Pushing using Compliance

Dennis Nieuwenhuisen

A. Frank van der Stappen

Mark H. Overmars

Department of Information and Computing Sciences, Utrecht University

Technical Report UU-CS-2006-036

www.cs.uu.nl

ISSN: 0924-3275

Pushing using Compliance

Dennis Nieuwenhuisen A. Frank van der Stappen Mark H. Overmars

June 29, 2006

Abstract

This paper addresses the problem of maneuvering an object by pushing it through an environment with obstacles. Instead of only pushing the object through open areas, we also allow it to use compliance, e.g. allowing it to slide along obstacle boundaries. Using compliance has a number of advantages: it extends the number of situations in which a push plan can be found, it allows for simpler (i.e. less complicated) paths in many cases and it often helps solving narrow passage problems. Here, we present an approach based on the Rapidly-exploring Random Tree (RRT). Our approach yields paths through the open space, but also exploits the power of compliance.

1 Introduction

Over the years various techniques have been developed that address the problem of navigating through or interacting with a real or virtual world. These techniques can roughly be divided in two classes, both solving a different type of problem: motion and manipulation planning. Motion planning essentially deals with robots that navigate autonomously while manipulation planning concerns passive objects, that cannot move by themselves, that are manipulated by an autonomous robot.

An example of manipulation is a robot arm in a manufacturing plant that needs to insert a part (a passive object) into an engine. Because of the complex structure of the engine, the goal configuration for the part may be difficult to reach, requiring a very complex and precise motion of the robot arm. Also in virtual environments used for e.g. computer assisted training in which the user must perform a complicated manipulation task, difficulties often occur because of the fine motions required.

A number of causes for the problems that arise while executing such a complex task can be distinguished. Firstly, while the motion required by the object alone may be feasible, the manipulation as a whole may become infeasible because the manipulator itself also needs room to maneuver. Secondly, manipulations may be highly constrained, therefore even small errors in sensor data can lead to failure in the planning process. Finally, finding a manipulation plan in highly constrained environments is often computationally expensive because of the well-known narrow passage problem.

In this paper we propose a novel approach that combines manipulation planning with compliant motions. Here we define compliant motions as motions in which the manipulated object is in contact with the obstacles of the environment. Thus, the object slides along an edge of an obstacle while it is manipulated.

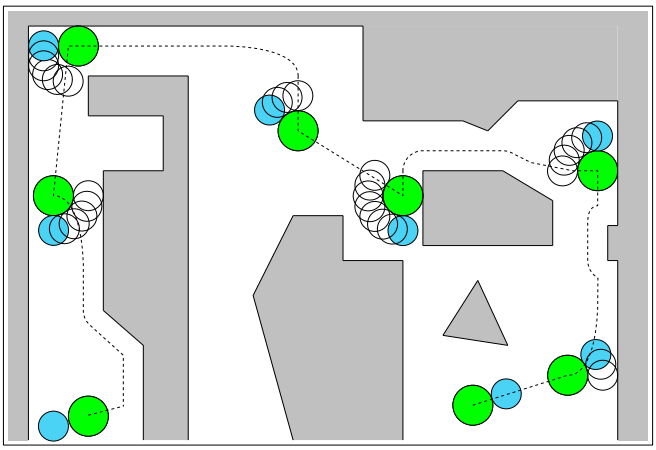

Figure 1: An example created by our implementation. At the left bottom the start configuration is shown. The object is pushed to the goal using a combination of compliant and non-compliant segments. The dotted line represents the path of the object. At some points the motion of the pusher is shown.

Objects can be *manipulated* in numerous ways, and each type of manipulation implies different constraints on the combined motion of manipulator and object. All types of manipulation have in common that the manipulator needs to apply force to the object. Research on how to apply this force has led to a broad range of different forms of manipulation that include grasping [16], squeezing [11], rolling [3] and even throwing [18] an object. However, pushing [17] is one of the most widely studied types of manipulation. The objective is to maneuver an object, incapable of moving by itself, from an initial configuration to a goal configuration by pushing with an autonomous robot (the pusher). Because of the many parameters involved, e.g. mass distribution of the object, different types of friction and limitations of the pusher, pushing is often a difficult problem.

Compliance has often been used to compensate for uncertainty, for example to solve the peg-in-hole problem. In [15] the *pre-image* backchaining approach is introduced to solve robot motion planning problems. The idea is to compute the points from which the goal is reachable with a single manipulation (the pre-image). Next, the pre-image is iteratively treated as a new goal until the initial robot configuration is within a pre-image. Compliance is used to confine the motions to be tangent to the constraining surfaces. In [7] an algorithm introduced in [9] is improved to find a trajectory from a start region to a goal region amidst planar polygonal obstacles where control is subject to uncertainty using compliance.

Our algorithm combines manipulation by pushing with compliance. The solutions are allowed to consists of both compliant and non-compliant sections. In the compliant sections the object touches the boundary of the environment while being pushed by the pusher. The resulting motion of the object is then parallel to the boundary of the obstacles in the environment. An example is shown in Fig. 1.

