtime sensory inputs and correctly deals with occlusions that can occur when robot body parts block the sensor view of the environment. The system is tested on a 7 degree-of-freedom robot arm with sensory input from a tilting laser scanner that provides 3D information about the environment.**

I. Introduction

Motion planning for manipulation has gained a great deal of attention recently [1]–[7]. Most of this work presents integrated systems intended for manipulating household items in indoor environments, under varying levels of assumptions. These assumptions usually relate to the way the environment around the manipulation platform is specified. The environments are usually generated from CAD representations or known information about the obstacles. For instance, a higher level representation of the environment can be constructed by identifying known types of objects in the environment using sensed data from a camera on the ceiling [7]. While this approach has its merits, it is limited to objects that are known and can be identified from partial view data. Previous work on planning with real sensor data exists as well, but mostly for outdoor mobile platforms with no manipulators [8]–[10].

The problem of constructing the environment in realtime from noisy sensor data while planning for robot arms has not gained as much attention. Recent work [1] explored quick *replanning* using sampling-based planners [11], [12] with an environment generated from real laser-scanned sensor data, in the presence of moving obstacles. However, the problem of planning to goals that are close to or touching perceived obstacles has not been explored in detail. This is a difficult problem in manipulation since the planner needs to balance the need to get to the goal while staying away from obstacles.

The problem we are dealing with is exemplified in Figure 1. Due to noise in sensor data and approximations we are forced to make for safety reasons (adding padding to the environment data based on the error margin in the sensor specification), states that are close to objects can seem to be in collision. We may need to move a manipulator to such states, for instance, when we try to grasp an object. In some cases, depending on the object we intend to grasp, we may even accept touching of the object, so the goal state would not only seem to be in collision, but actually be in collision.

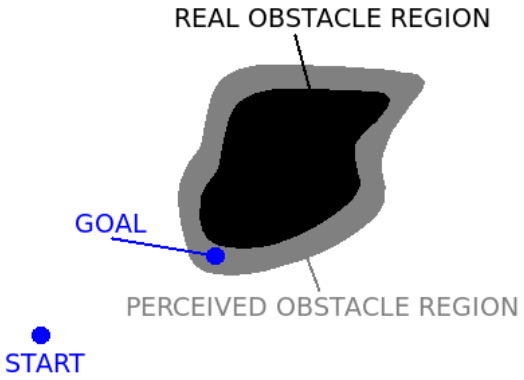

Fig. 1. Stylized representation of the configuration space of a possible motion planning task.

In this paper, we present an integrated system to address these problems in manipulation planning. Our focus is on developing planners that can function well in cluttered indoor environments while handling dynamic obstacles. We use a combination of planners, based on the task to be executed. We use fast sampling-based planners [11], [12] for *non-contact* tasks, including getting close to the goal region, when trying to grasp an object. Once close to the goal, we continue with an optimization-based approach called `CHOMP` [2] to complete the motion plan. `CHOMP` allows us to move to the goal pose while minimizing contacts with the environment. When the starting state is perceived to be in collision, `CHOMP` is also used to move to a nearby state where there are no collisions.

The system we developed is modular and implemented using the `ROS`[1] framework. It can take inputs from a variety of sensing devices and output generic plans that can be tailored to serve as input to the specific controllers for a robotic platform. It has been interfaced to three different planners demonstrating its versatility in using different implementations of planners.

We demonstrate the application of this system on the `PR2` – a service robot designed to function in human environments

[1] I. A. Şucan is with Rice University, Houston, TX 77005 `isucan@rice.edu`
[2] Mrinal Kalakrishnan is with University of Southern California, Los Angeles, CA 90089 `kalakris@usc.edu`
[3] Sachin Chitta is with Willow Garage, Inc., Menlo Park, CA 94025 `sachinc@willowgarage.com`

[1] `http://www.ros.org`

[1]. This robot has an extensive sensor suite that provides realtime updates about changes in the environment. We use this information to build a realtime representation of the world that deals with sensor noise and occlusions. Using this representation, we are able to create and execute plans while being aware of dynamic changes in the environment.

