Comparison of motion planners in an environment with removable obstacles

Carlos Rodríguez and Raúl Suárez
Institute of Industrial and Control Engineering (IOC)
Universitat Politècnica de Catalunya (UPC), Barcelona, Spain.

***Abstract*—This work deals with the problem of motion planning for a robotic system with two arms, considering the possibility of using one arm to remove potential obstacles in order to get a collision-free path to reach a desired object with the other arm. The paper compares different motion planning algorithms based on random sampling methods. In the used framework the planners do not discard the samples that imply collision with removable objects, instead these samples are classified according to the obstacles that produce a collision and then processed to decided whether it is necessary to remove these obstacles to get a proper free path. The efficiency of the motion planners are compared by solving planning problems in three different scenarios.**

I. Introduction

The manipulation of objects in robotics implies a number of associated problems among which it is included the determination of collision free paths to achieve the grasp configuration of a desired object. In this work we have focused on the problem of motion planning, by using motion planning algorithms based on sampling methods in the following context: two arms and two hands work in a shared workspace, a particular object is the object of interest but there are also other objects (both fixed and removable) that may act as obstacles that block the direct access to the object of interest. One robot is in charge of grasping the desired object while the other must remove, if necessary, the removable obstacles that interfere with the path to the object of interest in order to obtain a collision free path. Note that this is a daily problem for humans and so will be for robots.

This work presents a comparison between five different motion planners, all of them based on sampling methods. The planners that are compared below search a geometric path in the configuration space of the robots, taking into account that exist fixed and removable obstacles. The following aspects are considered: the environment is known (this include the positions, rotations, and the 3D models of the robots and objects), the grasp configurations of the removable objects have been computed in advance, and they are used as target configuration for the planners. With these considerations the planners try to find a path to grasp the desired object with one robot, while the other robot is in charged of removing the obstacles from the desired object path.

The planning algorithms have been applied in three different environments in order to make a comparison of the required time to find a solution, the number of generated samples, and the rate of failure.

This work was partially supported by the Spanish Government through the projects DPI2011-22471, DPI2013-40882-P and DPI2014-57757-R.

After this introduction the paper is organized as follows. Section II presents a review of related works and Section III describes the planning testbed, simulated environment, and sofware tools. Section IV presents the used motion planners. Then, Section V presents the experimental results and, finally, Section VI presents some conclusions of the work and presents some topics deserving future work.

II. Related Work

There is significant research work concerning robot motion planning problems [1]. Quite effective general strategies have been developed using sampling-based techniques, and among the most relevant approaches are the *Rapidly-exploring Random Trees planners* (RRT) [2] and *Probabistic Road Map planners* (PRM) [3]. Several variations based on these approaches have been developed to solve different kind of problems in the motion planning field. These original approaches have some problems when there are narrow passages and in order to overcome them several variations were developed, like dynamics domain RRTs [4], retraction base RRTs [5], adaptive workspace biasing [6], a sampling method based on *Principal Component Analysis* [7], and a multi-resolution PRM planner [8], which increase the density of samples in narrow passages and/or regions of interest. In order to speed up query path planning, some variants of PRM planners (e.g. the Lazy PRM planner [9]) build a roadmap without checking the collisions, then, if a collision with the obstacles occurs, the corresponding nodes and edges are removed from the roadmap to start a new search, and the process is repeated until a collision-free path is found.

Some extensions of PRMs include the use of object symmetries in order to improve the performance of the planner [10], and the consideration of scenarios where there exist known obstacles with collision-free paths running around them which have to be connected to the roadmap generated, thus yielding a solution path that skirts the obstacles [11]. Another approach improves the post-processing phase of a planner by combining an efficient PRM with a lazy A* search algorithm that reduces the validation time of the edges [12]. It is important to highlight that the planners mentioned above have been designed to avoid collisions with fixed obstacle. In contrast, other contributions propose motion planners that take into account the objects from the environment that can be removed.

Relevant works that allow robot motion planning considering obstacles include [13], that was a precursor and has shown that motion planning among obstacles is a NP-hard problem, and [14] that proposed a grid based planner that heuristically tries to minimize the cost of moving obstacles out of the way. When the environment is not known, the use of sensors becomes relevant to recognize and identify obstacles in the environment [15], recent approaches use cameras to obtain partial online recognition. One example of these approaches is focused on the identification of visible objects, then calculates the occluded volume and searches the desired object by removing the visible obstacles that more likely occlude it [16]. Following another approach, some works include the possibility of pushing the obstacles out of the way rather than avoiding them (it is a common strategy used for humanoid robots), in order to reach the goal when it is blocked by removable obstacles [17], [18]. Exists an increasing trend in developing robots for environments designed for humans, and exists a wide variety of multi-arm robotic systems (which in general are dual arm systems with anthropomorphic hands or grippers) that may be installed on a mobile platform or be fixed in the environment [19].