Allowing compliant segments in the motion plan has several advantages. Firstly, if no compliance is used, there is often only one position on the boundary of the object from which it can be pushed to move in a specific direction. If this position is not reachable by the pusher (because it is obstructed by obstacles as it operates in a highly constrained

environment) no manipulation plan may be found. If the object moves compliantly to the obstacles of the environment there is a range of positions on the boundary of the object that all cause it to move in the same direction parallel to the edges of the environment. Thus it is more likely that the pusher can reach such a position. This may also lead to the additional advantage that the complexity of the manipulation plan is reduced. Secondly, traditional motion planning problems often have difficulties passing narrow passages. These are caused by highly constrained parts of the environment. In these parts of the environment, compliance often helps by passing narrow passages.

A few examples of the advantages of compliance are shown in Fig. 2. The first example, shown in Fig. 2a, shows a situation where no push plan exists without compliance because of the lack of space for the pusher to maneuver. With compliance a simple plan can be created. In the situation of Fig. 2b, a complicated push plan can be created in which the pusher alternates between its current position and the dotted position. With compliance, a much simpler solution exists. Fig. 2c shows a narrow passage. Many manipulation algorithms have problems finding a path through such a passage. Using compliance it is often easy to find a path.

As a first step toward exploring the potentially powerful combination of pushing and compliance and to gain insight in this combination, we look at the simplified problem of two disks. One disk is an autonomous robot (*the pusher*), capable of pushing the other (passive) disk (*the object*). The goal is to push the object from its initial placement to a given goal placement. While being pushed by the pusher, the object is allowed to slide along the boundary of the environment (compliant motion). Even in this reduced problem setting the encountered problems are challenging and their solutions provide useful insights for future research.

Our approach is based on the Rapidly-exploring Random Tree (RRT) [14] algorithm. It combines random exploration of the open space with an exact computation of reachable compliant configurations. For the open areas, a tree is created by generating random configurations. For every random configuration a path is tried from the tree to that random configuration. If the random configuration cannot be reached because of a collision, a configuration as close as possible to the obstacle that caused the collision is added to the tree. This configuration is then used as a starting point to explore the compliant configurations using an exact approach, eventually capturing the structure and connectivity of all compliant configurations. The results of compliant exploration are added as configurations to the tree.

2 Preliminaries

Given disks O and P with radii r_o and $r_p < r_o$ in an environment consisting of k disjoint line segments $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_k\}$, and given a start and goal position for O, we compute a push plan for P such that if P follows this plan, it pushes O from its start to its goal position. In this paper it is assumed that P always keeps contact with O (see the conclusions for some remarks on this). As a consequence the configuration space is three dimensional: two parameters specifying the position $q = (q_x, q_y)$ of the center of O and one that specifies the relative position α of P, the *push position*. Push position α is the angle the line from the center of O to the center of P makes with the positive x-axis. The configuration space is parametrized by the tuple (q, α). A configuration $c = (q, \alpha)$ is said to be *free* if both O and P

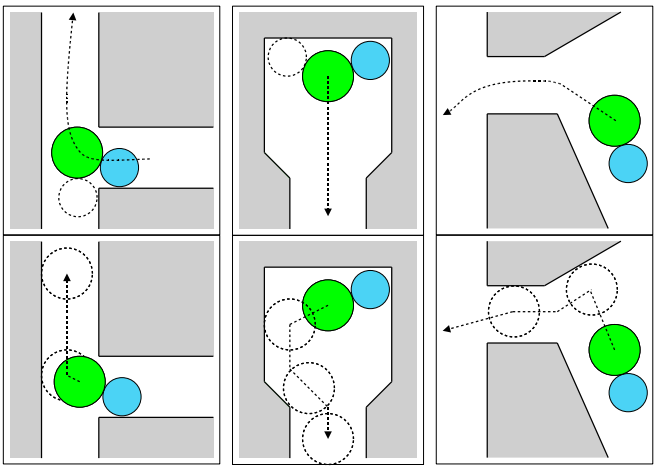

Figure 2: Three examples in which compliance helps finding a push plan for a disk pushing another disk. The top figures show the path toward the goal with a dotted arrow. The bottom figures show the solutions if compliance is allowed.

do not collide with an obstacle. Configuration c is *compliant* with obstacle γ_i if the shortest distance between q and γ_i is r_o. It is assumed that the friction between O and the supporting plane is large enough such that there is no motion of O after pushing ceases (quasi-static assumption). The center of friction of O is the center of the disk.

If the direction of motion of O needs to be changed, the relative position of P needs to be changed while keeping contact with O. Such a transit is called a *contact transit*.