The contribution of this work is two-fold: (a) in combining planners to address the problem of performing manipulation tasks in cluttered environments using sensed data and (b) in handling occlusion and noisy sensor data correctly. The system correctly handles occlusions in realtime by explicitly detecting and labeling occluded parts of the environment. The system can also deal with noisy laser scanner data and remove *shadowing or veiling effects* that are a characteristic of such scanners, especially in indoor environments. The system has been integrated into a wide variety of high-level tasks, some of which are presented in Section VI.

This paper is structured as follows: in Section II we describe in detail the intended capability of the system we present. We then continue to present the perception and planning pipelines employed by the system, in sections III and IV. We present several applications of our approach including planning for grasping and planning with a large object grasped in the hand in Section VI. Conclusions follow in Section VII.

Fig. 2. The PR2 Robotic Platform.

II. System Architecture

We aim to build a planning and execution system that is general, robust and easy to use. This requires the design of a flexible architecture. In particular, it is important to define interfaces to the different components of the system that are generic enough to deal with a wide variety of robots. There are a minimum set of requirements that any motion planning system for manipulation must implement. These include the following:

1) A standard description of the robot that can be used to build kinematic and collision models for the robot
2) Interfaces to lower-level controllers that can execute the plans specified by the planner
3) Generic sensing interfaces that serve as virtual sensors that are independent of the hardware implementation of the sensors
4) Kinematic models that can return forward, inverse and differential kinematic solutions
5) A standard interface to the current robot state specifying positions of all joints in the robot
6) Interfaces to call planners with a desired goal for a subset of joints or links on the robot

Our system implements these interfaces using the ROS framework. A *robot description* provides complete kinematic, collision and visual information for the robot. This information is used to build kinematic, planning and collision models.

Since we are interested in tasks where constrained goals may be specified for the arm, the interface for specifying goals allows them to be specified as kinematic constraints. These constraints can be of two types:

- Joint constraints. These constraints simply specify lower and upper bounds for a particular joint. Having the lower and upper bounds equal essentially specifies we are interested in reaching a specific joint value, which would be the case if we know the goal state exactly.
- Cartesian pose constraints. These constraints specify limits for the pose of a particular link. The user specifies which degrees of freedom are to be constrained (X, Y, Z, roll, pitch, yaw) and bounds for these degrees of freedom. Such a constraint is useful if we need to move to a grasping pose.

Any non-empty set of constraints (there can be multiple pose and/or joint constraints) defines a goal. If the constraints are contradictory and the contradiction can be detected, the code that interprets them warns the user.

In certain cases it may be useful to impose constraints on the entire motion of the robot, at every state it passes through: for instance, moving the arm while holding an open container requires little variation in the roll of the end-effector link. To allow this, the same set of constraints used for goals is used to optionally impose constraints on the complete path.

By default, the system will not allow any contact with the environment. When grasping however, contact with the environment may be needed. For this reason, the user has the option to specify regions in space (bounded by boxes, spheres, cylinders or meshes) where contacts are allowed for a specified set of links, up to a maximal penetration depth.

We will now present in detail the perception and motion planning pipelines used in this system.

III. Perception Pipeline

A perception pipeline for fast replanning was presented in [1]. In this section, we build upon that pipeline and define a generic framework that can deal with a wide variety of sensors. This work is different from [1] in two critical aspects:

1) It accounts for occlusions correctly by maintaining a model of the environment

2) It deals with noisy sensor data, especially data from scanning lasers, by removing noise using knowledge of the robot model.

The perception pipeline performs the key task of creating a representation of the world that can be used for collision checking. We aim to create a representation that can be updated in realtime, is easily accessible for collision checking and deals correctly with occlusions. We also aim to design the interface to the pipeline to be generic enough to incorporate a wide variety of sensor inputs.