In motion planning for multi-arm systems different approaches have been developed, which can be classified into centralized and decoupled approaches. In the centralized approaches multiple arms are considered as a single multi-bodied robot with a number of degrees of freedom (*DOF*) of the whole system, and the planning algorithms find a collisions free path for each arm properly coordinated with the others. In the decoupled approaches each arm is treated as a single independent system and the motion planning process returns first a path for each arm and then requires a coordination phase to avoid potential collisions between the arms when they execute their movements simultaneously. Even when centralized approaches are complete, they have to deal with a planning space with higher number of *DOF* and therefore they are computationally more expensive than the decoupled approaches, which decompose the general problem into smaller problems (i.e. one for each arm) but that, as a drawback, need an additional synchronization phase [2]. Beside, there are works that simultaneously deal with the problems of grasp and motion planning [20].

In general, dual arm systems are used to perform coordinated manipulation tasks including regrasping [21], either arriving to a close kinematic chain (e.g. assembling a nut and a bolt [22] or a peg-in-hole tasks [23], in both cases with each part being manipulated by a different arm and using compliant robot control), or cooperating with open chain coordinated movements (e.g. the problem considered in this paper, the robots must work coordinately in order to properly remove obstacles and make a desired target object be reachable).

III. Planning testbed

A. Description of the testbed

Consider two robots R_i, $i = 1, 2$, in a shared workspace where there is an object of interest O_0 to be grasped and some other removable objects O_j, $j = 1, ..., n$, which can be grasped and removed from the scene by the robots themselves and that may act as obstacles that do not allow access to O_0. In this context, the problem to be solved is: find a geometric path $\pi_{i,0}$ for a robot R_i in order to grasp the desired object O_0 and, if necessary, a set of geometric paths $\pi_{l,j}$, $l = 1, 2$ $l \neq i$ to remove all the obstacles O_j in $\pi_{i,0}$.

In order to find a path the algorithm receives the model of the scene W, that includes the models of each object, the positions and a set of grasping configurations of the removable objects, and the models and the initial configurations of the robots. The algorithm returns an assignment of actions to each robot (that includes the corresponding geometric paths $\pi_{i,j}$ and the set of obstacles $SO_{i,j}$ to be removed) to grasp the desired object and to remove, if necessary, some obstacles.

A motion planner is used to find a path $\pi_{i,0}$ for each robot R_i to grasp the object of interest O_0, but, as a difference with the typical use of planners, a sample of the robot configuration that implies a collision of the robot with any removable object O_j in the environment is not neglected, instead it is considered to build the trajectory in an usual way but associating to it a list with the set of collided obstacles $SO_{i,j}$. The same is done when a local planner checks the validity of a local segment connecting two samples for the trajectory construction, if there are collisions with any removable object O_j it is just added to $SO_{i,j}$. Note that the collision check performed for the configurations of a robot path must be done considering the arm and the hand when the robot is going toward the object to be grasped, and considering the arm, the hand and also the grasped object when this is removed from the scene. The algorithm for finding a path is recursively executed until there is a solution path to remove all O_j in $\pi_{i,0}$ or until an exception occurs for both robots, i.e if the goal object is an obstacle for a given object that has to be removed and vice versa (in this case there is not solution for the problem).

The procedure to find a path $\pi_{i,j}$ for a robot R_i to grasp an object O_j is shown in Algorithm 1. First, it is *loadScene* using W, then, the subset of grasping configurations $SC_{i,j}$ that are kinematically reachable by R_i are selected from a given set SC_j. This is done by the function *selectGrasps*, that simply solves the inverse kinematics of R_i for each configuration in SC_j. Then, a path $\pi_{i,j}$ (with the associate set of obstacles $SO_{i,j}$) is searched for each reachable configuration $\mathbf{c}_k \in SC_{i,j}$ and the one with smaller number of obstacles (computed by the function range) in $SO_{i,j}$ is selected. $\pi_{i,j}$ and $SO_{i,j}$ are generated with the function *MotionPlanner*. The *MotionPlanner* function varies depending on the planner used to solve the problem. The five motion planners evaluated in this work are presented in Section IV.

B. Simulated environment and software tools

The simulated environment used in this work to test the different motion planners correspond to the real work cell of our robotic lab, where there are two 6 *DOF* robots Stäubli TX-90. The robots are located facing each other, one of them is in a fixed location while the other is on a linear mobile platform that adds an additional linear *DOF* but that is not used in this work. The robot R_1 is equipped with a Schunk Anthropomorphic Hand (SAH) with 13 *DOF*, and the robot R_2 is equipped with a Schunk Dextrous Hand (SDH2) with 7 *DOF*. The simulated environment has the corresponding 3D models of the two robots and hands, and a data base with different types of objects in order to perform manipulation tasks.