Definition 2.1 (contact transit)**.** *A* contact transit *is a motion in which P slides along the stationary O, thus a contact transit transforms a configuration $c = (q, \alpha)$ into a configuration $c' = (q, \alpha')$.*

Suppose q is not compliant. If O is pushed by P, the path O will follow is on the line through the centers of O and P. If q is compliant however, its behavior when pushed by P will also depend on the obstacle to which it is compliant. Therefore we make a distinction between *compliant space* and non-compliant space. Compliant space is defined as $CS^{r_o} = \partial\Big(\bigcup_{\gamma\in\Gamma}(\gamma \oplus D(r_o))\Big)$, where $D(\rho)$ is a disk of radius ρ and $\oplus$ denotes the Minkowski sum. (The Minkowski sum $A \oplus B$ of two objects A and B is defined as $A \oplus B = \{(a+b)|a \in A \wedge b \in B\}$.) An example of compliant space is shown in Figure 3a.

The *racetrack* (Fig. 3b) of a single obstacle γ_i is defined by $R^{r_o}_{\gamma_i} = \partial(\gamma_i \oplus D(r_o))$. If $q \in R^{r_o}_{\gamma_i}$, then q is called a *compliant position* on obstacle γ_i.

If q is compliant, there is a range of push positions that all cause O to follow the same path in the same direction. Such a range is called the *push range*. The size of this range is dependent on the friction.

Definition 2.2 (push range)**.** *At compliant position $q \in R^{r_o}_{\gamma_i}$, the* push range *describes the continuous range of push positions that cause O to follow the same path (regardless of the push position being collision free). Since there are two directions of motion around γ_i, there are two push ranges:* PR^+ *for the clockwise motion and* PR^- *for the counterclockwise motion.*

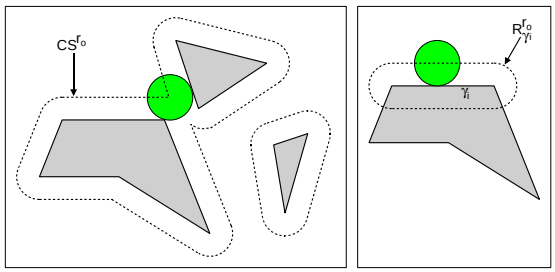

Figure 3: (a) An example of the compliant space CS^{r_o} shown as the dotted lines. (b) The racetrack $R^{r_o}_{\gamma_i}$ of an obstacle γ_i. The position of O can be described by its compliant position.

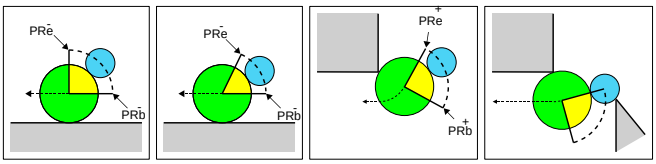

Figure 4: The push range. The path of the object is shown by the dotted arrow. (a) The (counter clockwise) push range if no friction is assumed between the object and the obstacle. (b) The push range if there is friction between O and the obstacle. (c) The (clockwise) push range if O is pushed around an endpoint of an obstacle (no friction). (d) Pushing cannot continue because P is forced outside the push range by an obstacle.

Since the length of the push range is always equal to or smaller than 90 degrees, we can define the two extreme values of PR^+ as starting at PR^+_b and ending at PR^+_e using the shortest counterclockwise rotation. The same holds for PR^-. Using a contact transit, it is not guaranteed that P can reach either $\text{PR}^+_b/\text{PR}^-_b$ or $\text{PR}^+_e/\text{PR}^-_e$ because one or more obstacles can impede the transit. Examples of the push range are shown in Fig. 4.

We will now describe the role of friction on the size of the push range. If P pushes O, it can either slide or roll along the boundary of an obstacle γ_i. This behavior is dependent on two coefficients of friction. μ_1 is the coefficient of friction between O and γ_i and μ_2 is the friction between O and P. For P to stay at the same relative position to O, it needs to push O in the direction of the center of O. The force that P uses to push O is called F. The force orthogonal to F is called F_f. It is easy to see that if the force $F_f \geq F'_f$ (the force due to the friction between O and the obstacle), then O will slide along γ_i, else it will roll. The moment arms of both forces are equal and do not need to be taken into account. The situation is sketched in Fig. 5.

Since $F_f = \mu_2 F$ and $F'_f = \mu_1 F \sin(\beta)$, the following holds:

$$\frac{\mu_2}{\mu_1} \geq \sin(\beta) \tag{1}$$

If Equation 1 holds, O will slide, else it will roll. However, not every value of β results in a motion of O and P. Let F'' be the horizontal component of F. If $F'' \geq F'_f$, O and P will move, else they will stand still. Since $F'' = \cos(\beta)F$, we can derive the following equation:

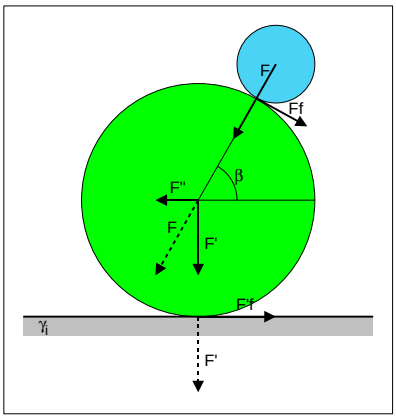

Figure 5: The role of friction on the behavior of O and P. F is the force applied by P on O. μ_1 is the friction between O and γ, μ_2 is the friction between O and P.

$$\tan(\beta) < \frac{1}{\mu_1} \tag{2}$$

If Equation 2 holds, O and P will move, else they will stand still. Stated differently, the size of the push range is dependent on the value of μ_1: $\text{PR}^+/\text{PR}^- = [0, \arctan(\frac{1}{\mu_1})]$. For the rest of this paper we will, without loss of generality, assume that $\mu_1 = 0$ and $\text{PR}^+/\text{PR}^- = [0, \frac{1}{2}\pi]$.