A. Sensor Input

The raw input received from the sensor is in the form of a point cloud: a set of points in space that correspond to observed objects in the environment. Most 3D sensors provide information in this format and can be easily plugged into our system. For our implementation on the PR2, the system was interfaced with two different sensors: a Videre stereo camera with projective texture and a tilting Hokuyo laser scanner. The laser sensor is mounted on a tilting stage and it moves up and down at a specified velocity. The viewing angle of the sensor is 270°. This allows the robot to create a detailed representation of the environment in front of it. The stereo camera can provide a denser representation of the environment but was not used as extensively in our implementation.

B. Processing Noisy Point Clouds

The sensor data is often noisy and needs to be processed carefully before being incorporated into the robot's view of the environment. During manipulation, the arms of the robot are frequently in the sensor field of view. The system must then be able to distinguish sensed points that are coincident with points on the robot itself (see Figure 3), i.e. it must be able to infer that sensed points that are on the robot itself are not part of the environment and therefore should not be considered obstacles.

Such points are separated from the sensor input using a simple approach: for each robot link that could potentially be seen by the robot's sensors, the system checks if any points in the input cloud are contained in the geometric shape corresponding to the convex hull of that link. This is a simple test and quickly lets the system partition the sensor data into two parts: points that are part of the environment and points that are part of the robot itself and should not be considered obstacles.

An additional problem, which we refer to as a *shadowing effect*, is especially prevalent in laser scanner data. This problem arises when laser scans graze very slightly the different parts of the body of the robot. Points cast by the edges of the arms now appear to be further away and part of the environment. They form a virtual barrier below the arm, on each side, and greatly constrain the motion of the arm. Furthermore, as the arm moves, these shadow points often appear to lie on the desired path of the arm, so execution is halted. To remove these points, a small padding distance is added to the collision representations of the the robot links. If the line segment between a point in the input cloud and the sensor origin intersects the extended collision representation, the point is classified as a shadow point and removed.

The filtering process described above is also applied for bodies the robot is manipulating: if the robot is holding an object, that object must not be part of the collision environment any more and the shadow points it casts need to be removed. The processed point cloud with shadow points removed can now be processed further for incorporation into the collision environment representation of the robot.

C. Constructing a Collision Environment

The representation of the collision environment, also referred to as a *collision map*, consists of axis aligned cubes where points from the input cloud are incorporated (see Figure 4). Cubes with 1 cm sides were used in the collision map implemented on the PR2. A cubic box is added to the map at a particular location as soon as at least one sensor point is found to occupy the grid cell corresponding to that location. This process is simple and can be executed very quickly.

A proper implementation of a collision environment with frequent sensor updates must deal correctly with occluded data. Replacing the original collision map with only fresh sensor data on every sensor update implies that the map will have no memory about obstacles that may now be occluded. One approach to handling occlusions is to use ray-tracing to trace out every ray coming from the sensors up to a large distance and retain parts of the previous map that are now found to be occluded. This can be very computationally expensive. Since we strive to obtain a perception pipeline that runs close to realtime, we only account for occlusions caused by the robot itself: e.g., when the robot arm is in front of the sensor and parts of the environment are occluded. The simplified approach starts with the previous world representation C (initially empty) and a new world representation N. We first determine the set difference D between the two views, i.e. we look for parts of C that are not part of the new view N, i.e.,

$$D = C - N$$

These parts in D are either moving obstacles that have changed their position or are parts that have become occluded. For every box $d \in D$, we then check whether the line segment between d and the sensor origin intersects a body part of the robot. If it does, the box is considered occluded and is added to the new world view N. N now becomes the current representation of the world that retains a memory of the objects seen previously in the environment but now occluded by parts of the robot. This implementation is fast enough to satisfy our requirements for realtime implementation (it runs around 30Hz - 50Hz with approximately 10000 boxes in the environment).