Algorithm 1: findPath

```
input : $W$, that includes $R_i$,$\mathbf{c}_o$, $SC_j$
output: $\pi_{i,j}$, $SO_{i,j}$
1  $\pi_{i,j}$ = $\emptyset$
2  $SO_{i,j}$ = $\emptyset$
3  loadScene($W$)
4  $SC_{i,j} \leftarrow$ selectGrasps($R_i$, $SC_j$)
5  $max$ = number of removable objects in the workspace
6  for each $\mathbf{c}_k \in SC_{i,j}$ do
7      $\pi_{aux_{i,j}}$, $SO_{aux_{i,j}} \leftarrow$ MotionPlanner($R_i$, $\mathbf{c}_o$, $\mathbf{c}_k$)
8      if range($SO_{aux_{i,j}}$) $< max$ then
9          $\pi_{i,j} \leftarrow \pi_{aux_{i,j}}$
10         $SO_{i,j} \leftarrow SO_{aux_{i,j}}$
11         $max \leftarrow$ range($\mathrm{SO}_{aux_{i,j}}$)
12 return $\pi_{i,j}$, $SO_{i,j}$
```

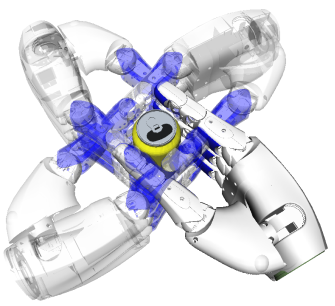


Fig. 1: Example of grasping configurations for a given object.

In order to increase the probability of finding feasible paths, it is considered that the objects can be grasped in different ways with a set of different hand configurations, which are taken as different goal points when the grasp trajectory is built.

The grasping configurations for each object can be obtained using different procedures (see for instance [24], [25]). In this work we assume that the set of possible grasping configurations of an object has been computed in advance and it is provided with the model of the object (the set SC_j). The given grasping configurations of the hand are related to the object reference frame, but since the position of the object in the work environment is known, the grasping configurations can easily be converted to the robot configuration space. Figure 1 shows four different grasp configurations of the SAH grasping a soda can.

The motion planning and simulations have been performed using *The Kautham Project* [26] software tool, it is a home-developed open source environment. *The Kautham Project* provides the developer with several useful tools for the development of planners, like, for instance, random and deterministic sampling methods, metrics to evaluate the performance of planners (number of generated samples, collision check callings, number of nodes in the graph solution, connected components) and simulation tools (including direct and inverse kinematic models of the robots). *The Kautham Project* uses several libraries as *Coin3D* for the graphical rendering, *PQP* for the collision detection, *ROS* for the communication layer, and more recently OMPL has been integrated providing the motion planning tool with more options.

IV. Compared motion planners

This section presents the motion planners to be compared, which work in the configuration space and are based on random sampling methods to generate configuration samples used to search a round trip path from the initial to the goal configuration. The samples or segments that collide with removable objects are not rejected, instead they are added to $SO_{\mathbf{c_i}}$ or $SO_{\mathbf{s_i}}$ respectively, only the samples or segments that collide with fixed objects are discarded. When a path is found the corresponding sets $SO_{\mathbf{c_i}}$ and $SO_{\mathbf{s_i}}$ are added to $SO_{i,j}$.

A. Probabilistic Roadmap

The probabilistic roadmap [27] is a motion planner based on the classic PRM [3], but in this case the random samples are generated around a straight line in the configuration space from the initial to the goal robot configuration.

The PRM follows the single query approach, this is, a geometric path is found when the initial and the goal configurations are connected on the same roadmap. Algorithm 2 describes the PRM planner. The algorithm executes a searching loop until the initial and goal configurations, $\mathbf{c_o}$ and $\mathbf{c_g}$ respectively, are connected in the PRM or a predefined maximum number of configuration samples has been generated. Within this loop, each iteration starts with the generation of a new configuration sample $\mathbf{c_i}$ by using *genConfig* function, and the function *collisionCheck* is used to check whether the sampled configuration implies collisions, and, if this is the case, the collided objects are added to a set of obstacles $SO_{(\mathbf{c_i})}$. If $SO_{(\mathbf{c_i})}$ contains fixed non-removable obstacles, then $\mathbf{c_i}$ is directly rejected; otherwise if $SO_{(\mathbf{c_i})}$ contains only removable obstacles, then $SO_{(\mathbf{c_i})}$ is associated with $\mathbf{c_i}$ and it is added to the set of vertices SV. The nearest neighbor to $\mathbf{c_i}$ in SV, called $\mathbf{c}_{nn}$, is determined and used to generate the segment $\mathbf{s}$ between $\mathbf{c_i}$ and $\mathbf{c}_{nn}$, and the function *localPlanner* is used to check for the existence of collisions when R_i is moved along $\mathbf{s}$, and, as in the case of the sample $\mathbf{c_i}$, if there are collisions the collided obstacles are added to a set of obstacles $SO_{(\mathbf{s})}$. If the obstacles in $SO_{(\mathbf{s})}$ are all removable objects then $SO_{(\mathbf{s})}$ is associate to $\mathbf{s}$ and $\mathbf{s}$ is added to the PRM, otherwise $\mathbf{s}$ is rejected. Then, when $\mathbf{c_o}$ and $\mathbf{c_g}$ are connected in the PRM, the path $\pi_{i,j}$ with minimum number of vertices between them is searched and the set $SO_{i,j}$ that contains the obstacles associated with the configurations $\mathbf{c_i}$ and edges $\mathbf{s}$ included in $\pi_{i,j}$ is computed. Finally the algorithm returns $\pi_{i,j}$ and $SO_{i,j}$.