Given a compliant configuration, we define the *bounding obstacles* (Fig. 6).

Definition 2.3 (Bounding obstacles)**.** *Given a compliant configuration $c = (q, \alpha)$, the* bounding obstacles *$BO(q, \alpha)$ are the obstacles (γ_j, γ_k) that P hits first if it rotates from α to either $\text{PR}_b^+/\text{PR}_b^-$ and $\text{PR}_e^+/\text{PR}_e^-$ respectively. The absence of a bounding obstacle, is denoted by $\perp$.*

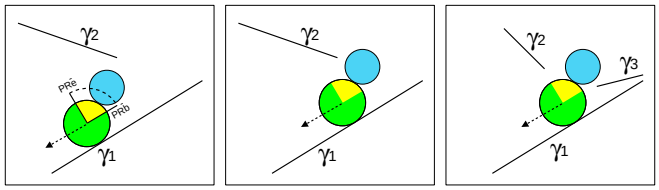

Figure 6: Bounding obstacles. (a) If P rotates to either PR_b^- or PR_e^- it does not encounter any obstacle, therefore the bounding obstacles are $(\perp, \perp)$. (b) P cannot reach PR_e^-, the bounding obstacles are $(\perp, \gamma_2)$. (c) The bounding obstacles are (γ_3, γ_2).

We introduce the notion of *compliant intervals*. A compliant interval is a continuous subset of the racetrack of an obstacle in which the bounding obstacles are fixed.

Definition 2.4 (Compliant interval)**.** *A compliant interval $I_{\gamma_i}(\gamma_j, \gamma_k) = \{q \in R^{r_o}_{\gamma_i} | \exists \alpha : BO(q, \alpha) = (\gamma_j, \gamma_k)\}$ defines a (continuous) subset of $R^{r_o}_{\gamma_i}$ in which the bounding obstacles are fixed. The compliant start position of I_{γ_i} is denoted by $q_b(I_{\gamma_i})$, its end position by $q_e(I_{\gamma_i})$.*

Because of the definition of bounding obstacles, an interval is directed, i.e. both the clockwise and counterclockwise paths on the racetrack have their own unique set of intervals.

At one compliant position of an obstacle, multiple (overlapping) intervals can be defined, an example is shown in Fig. 7. A compliant interval is unique, i.e. an interval with the same bounding obstacles cannot occur twice on a racetrack of an obstacle. Exceptions are the compliant configurations where no obstacles bound PR^+/PR^-(as is the case in Fig. 7b). But since these cannot overlap, they can still be uniquely defined. The following lemma follows directly from definitions 2.3 and 2.4.

Lemma 2.1 (Reaching the end of an interval)**.** *Given a compliant configuration* $c = (q, \alpha)$: $q \in I_{\gamma_i}(\gamma_j, \gamma_k) \wedge BO(q, \alpha) = (\gamma_j, \gamma_k)$*, it is assured that* P *can push* O *compliantly to* $q_e(I_{\gamma_i})$*.*

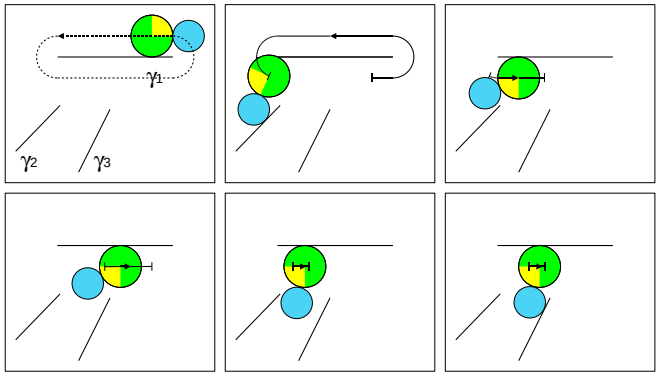

Figure 7: The compliant intervals of the counterclockwise path on the racetrack of obstacle γ_1. (a) Pushing O counterclockwise around γ_1, every compliant position can be reached. The counterclockwise compliant positions are divided in 5 intervals. (b)...(f) The intervals of γ_1. The respective bounding obstacles are: $(\perp, \perp)$, $(\perp, \gamma_2)$, $(\perp, \gamma_3)$, $(\gamma_2, \perp)$ and (γ_2, γ_3). Note that the intervals of (c), (e) and (f) overlap.