The collision map is a critical input to the motion planning and motion execution processes. In our implementation on the PR2 robot, the environment was restricted to a box of dimensions 2 m forward, 1.5 m on each side and 2 m upward,

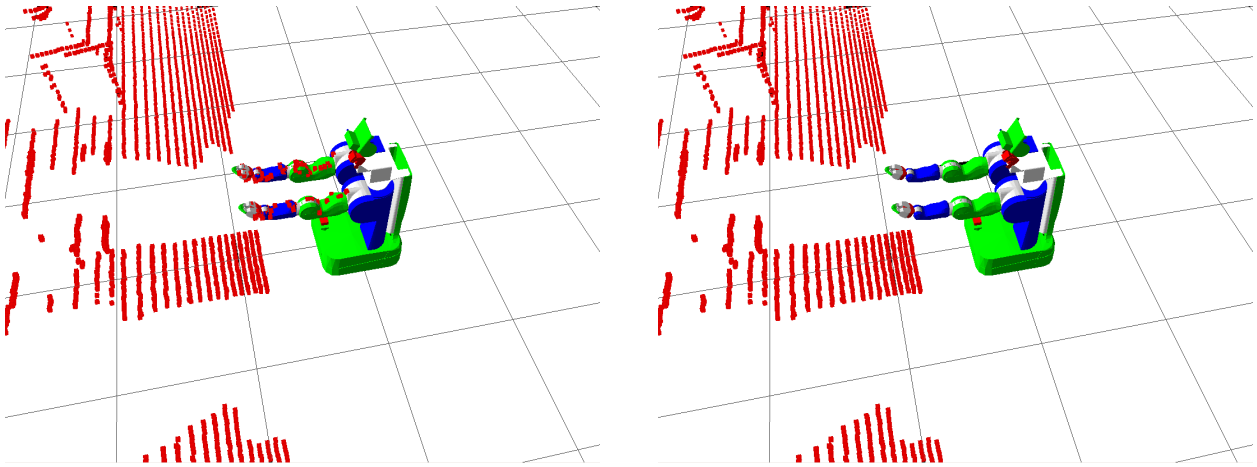

Fig. 3. The robot's world view using its laser without (left) and with (right) filtering.

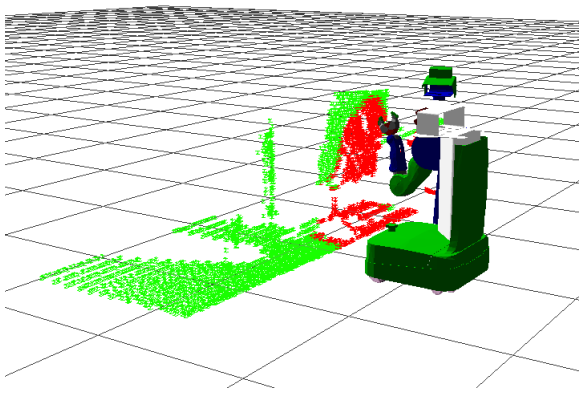

Fig. 4. Example collision map in an office showing retention of occluded data in the environment. Part of the chair is occluded by the arm (marked in red).

with respect to the robot's base. The box contains the entire reachable workspace of the arm. Restricting the environment size has a significant performance impact and helps in the goal of updating this environment in realtime.

IV. Motion Planning Algorithms

To make the process of incorporating different kinds of motion planning algorithms easier, common interface requirements were established that all motion planning processes need to satisfy: offer a `ROS` service[2] for computing a motion plan. In essence, a motion planner maintains a copy of the collision map it received and attempts to satisfy the requests it receives. Currently three classes of motion planners are offered through this interface:

- Sampling-based motion planning, based on the `ompl` library [13]
- Grid-search techniques (A*-like) based on the `sbpl` library [14]
- Trajectory optimization techniques: `CHOMP` (Covariant Hamiltonian Optimization for Motion Planning) [2]

[2]A `ROS` service is akin to a Remote Procedure Call (`RPC`)

The experiments presented in later sections of this paper used the sampling-based motion planners and `CHOMP`.