B. Single-Query Bi-directional Probabilistic Roadmap with Lazy Collision Checking

The single-query bi-directional probabilistic roadmap planner with lazy collision checking (SBL) [28] is based on classic PRMs. It builds two roadmaps from the initial and goal configuration respectively, and it is an adaptive planner able to make great strides in the open areas of the configuration space and small steps in areas where the probability of collisions due to the lazy collision check is larger. Algorithm 3 describes the SBL planner. The algorithm execute a searching loop for a preset number of iterations. The function *expandTrees* expands the two roadmaps generating new samples on each one, evaluates

Algorithm 2: PRM

```
input : $R_i$, $\mathbf{c}_o$, $\mathbf{c}_g$
output: $\pi_{i,j}$, $SO_{i,j}$
1  $SV$ = {$\mathbf{c}_o$ ,$\mathbf{c}_g$}
2  $k$ = 0
3  while ($k < k_m$) $\vee$ ($\mathbf{c}_o$ and $\mathbf{c}_g$ are not connected) do
4      $\mathbf{c_i} \leftarrow$ genConfig()
5      $SO_{(\mathbf{c_i})} \leftarrow$ collisionCheck($\mathbf{c_i}$, $R_i$)
6      if there are not fixed obstacles in $SO_{(\mathbf{c_i})}$ then
7          associate $SO_{(\mathbf{c_i})}$ to $\mathbf{c_i}$
8          add $\mathbf{c_i}$ to $SV$
9          find the nearest neighbor $\mathbf{c}_{nn} \in SV$ to $\mathbf{c_i}$
10         generate the segment $s = \overline{\mathbf{c}\mathbf{c}_{nn}}$
11         $SO_{(\mathbf{s})} \leftarrow$ localPlanner(s)
12         if there are not fixed obstacles in $SO_{(\mathbf{s})}$ then
13             associate $SO_{(\mathbf{s})}$ to s
14             add s to PRM
15         else
16             reject s
17     else
18         reject $\mathbf{c_i}$
19     $k = k + 1$
20 $\pi_{i,j} \leftarrow$ find the path with minimum number of vertices
   in the PRM
21 $SO_{i,j} \leftarrow$ collect the obstacles $SO_{(\mathbf{c}_i)}$ and $SO_{(\mathbf{s}_i)}$
   associate with each vertice $\mathbf{c}_i$ and segment $\mathbf{s}_i$ contained
   in $\pi_{i,j}$
22 return $\pi_{i,j}$, $SO_{i,j}$
```

Algorithm 3: SBL

```
input : $R_i$, $\mathbf{c_o}$, $\mathbf{c_g}$
output: $\pi_{i,j}$, $SO_{i,j}$
1  set $\mathbf{c_o}$ and $\mathbf{c_g}$ as the roots of the trees $\tau_o$ and $\tau_g$
2  repeat
3      if expandTrees($\tau_o, \tau_g, R_i$) then
4          $\tau \leftarrow$ connectTrees($\tau_o, \tau_g$)
5          if $\tau \neq \emptyset$ then
6              $\pi_{i,j} \leftarrow \tau$
7              $SO_{i,j} \leftarrow$ collect the obstacles $SO_{(\mathbf{c}_i)}$ and
               $SO_{(\mathbf{s}_i)}$ associate with each vertice $\mathbf{c}_i$ and
               segment $\mathbf{s}_i$ contained in $\pi_{i,j}$
8              return $\pi_{i,j}$, $SO_{i,j}$
9  until a predefined number of iterations
10 return failure
```

with a lazy collision check whether these samples are valid and tries to connect them to the respective roadmap and returns true if the expansion can be done. Then, the function *connectTrees* is in charge of connecting both roadmaps in order to obtain a geometric path τ from the initial to goal configuration, if there is a path copy τ to $\pi_{i,j}$. Finally the obstacles associated to the path $\pi_{i,j}$ are collected in $SO_{i,j}$.

Algorithm 4: EST

```
input : $R_i$, $\mathbf{c_o}$, $\mathbf{c_g}$
output: $\pi_{i,j}$, $SO_{i,j}$
1  set $\mathbf{c_o}$ and $\mathbf{c_g}$ as the roots of the trees $\tau_o$ and $\tau_g$
2  repeat
3      for $\tau_o$ and $\tau_g$ do
4          choose $\mathbf{c_i}$ from $\tau_k$ with probability $1/w(x)$,
           where {k = o, g}
5          $\mathbf{c_{new}} \leftarrow$ expansion($\mathbf{c_i}, \tau_\mathbf{k}$, $R_i$)
6          if $\mathbf{c_{new}}$ is valid then
7              if distance($\mathbf{c_i}$, $\mathbf{c_{new}}$, $\tau_k$) then
8                  $\tau_k \leftarrow \mathbf{c_{new}}$
9                  $\tau_k \leftarrow$ edge($\mathbf{c_i}$, $\mathbf{c_{new}}$)
10     $\tau \leftarrow$ connectTrees($\tau_o, \tau_g$)
11     if $\tau \neq \emptyset$ then
12         $\pi_{i,j} \leftarrow \tau$
13         $SO_{i,j} \leftarrow$ collect the obstacles $SO_{(\mathbf{c}_i)}$ and
           $SO_{(\mathbf{s}_i)}$ associate with each vertice $\mathbf{c}_i$ and
           segment $\mathbf{s}_i$ contained in $\pi_{i,j}$
14         return $\pi_{i,j}$, $SO_{i,j}$
15 until a predefined number of iterations
16 return failure
```