Since our final path will consist of both compliant and non-compliant sections, we need a way to connect a compliant configuration to a non-compliant configuration. To leave a compliant position on obstacle γ_i efficiently (i.e. limiting the length of the path), a contact transit is used to revolve P to a position as close to γ_i as possible. If P then follows a path parallel to γ_i, it pushes O away from γ_i. If no other obstacles impede the path, the resulting motion of the object is a curve called a *hockey stick curve* (see Fig. 8). A hockey stick curve minimizes the distance needed to push O away from an obstacle.

In [1] a mathematical description of a hockey stick curve is given for a point pushing a disk. The resulting motion of a pusher of radius r_p pushing an object of radius r_o is equivalent to that of a point pushing a disk of radius $r_p + r_o$ where the point represents the center of P. The coordinates of O as a function of time are shown as Equation 3, β is the initial angle the line through the centers of O and P makes with the obstacle.

$$
\begin{aligned}
x(t) &= t + \frac{1 - \tan^2\left(\frac{\beta}{2}\right) \cdot e^{2t}}{1 + \tan^2\left(\frac{\beta}{2}\right) \cdot e^{2t}} - \cos(\beta) \\
y(t) &= \frac{2\tan\left(\frac{\beta}{2}\right) \cdot e^{t}}{1 + \tan^2\left(\frac{\beta}{2}\right) \cdot e^{2t}} - \sin(\beta)
\end{aligned}
\tag{3}
$$

To push O from a compliant configuration to a non-compliant configuration, the shortest path is to create a hockey stick curve, followed by a straight line push. We will refer to a combination of a hockey stick curve and a straight line as a *hockey stick push*. An example of a hockey stick push is shown in Fig. 8d.

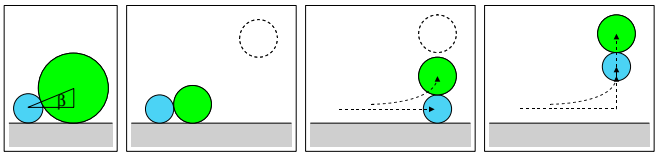

Figure 8: (a) β is the initial angle between O and P. (b) The start situation, the target of O is shown as the dotted disk. (c) A hockey stick curve. The dotted lines show the paths of O and P. (d) A hockey stick push. First a hockey stick curve is used, followed by a straight line push. The resulting path is the shortest path to the goal from the start configuration.

3 Rapidly-exploring Random Trees

Many motion planning algorithms are based on the generation of collision-free samples. Between these, connections are tried and a graph is formed that can be used to solve motion planning queries. Since random sampling does not result in compliant samples, we need a method to create them. Retracting non-compliant samples to the obstacles [2] seems a straightforward solution. However this introduces the problem of deciding when and how samples need to be retracted. Also, retraction of samples tends to be a costly operation.

After a random sample has been created, most algorithms try to connect it to already existing samples in the graph. If the connection fails (because of a collision), an obstacle must be impeding the path. Usually such connections are discarded. In our case however, these collisions are valuable in creating compliant samples. Since we aim at creating a path from a start to a goal configuration (single shot approach), we use a bidirectional version of the Rapidly-exploring Random Trees (RRT) [14] to create the necessary graph. We will first briefly explain how the basic RRT algorithm works and then elaborate on how to adapt the RRT such that it is suited to solve our problem.

3.1 The basic RRT algorithm

The RRT is a single shot approach in which a tree is constructed that gradually improves resolution. Here, we will use the bidirectional version of the RRT that grows two exploration trees: T_s from a start configuration c_s and T_g from a goal configuration c_g. As the trees grow larger, the two trees are more likely to connect. If a connection is established, a path is found. The advantage of the bidirectional version over the single directional version is that it is better at escaping local minima. The algorithm described in this paper is also suited for the approach of using more than two trees, for more information see [13].

Initially the start configuration c_s is added as a vertex to T_s and the goal configuration c_g is added to T_g. Next, a random configuration c_r is generated. The nearest configuration $c_n \in T_s$ to c_r is found (using a Euclidean distance metric). c_n can either be associated with a vertex in T_s or it can be a configuration in the interior of an edge in T_s. In either case, a *local planner* is used that tries to create a path from c_n to c_r. If on this path an obstacle is

encountered, the closest configuration c_c to the boundary of the obstacle is determined. If no obstacle is encountered and thus c_r is reached, then $c_c = c_r$. Next, c_c is added to T_s as a vertex, together with the edge (c_n, c_c). In order to grow T_g toward T_s, an attempt is made to connect c_c to the nearest configuration c'_n in T_g. Again, if the local planner encounters an obstacle on the path between c'_n and c_c, we connect c'_n to the closest reachable configuration c'_c on the boundary of the obstacle and add c'_c to T_g as a vertex together with the edge (c'_n, c'_c). If $c'_c = c_c$, the two trees are connected and a solution is found.

3.2 Tailoring the RRT

The basic RRT algorithm is not suited for pushing problems since it does not incorporate the role of P. We need a special version of the local planner that takes this role into account (Section 4). In the algorithm of the RRT, we only generate random positions for O, the positions for P is determined by the local planner (Fig. 9a).