A. Sampling-based Motion Planning

Sampling-based motion planning works by constructing an approximation of the state space of the robot through sampling and collision checking [11], [12]. This allows computing collision-free solutions very quickly. Typical computation times for the implementation on the `PR2` robot were on the order of 100 milliseconds. Due to the randomized nature of such algorithms, the computed solution paths are not smooth and can look unnatural. Shortening and smoothing steps are thus applied to these paths to get more natural looking, smoother paths [11], [12].

The implementation of sampling-based planners for the experiments presented in this paper used a set of tree-based algorithms from the `ompl` library. The particular planners used included the following:

- `LBKPIECE`, a lazy, bi-directional implementation of `KPIECE` (Kinematic Motion Planning by Interior-Exterior Cell Exploration) [1], [15]
- `SBL` (Single-query Bi-directional probabilistic roadmap planner with Lazy collision checking) [16]
- `KPIECE` [1], [15]
- `RRT` (Rapidly-exploring Random Trees) [17]

When a request for planning is received, the planner to be used is selected based on the type of request. If a goal state can be extracted from the request, a bi-directional planner is preferred (`LBKPIECE` or `SBL`). Otherwise, if a goal state is not available, only single-tree planners can be used (`KPIECE` or `RRT`). The choice of planner to be used depends on the planner's priority. The priority is an integer value that is updated (increased or decreased by 1, within fixed bounds) based on whether the planner was successful or unsuccessful in finding a solution to the request it received. The initial priorities enforce the order in which the planners are listed above: `LBKPIECE` has highest priority and `RRT` has lowest priority.

B. CHOMP

CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [2] is a trajectory optimizer that is based on covariant gradient descent techniques. It can be used to smooth paths generated by sampling-based motion planners. More importantly, it can also optimize a naïve initial trajectory that may be in collision (e.g., a straight line in configuration space) to be collision-free. This property of CHOMP allows it to be used as a stand-alone motion planner in a variety of situations.

We first define a cost function over the trajectory as a sum of smoothness cost and collision cost. Smoothness is typically defined as a sum of squared derivatives along the trajectory for each joint. The collision cost and its gradient are obtained from a Signed Distance Field [2]. These Cartesian collision cost gradients are transformed into joint space using the Jacobian of the robot, and covariant updates ensure that every gradient update to the trajectory is smooth. In our implementation, CHOMP takes between 0.5 and 2 seconds to optimize a trajectory out of collision, depending on the complexity of the problem.

Since the cost function for CHOMP includes a collision cost, trajectories generated using this planner tend to approach an object along a path of low collision cost. Such paths may not be optimal but often represent a good approach direction for the end-effector to grasp the object. We exploit this capability of CHOMP to complement sampling-based planners. In particular, we use CHOMP as part of a two-stage planning process where sampling-based planning is used to quickly generate the initial path to a collision free state close to the goal specified by the user. CHOMP is then used as a second stage planner to complete the plan towards a goal that could possibly be in collision.

V. Motion Execution Monitor

Using the perception pipeline and the motion planning services described above, motion plans can be computed for a robot in an environment that is continuously updated in realtime. These motion plans can then be sent to a trajectory controller which attempts to follow them as best as possible. In this section we describe how the interface between perception, motion planning and control works and we show how it is applied to the PR2 arm as an example.

The motion execution monitor is aware of two motion planners: a long-range planner, which should be fast and safe, but cannot always reach the goal if it is too close to a contact point, and a short-range planner which always moves to the desired goal. In this work, we used sampling-based motion planning (with a preference for bi-directional planners) for the long-range planner and CHOMP for the short-range planner.

If the goal states are in collision, using sampling based bi-directional planning alone may not be feasible. Ignoring collisions up to a small depth with a sampling-based planner will allow the planner to potentially touch the object we are trying to grasp multiple times, thus pushing it away or knocking it over before reaching the final state.

Using a trajectory optimization technique alone would be computationally intensive and too slow for replanning. Fast replanning is needed to account for a rapidly changing environment, especially in the presence of humans.