C. Expansive Space Trees

The expansive space trees planner (EST) [29] grows two trees using the initial and goal configurations as roots. The EST planner performs an exploration in the space by random sampling it from a configuration belonging to one of the trees. The sampler generates samples around the configurations of the trees, if the samples are collision-free then they are connected to the nearest leaf, whatever tree it belongs to. When $\mathbf{c_o}$ and $\mathbf{c_g}$ are in the same tree the algorithm searches a solution path. Algorithm 4 describes the EST planner. First, it sets $\mathbf{c_o}$ as a root of the tree τ_o and $\mathbf{c_g}$ as a root of τ_g; then enters into a loop where a sample $\mathbf{c_i}$ is chosen from one of the trees. Then $\mathbf{c_i}$ is used to find a new configuration $\mathbf{c_{new}}$ using the function *expansion*. If $\mathbf{c_{new}}$ is collision-free, and if it complies with a *distance* criterion it is added to one of the the trees. After that, the *connectTrees* function tries to connect τ_o with τ_g, when they are connected it means that there exists a path from $\mathbf{c_o}$ to $\mathbf{c_g}$. The loop is repeated a predefined number of iteration or until a path between the trees is found, and returns $\pi_{i,j}$ and the corresponding $SO_{i,j}$.

D. Rapidly-Exploring Random Tree planner

The rapidly-exploring random tree planner (RRT) [2] explores the configuration space with a random tree. The tree is rooted at the initial configuration and by using random samples grows the tree to explores the space until a solution path is found. Algorithm 5 describes the RRT which performs the following steps, First $\mathbf{c_o}$ is set as a root of the tree τ and then a loop starts where a new $\mathbf{c_i}$ is randomly generated by using the function *genConfig*. Then, the *collisionCheck* function checks if $\mathbf{c_i}$ is a valid configuration (i.e. if $\mathbf{c_i}$ is collision-free with respect to fixed objects in the environment, and if it is kinematically reachable by the robot). The *nearestNeighbor* function is in charge of computing the distance between $\mathbf{c_i}$ and all the

Algorithm 5: RRT

```
input : $R_i$, $\mathbf{c_o}$, $\mathbf{c_g}$
output: $\pi_{i,j}$, $SO_{i,j}$
1  set $\mathbf{c_o}$ as the root of $\tau$
2  for $k$ = 1 to $K$ do
3      $\mathbf{c_i} \leftarrow$ genConfig()
4      $SO_{(\mathbf{c_i})} \leftarrow$ collisionCheck($\mathbf{c_i}$, $R_i$)
5      if there are not fixed obstacles in $SO_{(\mathbf{c_i})}$ then
6          associate $SO_{(\mathbf{c_i})}$ to $\mathbf{c_i}$
7          $\mathbf{c_{near}} \leftarrow$ nearestNeighbor($\mathbf{c_i}, \tau$)
8          addConfig($\mathbf{c_i}, \mathbf{c_{near}}, \tau$)
9          if $\mathbf{c_i}$ = $\mathbf{c_g}$ in $\tau$ then
10             if solutionPath($\tau$) then
11                 $\pi_{i,j} \leftarrow \tau$
12                 $SO_{i,j} \leftarrow$ collect the obstacles $SO_{(\mathbf{c}_i)}$
                   and $SO_{(\mathbf{s}_i)}$ associate with each vertice
                   $\mathbf{c}_i$ and segment $\mathbf{s}_i$ contained in $\pi_{i,j}$
13                 return $\pi_{i,j}$, $SO_{i,j}$
14     else
15         reject $\mathbf{c_i}$
16 return failure
```

configurations in τ and returns the nearest configuration $\mathbf{c_{near}}$, and connects $\mathbf{c_i}$ with $\mathbf{c_{near}}$ using the function *addConfig*. Then, when the tree includes $\mathbf{c_g}$ the algorithm searches for solution path from $\mathbf{c_o}$ to $\mathbf{c_g}$, using the *solutionPath* function, which returns true if there is a path or false when $\mathbf{c_o}$ and $\mathbf{c_g}$ can not be connected. Finally, Algorithm 5 returns $\pi_{i,j}$ and the corresponding $SO_{i,j}$.

E. Rapidly-Exploring Random Tree Connect

The RRT-Connect planner is an extension of the RRT planner [30], this planner builds two trees in the configuration space using the initial and goal configuration as a root of each tree, the trees rapidly explore the space and cover it proportionally until one of them intercepts the other, this means that a trajectory between the initial and goal configuration has been found. Algorithm 6 describe the RRT-Connect, and follows the same base of the RRT but in this case two trees are built, τ_o and τ_g, one from $\mathbf{c_o}$ and another from $\mathbf{c_g}$, respectively.