Using the RRT as a basic planning algorithm generates compliant configurations in a natural way. If the local planner encounters an obstacle when connecting two configurations, the point of collision is a compliant configuration. The more obstacles are present between the start configuration c_s and the goal configuration c_g, the more compliant configurations will be found by the RRT. The result of this process is shown in Fig. 9b.

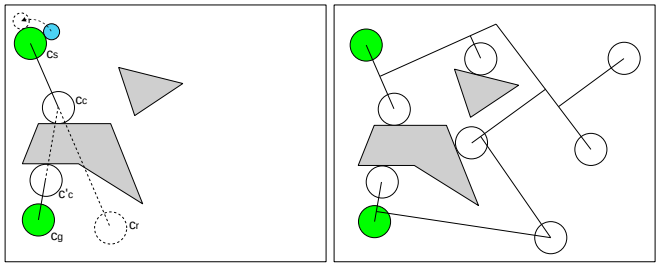

Figure 9: Example of the creation of an RRT. (a) The first iteration, the position of P is determined by the direction of the path from c_s to c_c. (b) The final RRT. As can be seen, a couple of compliant configurations have been found.

To handle compliance, the RRT algorithm needs to be extended. If a compliant configuration has been found we will need to discover which part of the compliant space is reachable from this configuration. This procedure is called *compliant exploration* (Section 5). Every obstacle has two distinct compliant exploration directions: one in the clockwise and one in the counter clockwise direction.

During compliant exploration, O is pushed around the obstacle until it returns at the start position or until no further pushing is possible. If obstacles belong to the same *compliant component*, a path through compliant space may exist between them. A compliant component is a connected component in compliant space (see Fig. 10a for an example). The fact that two obstacles belong to the same compliant component by no means guarantees that a path through compliant space can be created (Fig. 10b). This also depends on the size and configuration of the obstacles and the size of P.

Because of the physics of pushing, pushing motions are irreversible. As a result, the connections between vertices of the RRT are directed and the paths the local planner creates are only suited for motions in one direction. Reversing the direction of collision checking (e.g.

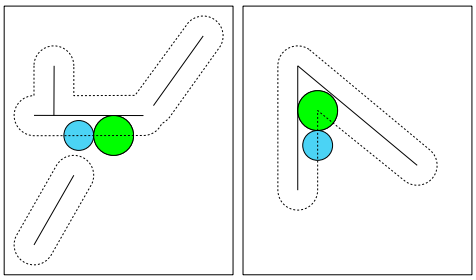

Figure 10: (a) A scene consisting of 4 obstacles. As can be seen from the compliant space CS^{r_o} (the dotted lines), the top 3 obstacles form a compliant component while the bottom obstacle is a compliant component on its own. (b) Even though the two obstacles form a compliant component, the object runs into a dead-end.

reversing the endpoints) for connections to T_g is no solution because if an obstacle is encountered, no connection to T_g can be created, which undermines the proper working of the RRT. The solution is to use a "reversed" version of the local planner, including a reversed version of compliant exploration. The complete RRT algorithm is shown in pseudo-code as Algorithm 1.

Algorithm 1 PUSHINGRRT(S, G, c_s, c_g)

```
1:  T_s.ADDVERTEX (c_s)
2:  T_s.ADDVERTEX (c_g)
3:  repeat
4:    q_r ← random position for O
5:    c_n ← GETNEARESTNEIGHBOR(T_s, q_r)
6:    c_c = (q_c, α_c) ← LOCALPLANNERFORWARD(T_s, c_n, q_r)
7:    if c_c ≠ NULL {did we find a valid vertex?} then
8:      T_s.ADDVERTEX (c_c)
9:      T_s.ADDEDGE (c_n, c_c)
10:     c'_n ← GETNEARESTNEIGHBOR(T_g, q_c)
11:     c'_c ← LOCALPLANNERREVERSE(T_g, c'_n, q_c)
12:     if c'_c ≠ NULL {did we find a valid vertex?} then
13:       T_g.ADDVERTEX (c'_c)
14:       T_g.ADDEDGE (c'_n, c'_c)
15:     if PATHEXISTS (c_s, c_g) then
16:       RETURN FOUND
17: until stopping criterion is met
```

In lines 5 and 10 the nearest configuration in the tree needs to be found. If the nearest configuration (c_n) in the tree is non-compliant, the shortest distance is simply the straight line path. If a compliant configuration is involved, the shortest distance between two configurations may however consist of a (partial) hockey stick curve.

4 Local Planner

The local planner (lines 6 and 11 of Algorithm 1) is a crucial part of many motion planning algorithms. It connects two configurations to each other, usually by trying a straight line path between them. If the local planner succeeds in finding a path, a connection in the tree between the corresponding vertices is created.

As our configurations consist of a position of O along with a relative orientation for P, and configurations may or may not be compliant, the local planner needs to adapt its approach according to the local situation. Also we need two versions of the local planner: a "forward" one (Algorithm 2) and a "reversed" one (Algorithm 3); both will be discussed.