For these reasons, we use a combination of the two techniques. Sampling based planning is used to plan a path to a feasible state near the goal. It is very fast and allows us to replan quickly if needed. Once we are close to the goal, we use CHOMP to move to the desired final state, relying on its ability to minimize the collision cost for a path to the goal.

Algorithm 1 details the actual workings of the motion planning and execution system. In particular, the system makes an attempt to satisfy the request for a motion plan by making intelligent choices about the types and number of planners to be used. It also uses information about the environment to execute safe plans, i.e. collision checking is continuously used in concert with the changing representation of the environment to ensure that the executed plans do not collide with the environment. We now examine the components in the system in more detail.

Algorithm 1 HANDLEREQUEST(req)

```
GS ← FINDSTATESINGOALREGION(req)
SORT(GS, state with fewest contacts first)
if GS ≠ ∅ and ISSTATEVALID(GS[0]) then
  return RUNLONGRANGE(GS[0])
else
  if (v ← SEARCHVALIDNEARBY(req)) then
    success ← RUNLONGRANGE(v)
    if success then
      if GS ≠ ∅ then
        return RUNSHORTRANGE(GS[0])
      else
        return RUNSHORTRANGE(req)
      end if
    else
      return false
    end if
  else
    return RUNLONGRANGE(req)
  end if
end if
```

1) Attempt to find states in goal region: The request for a motion plan is often to reach a goal region instead of a single goal state. If the request is simply a set of joint constraints, it is easy to sample states in the goal region. E.g., in the implementation on the PR2, we simply construct a state in the "middle" of the goal region (mid-point for bounds in each dimension). If the request constrains 6 degrees of freedom for the end-effector, inverse kinematics can often be used to find states in the goal region. If a valid goal state is found (all constraints are satisfied and there are no collisions), the long-range planner can be run to obtain a solution path. In the case of the PR2, this is the course of action that is usually

taken when the arm is asked to move to a location that is not very close to obstacles.

2) Finding an intermediate state: If no valid goal state was found based on the imposed requests, we run a genetic algorithm (`GAIK` [1], from `ompl`) to find a state that is valid, but as close as possible to the specified goal region. If a valid state is found, it serves as an intermediate state for the motion plan, since it is close to the goal but not really in the goal region. The long-range planner is used to move to this intermediate state. If successful, the short-range planner is then used to move to the desired grasping pose. In the case of the `PR2`, this is the usual course of action taken when attempting to grasp an object.

3) Executing the motion: Algorithm 2 details the execution of the motion plan. A request is sent to a motion planner (long-range or short-range) and the resulting path is forwarded to the trajectory controller. While the path is being executed, it is checked for collision periodically, using new environment data. If the path becomes invalid, the controller is asked to stop the execution and a new motion is to be computed. This setup achieves a simple version of replanning [1].

Algorithm 2 RUNPLANNER($goal$)

```
done = false
while not done do
  done = true
  path ← FINDMOTIONPLAN(goal)
  if ISPATHVALID(path) then
    STARTEXECUTION(path)
    while execution of path not complete do
      if not ISPATHVALID(path) then
        STOPEXECUTION(path)
        done = false
        break
      end if
    end while
  end if
end while
```

Fig. 5. Implementing a grasping task as part of a larger system task of serving drinks at a restaurant