After the generation of a sample $\mathbf{c_i}$, the *extend* function selects the configuration $\mathbf{c_t}$ from τ_o nearest to $\mathbf{c_i}$ and attempt to connect them. The function returns one of the following states: *Reached*, when $\mathbf{c_i} = \mathbf{c_t}$, *Added* when $\mathbf{c_i} \neq \mathbf{c_t}$ and $\mathbf{c_i}$ was added to the tree, and *Trapped* when $\mathbf{c_i}$ could not be added to the tree because it implies a collision. If the *extend* function returns a state different from *Trapped*, then it is applied to the other tree τ_g. If in this case the function returns *Reached*, it means that the two trees have been connected. In any other case, the *swap* function exchange the trees to start a new iteration. This process is repeated until reaching a given number of iterations K or until τ_o and τ_g are connected, and in the latter case the Algorithm 6 returns $\pi_{i,j}$ and $SO_{i,j}$.

Algorithm 6: RRT-Connect

```
input : $R_i$, $\mathbf{c_o}$, $\mathbf{c_g}$
output: $\pi_{i,j}$, $SO_{i,j}$
1  set $\mathbf{c_o}$ and $\mathbf{c_g}$ as the roots of $\tau_o$ and $\tau_g$
2  for $k$ = 1 to $K$ do
3      $\mathbf{c_i} \leftarrow$ genConfig()
4      $SO_{(\mathbf{c_i})} \leftarrow$ collisionCheck($\mathbf{c_i}$, $R_i$)
5      associate $SO_{(\mathbf{c_i})}$ to $\mathbf{c_i}$
6      if not extend($\tau_o$, $\mathbf{c_i}$) = Trapped then
7          if extend($\tau_g$, $\mathbf{c_i}$) = Reached then
8              $\pi_{i,j} \leftarrow$ path($\tau_o$, $\tau_g$)
9              $SO_{i,j} \leftarrow$ collect the obstacles $SO_{(\mathbf{c}_i)}$ and
               $SO_{(\mathbf{s}_i)}$ associate with each vertice $\mathbf{c}_i$ and
               segment $\mathbf{s}_i$ contained in $\pi_{i,j}$
10             return $\pi_{i,j}$, $SO_{i,j}$
11     swap($\tau_o$, $\tau_g$)
12 return failure
```

V. Experimental Results

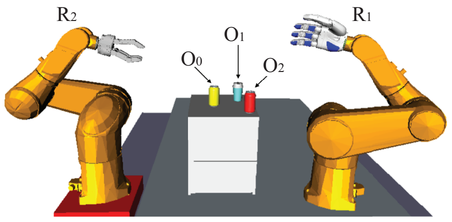
The experiments presented below show the performance of each motion planner in three scenarios with different complexity, the experiments were performed using the testbed described in Section III.

In order to execute the experiments it was considered that each arm is a single independent system and the motion planning process has two phases, first, the paths for each arm are independently determined and, then, a coordination method [31] is used in order to avoid potential collisions between the robots when they execute their movements simultaneously. In the three experiments the robot R_1 is in charge of grasping the goal object O_0 and R_2 is in charge of removing the obstacles.

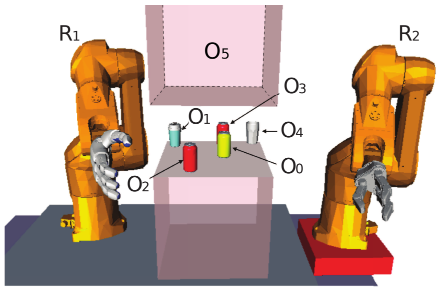
100 runs were computed for each experiment. For each run, the limits have been set to 600 s for the running time and to 10000 generated samples. The experiments were run using a computer with processor Intel I7 2.3 GHz, 4Gb RAM, and Ubuntu OS.

In the first experiment the can of interest lie on a table with some other removable objects around it (see Figure 2a). The grasp configurations for robot R_1 allows only side grasps as shown in Figure 1 (i.e. R_1 cannot grasp the objects from the top), while the gripper in R_2 can grasp the object only from the top. After performing the validation of the grasp reachability, 3 valid grasps configurations were found for the robot R_1 to grasp the goal O_0, and a path for R_1 was computed to reach each valid grasp configuration. One of the grasp configuration collided with the removable object O_1 and the other 2 had more collisions, so the former is selected and a path for R_2 is computed to remove O_1, which was found without any additional collision. All the planners solved always the problem. Figure 3 shows a bar chart representing graphically the average time used to solve the experiment 1.

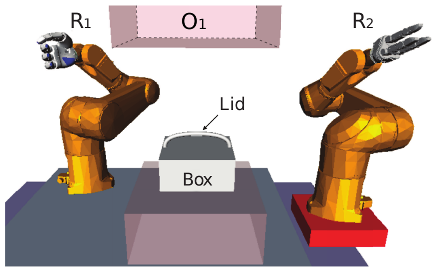
In the second experiment several cans lie on a table being again the yellow can the desired object (see Figure 2b), in this case there is a fixed obstacle O_5 over the table. This variation of the scene generates a narrow passage and increases the



(a) Experiment 1



(b) Experiment 2



(c) Experiment 3

Fig. 2: Setup for each experiment.