4.1 Forward local planner

We will consider the various types of local paths necessary to create the local planner. Suppose the nearest neighbor configuration c_n resulting from line 5 of Algorithm 1 is non-compliant (Fig. 11a). The local planner has to verify whether a straight line path exists that connects the nearest neighbor configuration $c_n = (q_n, \alpha_n)$ to the random position q_r. There is only one push position for P such that O follows a straight line path from q_n to q_r. To reach this push position, a contact transit is used at q_n.

If c_n is compliant to obstacle γ_i (Fig. 11b) it depends on the position of q_r whether P is able to transit to the desired push position for a straight line path. It is likely that this will fail because γ_i will probably impede the contact transit. Thus, we try to push O a distance of $2r_p$ away from γ_i. If this succeeds, we are certain that P now fits between O and γ_i. The most effective way of pushing O away from γ_i is to use a hockey stick push. A hockey stick push maximizes the angle in which P pushes O away from γ_i. Before the end of the hockey stick is reached, P may encounter another obstacle. If this happens, a new hockey stick push is started etc. (Fig. 12). In theory, if O is in a small confined area or if the radii of O and P are (almost) equal, this can continue infinitely. Restricting the total length of the consecutive hockey stick curves prevents that this process does not terminate.

In both situations (c_n being compliant or non-compliant) P has reached the desired push position to try to push O in a straight line from its current position to q_r. If q_r is reached, then a (non-compliant) vertex positioned at q_r is added to T_s together with an edge from c_n to the new vertex. The position of P at the new vertex is set to the position of P at the end of the push. If an obstacle is encountered at c_c before q_r is reached (Fig. 11c), a compliant configuration is found and this is used as a starting point for the compliant exploration algorithm of Section 5. The complete LocalPlannerForward algorithm is shown as Algorithm 2.

4.2 Reverse local planner

In line 11 of Algorithm 1 a connection is tried from configuration $c'_n = (q'_n, \alpha'_n)$ in T_g to q_c in T_s. As explained in Section 3, if a connection is made to the goal tree T_g, we need a different local planner because edges need to be directed *to* that tree as opposed to *away* from it in the previous section. A straightforward solution would seem to use the same local planner and just reverse the endpoints (starting at q_c and moving to c'_n). If however an obstacle blocks the path, no connection to T_g is found and we only extend T_s which undermines the strength of the bidirectional approach (e.g. escaping narrow passages) and makes the algorithm unsuited

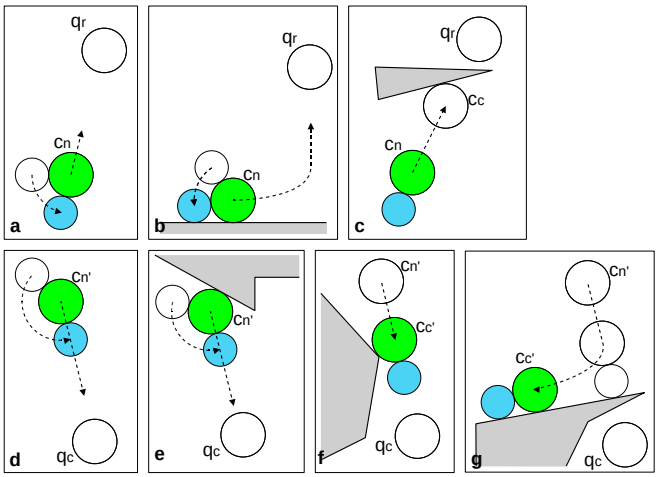

Figure 11: The different situations the local planner has to deal with, (a..c) are the situations of the forward local planner, (d..g) are the situations of the reversed local planner. Note that in the reversed local planner the edges in the tree are directed opposite to the dotted arrows that show the direction of exploration.

Algorithm 2 LocalPlannerForward(T_s, c_n, q_r)

1: **if** Compliant(c_n) **then**
2: $c_n \leftarrow$ CreateHockeyStick(c_n)
3: **if** c_n = NULL **then**
4: return NULL {failure}
5: $c_n \leftarrow$ CreateContactTransit(c_n, q_r)
6: **if** c_n = NULL **then**
7: return NULL {failure}
8: $c_n \leftarrow$ Push(c_n, q_r) {push O from c_n to q_r}
9: **if** Compliant(c_n) **then**
10: ExploreCompliantCW(T_s, c_n)
11: ExploreCompliantCCW(T_s, c_n)
12: return c_n
13: **else**
14: return c_n {q_r reached by Push}

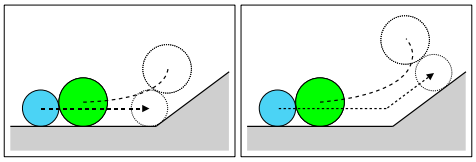

Figure 12: (a) During the hockey stick push, P encounters an obstacle. (b) A second hockey stick is used.