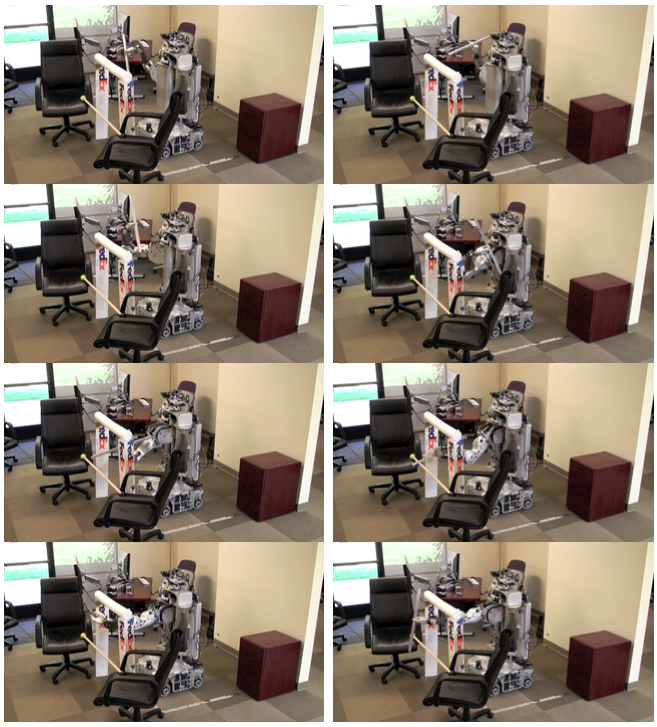

Fig. 6. Manipulating a grasped object in a cluttered environment.

VI. RESULTS

Our system was implemented on the `PR2` platform and has been validated by testing as part of a large set of high-level tasks. The tasks were designed to test the system's ability to grasp objects in cluttered environments, manipulate safely with large objects grasped in the gripper and deal with constraints on the end-effector. The first task was a large systems level task that required the robot to deliver drinks to people in a simulated restaurant (Figure 5). The bottles were placed on kitchen counter-tops and the robot had to grasp and manipulate them without colliding with any part of the environment. This task also involved a sub-task where the robot had to manipulate a bottle without spilling its contents. This was achieved by applying a pose constraint on the end-effector that placed bounds on its orientation to keep the bottle upright. A video of this entire system-level task can be found in [18].

A second task involved manipulating a long metal bar held by the arm of the robot in a cluttered environment. Here, the robot had to manipulate the metal bar through a small opening while avoiding a cluttered environment. The robot had a model of the arm a priori and included it in its kinematic model to account for possible internal collisions and external collisions with objects in the environment as well. Figure 6 shows a series of snapshots of one such planned motion[3]. Note the significant clutter and occlusion problems that arise when the robot is carrying out this task.

Figure 7 shows the *combined* planner in action, moving a

[3]A video of this task can be found at `http://www.kavrakilab.org/willow-demos/`

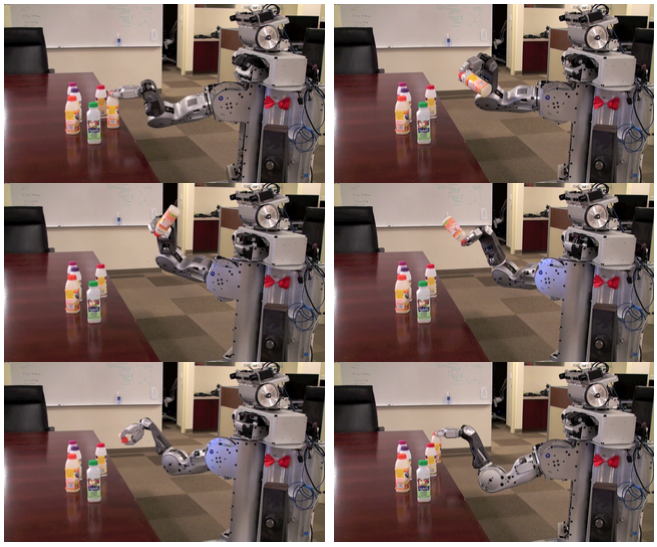

Fig. 7. Combining planners to carry out a manipulation task.

bottle from one position on a table to another. Small errors in the sensed collision environment due to sensor noise can sometimes cause the planner to perceive the initial position of the bottle to be in collision. However, `CHOMP` can be used to first plan away from the table, thus moving out of collision. Sampling-based planning can then plan a path to the goal position. If needed, the system again switches to `CHOMP` to execute the last phase of the manipulation, *i.e.*, placing the bottle back on the table[3]. All of these changes in used planning techniques are not apparent to the user.