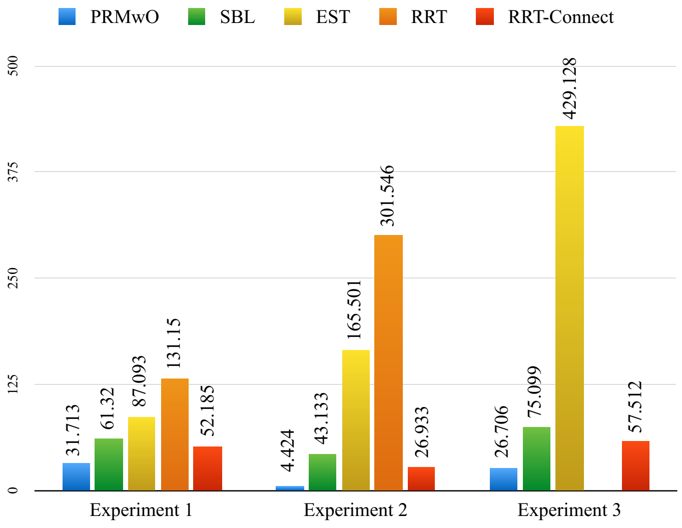


Fig. 3: Bar charts of the average time (s) for the experiments.

difficulty of accessing to O_o. In this experiment only one valid grasp was found for R_1 to grasp O_o, which had a collision with the red can O_2, and O_2 can be removed by R_2 without any additional collision. For this experiment only 2 paths were computed in each task execution, one for R_1 to grasp O_o and one for R_2 to grasp O_2. This is a harder experiment due to the narrow passage to the target, which is reflected in the rates of failures as shown in Table I. The PRM was the only planner able to complete the 100 task executions without failures. Figure 3 shows a bar chart representing graphically the average time used to solve the experiment 2.

In the third experiment the yellow can is located in the center of a box that has a lid, and the scene has also a fixed obstacle O_1 over the box (see Figure 2c). The yellow can can be grasped by R_1 using two grasp configurations, the paths to reach both of them collides only with the lid, so the one that was first obtained was selected, and a path for R_2 to remove the lid was found without any additional collision. The RRT had always failed, sometimes because it exceeded the maximum predefined limit of samples that can be generated and in other cases because it reached the maximum predefined limit time used to find the paths. Figure 3 shows a bar chart representing graphically the average time used to solve the experiment 3. Table I summarizes the average results obtained for each experiment with each motion planner.

VI. Conclusions and Future Works

This work has presented a comparison between five motion planners based on random sampling methods using three scenarios with different complexities. The performances were was compared looking to the require times to find a solution, the numbers of generated samples, and the rates of failures. The PRM planner solved the tasks with the higher performance followed by the RRT-Connect and the SBL, while the RRT had the lower performance, this is because it grows a tree by generating random samples without any criterion, while in the other cases some heuristics help in the search. The SBL, EST, and the RRT-Connect had the advantage over the RRT of growing and expanding two graphs which helps to solve the problems faster. As future work, we consider the problem of dealing with more than one desired object, assuming than each arm has to grasp one particular desired object (always in the presence of obstacles), for instance to assembly them, considering also the possibility of transfering the objects from one robot to the other. Then, analyzing the performance of different planners in this type of problem is a relevant future work.

References

[1] J.-C. Latombe, *Robot motion planning*. Kluwer Academic Publishers, 1991.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[3] L. E. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1996, pp. 566–580.

[4] A. Yershova, L. Jaille, T. Simeon, and S. LaValle, "Dynamic-Domain-RRTs: Efficient Exploration by Controlling the Sampling Domain." in *Proc. IEEE Int. Conf. Robotics and Automation*, 2005, pp. 3856–3861.

[5] L. Zhang and D. Manocha, "An Efficient Retraction-base RRT Planner." in *Proc. IEEE Int. Conf. Robotics and Automation*, 2008, pp. 3743–3750.

[6] J. Zucker, M. Kuffner and J. Bagnell, "Adaptive workspace biasing for sampling-based planners." in *Proc. IEEE Int. Conf. Robotics and Automation*, 2008, pp. 3757–3762.

[7] J. Rosell, L. Cruz, R. Suárez, and A. Pérez, "Importance Sampling based on Adaptive Principal Component Analysis," in *Accepted to the IEEE Int. Symposium on Assembly and Manufacturing*, May 2011.

[8] Y. Yang and O. Brock, "Adapting the sampling distribution in PRM-Planners based on an Approximated Medial Axis." in *Proc. IEEE Int. Conf. Robotics and Automation*, 2004, pp. 4405–4411.

Experiment 1							
Planner	Paths Time (sec)		Total Time (sec)	Samples Generated		Total Samples	Failures %
	R1	R2		R1	R2		
PRM	31.590	0.122	31.713	587	2	589	0
SBL	53.961	7.359	61.320	438	134	572	0
EST	79.557	7.535	87.093	176	20	196	0
RRT	123.623	7.526	131.150	1366	65	1431	0
RRT-Connect	45.218	6.967	52.185	15	5	20	0
Experiment 2							
Planner	Paths Time (sec)		Total Time (sec)	Samples Generated		Total Samples	Failures %
	R1	R2		R1	R2		
PRM	2.737	1.687	4.424	57	9	65	0
SBL	32.783	10.349	43.133	591	247	838	30
EST	150.535	14.967	165.501	765	136	901	40
RRT	191.755	109.791	301.546	3830	3497	7327	70
RRT-Connect	19.474	7.459	26.933	45	11	56	25
Experiment 3							
Planner	Paths Time (sec)		Total Time (sec)	Samples Generated		Total Samples	Failures %
	R1	R2		R1	R2		
PRM	23.275	3.431	26.706	256	62	318	0
SBL	63.848	11.251	75.099	1286	268	1554	0
EST	392.939	36.188	429.128	1952	475	2427	50
RRT	-	-	*	-	-	*	100
RRT-Connect	48.241	9.271	57.512	107	22	129	0

* One of these parameter reached the maximum allowed in all the runs (i.e. time=600 sec or number of samples=10000).

TABLE I: Average results of 100 runs of each motion planner in the three experiments.

[9] R. Bohlin and L. Kavraki, "Path Planning Using Lazy PRM," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2000, pp. 521–528.

[10] E. Cheng, P. Frazzoli and S. LaValle, "Improving the performance of Sampling-Based Planners by using a Symmetry-Exploition Gap Reduction Algorithm." in *Proc. IEEE Int. Conf. Robotics and Automation*, 2004, pp. 4362–4368.

[11] J.-M. Lien and Y. Lu, "Planning Motion in Similar Environments," in *Proc. of Robotics: Science and Systems.*, Seattle, USA, June 2009.

[12] R. Guernane and N. Achour, "Generating optimized paths for motion planning," *Robotics and Autonomous Systems*, pp. 789–800, 2011.

[13] G. Wilfong, "Motion planning in the presence of movable obstacles," in *Proc. of the 4th Annual ACM Symposium on Computational Geometry*, 1988, pp. 279–288.

[14] P. Chen and Y. K. Hwang, "Practical path planning among movable obstacles," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 1, 1991, pp. 444–449.

[15] K. Okada, A. Haneda, H. Nakai, M. Inaba, and H. Inoue, "Environment manipulation planner for humanoid robots using task graph that generates action sequence," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2004, pp. 1174–1179.

[16] M. Dogar, M. Koval, A. Tallavajhula, and S. S., "Object Search by Manipulation." in *Proc. IEEE Int. Conf. Robotics and Automation*, 2013.

[17] M. Stilman, K. Nishiwaki, S. Kagami, and J. Kuffner, "Planning and executing navigation among movable obstacles," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, October 2006.

[18] Y. Kakiuchi, R. Ueda, K. Kobayashi, K. Okada, and M. Inaba, "Working with movable obstacles using on-line environment perception reconstruction using active sensing and color range sensor," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2010, pp. 1696–1701.

[19] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. V. Dimarogonas, and D. Kragic, "Dual arm manipulation - A survey," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1340–1353, 2012.

[20] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Simultaneous Grasp and Motion Planning," *IEEE Robotics and Automation Magazine*, vol. 19, pp. 43–57, 2012.

[21] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R. Dillmann, "Humanoid motion planning for dual-arm manipulation and re-grasping tasks," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, oct. 2009, pp. 2464–2470.

[22] R. Shauri and K. Nonami, "Assembly manipulation of small objects by dual-arm manipulator," *Assembly Automation*, vol. 31, pp. 263–274, 2011.

[23] A. Edsinger and C. Kemp, "Two Arms Are Better Than One: A Behavior Based Control System for Assistive Bimanual Manipulation," *Lecture Notes in Control and Information Sciences*, vol. 370, pp. 345–355, 2008.

[24] C. Rosales, L. Ros, J. M. Porta, and R. Suárez, "Synthesizing grasp configurations with specified contact regions," *International Journal of Robotics Research*, vol. 30, no. 4, pp. 431–443, 2011.

[25] F. Gilart and R. Suarez, "Determining Force-Closure Grasps Reachable by a Given Hand," in *10th IFAC Symposium on Robot Control, SYROCO*, September 2012, pp. 235–240.

[26] J. Rosell, A. Pérez, A. Aliakbar, Muhayyuddin, L. Palomo, and N. García, "The kautham project: A teaching and research tool for robot motion planning," in *Proc. of the IEEE Int. Conf. on Emerging Technologies and Factory Automation, ETFA'14*, 2014. [Online]. Available: sir.upc.edu/kautham

[27] C. Rodríguez, A. Montaño, and R. Suárez, "Planning manipulation movements of a dual-arm system considering obstacle removing," *Robotics and Autonomous Systems*, vol. 62, no. 12, pp. 1816 – 1826, 2014.

[28] G. Sanchez and L. J.C., "A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking," in *Int. Symposium on Robotics Research (ISRR)*, 2001, pp. 403–417.

[29] D. Hsu, L. J.C., and R. Motwani, "Path Planning in Expansive Configuration Spaces," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1997.

[30] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2000, pp. 995–1001.

[31] C. Rodriguez, A. Montano, and R. Suarez, "Optimization of robot coordination using temporal synchronization," in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, Sept 2014, pp. 1–7.