VII. Conclusions

We present a system capable of planning and replanning in an environment that is constructed in realtime, using sensed data. Motion planning is achieved through a combination of techniques, taking advantage the speed of computation of sampling-based planning and the ability of trajectory optimization techniques to achieve contact while minimizing collisions. This combination of planners is implemented in a manner that is transparent to the user. The system has been successfully used by a number of researchers as a component in more complex tasks.

The system is general in the sense that its design is modular and can be applied to other similar hardware platforms, not only the `PR2`. All this code is freely available at `http://www.ros.org`.

VIII. Acknowledgments

The authors gratefully acknowledge the contributions of everyone at Willow Garage Inc. and thank Lydia Kavraki for providing valuable comments.

References

[1] R. B. Rusu, I. A. Şucan, B. Gerkey, S. Chitta, M. Beetz, and L. E. Kavraki, "Real-time perception guided motion planning for a personal robot," in *International Conference on Intelligent Robots and Systems*, St. Louis, USA, October 2009.

[2] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation*, 12–17 May 2009, pp. 489–494.

[3] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. J. Kuffner., "Bispace planning: Concurrent multi-space exploration," in *Robotics: Science and Systems*, Zurich, Switzerland 2008.

[4] D. Berenson, S. S. Srinivasa, D. Ferguson, A. Collet, and J. J. Kuffner, "Manipulation planning with workspace goal regions," in *IEEE International Conference on Robotics and Automation*, May 2009.

[5] D. Katz, E. Horrell, Y. Yang, B. Burns, T. Buckley, A. Grishkan, V. Zhylkovskyy, O. Brock, and E. Learned-Miller, "The UMass Mobile Manipulator UMan: An Experimental Platform for Autonomous Mobile Manipulation," in *IEEE Workshop on Manipulation for Human Environments*, Philadelphia, USA, August 2006.

[6] C. Borst, C. Ott, T. Wimbock, B. Brunner, F. Zacharias, B. Baeum, U. Hillenbrand, S. Haddadin, A. Albu-Schaeffer, and G. Hirzinger, "A humanoid upper body system for two-handed manipulation," in *IEEE International Conference on Robotics and Automation*, April 2007, pp. 2766–2767.

[7] S. Srinivasa, D. Ferguson, M. V. Weghe, R. Diankov, D. Berenson, C. Helfrich, and H. Strasdat, "The Robotic Busboy: Steps Towards Developing a Mobile Robotic Home Assistant," in *Intl. Conference on Intelligent Autonomous Systems (IAS-10)*, July 2008.

[8] T. Ihme and U. Ruffler, *Motion Planning Based on Realistic Sensor Data for Six-Legged Robots*, ser. Informatik aktuell, Autonome Mobile Systeme, pp. 247-253. Springer Berlin/Heidelberg, 2007.

[9] P. Sermanet, R. Hadsell, M. Scoffier, U. Muller, and Y. LeCun, "Mapping and planning under uncertainty in mobile robots with long-range perception," in *International Conference on Intelligent Robots and Systems*, 2008, pp. 2525–2530.

[10] Y. Kuwata, G. Fiore, J. Teo, E. Frazzoli, and J. How, "Motion planning for urban driving using RRT," in *International Conference on Intelligent Robots and Systems*, 2008, pp. 1681–1686.

[11] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, June 2005.

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

[13] "http://www.ros.org/wiki/ompl."

[14] "http://www.ros.org/wiki/sbpl."

[15] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *International Workshop on the Algorithmic Foundations of Robotics*, Guanajuato, Mexico, December 2008.

[16] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," *International Journal of Robotics Research*, vol. 6, pp. 403–417, 2003.

[17] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep. 11, 1998.

[18] W. G. Inc., "Intern challenge," http://www.willowgarage.com/blog/2009/08/17/intern-pr2-challenge-2009, 2009.