Algorithm 3 LocalPlannerReverse(T_g, c'_n, q_c)

```
1: c'_n ← CreateContactTransit(c'_n, q_c)
2: if c'_n = NULL then
3:    return NULL {failure}
4: c'_n ← ReversePush(c'_n, q_c)
5: if PusherCollide(c'_n) {did P hit an obstacle during the reverse push?} then
6:    c'_n ← ReverseHockeyStick(c'_n)
7:    if c'_n = NULL then
8:       return c'_n {failure}
9: if Compliant(c'_n) then
10:   ReverseExploreCompliantCW(T_g, c'_n)
11:   ReverseExploreCompliantCCW(T_g, c'_n)
12:   return c'_n
13: else
14:   return c'_n {q_c reached by ReversePush}
```

for an approach that uses multiple trees.

Because of the above considerations, we need a true reverse version of the local planner. Because the edges are directed to T_g, P is revolved 180 degrees (i.e. P precedes O during planning). Therefore it makes no difference whether c'_n is non-compliant (Fig. 11d) or compliant (Fig. 11e). In both situations we start by verifying whether P can reach the desired push position using a contact transit.

Next, the path is checked for collision. Three situations can occur. First, T_s and T_g can become connected. In that case, the algorithm ends successfully. Second, a collision of O with an obstacle can occur (Fig. 11f) and a new compliant configuration c'_c is found. The third situation that can occur is that P collides with obstacle γ_i (Fig. 11g). A "reverse" hockey stick curve (or multiple curves as in Fig. 12) is used to see if a path to a compliant configuration exists. If this succeeds, again a new compliant configuration c'_c is found.

If c'_c is compliant, the reverse exploration algorithm needs to be executed. Because the edges of T_g need to be directed to T_g we cannot use the same exploration algorithm but rather need a reversed version that is explained in Section 5. The complete reverse local planner algorithm is shown as Algorithm 3.

4.3 Geometric primitives

The task of the local planner is to verify if the random position q_r can be reached from the nearest configuration $c_n = (q_n, \alpha_n)$ in the tree or else report the first obstacle with which there will be a collision. To solve this, we will transfer this problem to basic geometric problems.

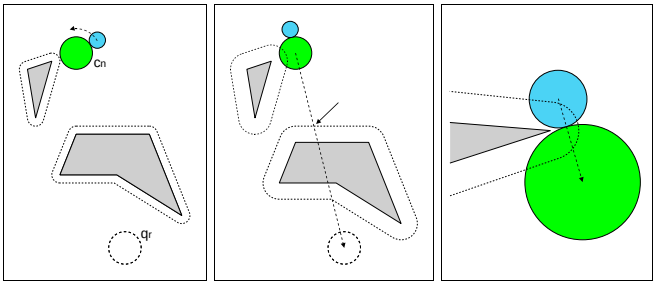

Figure 13: Ray shooting in the contact spaces. The rays are the dotted arrows. (a) A circular ray shoot in CS^{r_p} to collision check the contact transit. (b) A linear ray shoot in CS^{r_o} to collision check the path of O. The arrow shows the point of collision. (c) A linear ray shoot in CS^{r_p} to collision check the wedges.

The situation for the forward local planner is described, the situation for the reverse local planner is similar.

Before O can be pushed from q_n to q_r, P needs to reach the desired push position by means of a contact transit. To check the validity of the contact transit, we construct, $CS^{r_p} = \partial\Big(\bigcup_{\gamma\in\Gamma}(\gamma \oplus D(r_p))\Big)$. An example of CS^{r_p} is shown in Fig. 13a. In this environment, P is a point object. We will use ray shooting to check the validity of a contact transit. Ray shooting considers the problem of determining the first intersection between a ray (a directed line segment) and a collection of obstacles. The line segment can be either a straight line or a circular arc. We transform our problem to a circular ray shooting problem (the path of the center of P, having radius $r_o + r_p$) in an environment consisting of circular arcs and line segments (the features of CS^{r_p}). If the ray intersects CS^{r_p} before the desired push position is reached, then no contact transit is possible. Note that both the clockwise and counterclockwise contact transits for P need to be checked.

If P succeeds in reaching the desired push position, it can start pushing O in the direction of q_r. To check the validity of the path of O or to find the first obstacle O collides with, linear ray shooting is used from q_n to q_r in the environment of CS^{r_o} (Fig. 13b). If a collision occurs, then we have found a compliant configuration c_s. Note that on the path from q_n to q_r, P moves in the shadow region of O and thus does not need to be collision checked. An exception occurs when after the contact transit an obstacle is present in one of the "wedges" between O and P. This can be corrected by a linear ray shoot in CS^{r_p} from the center of P to the center of O (Fig. 13c).

Efficient algorithms to perform ray shooting can be found in [8,12]. To check a hockey stick curve for collision for both P and O requires numerical analysis. This can be implemented using techniques from e.g. [6].

In lines 5 and 10 of Algorithm 1 we need to find the configuration in the tree nearest to q_r. This nearest configuration is not necessarily a vertex of the tree, but can also be a position on an edge. Because it is not essential to find the nearest neighbor configuration exactly and because our edges do not solely consist of straight lines (but also hockey stick curves), we can use an approximate solution. Every edge in the tree is approximated by adding intermediate configurations along the edge that are used only for nearest neighbor searching [13].

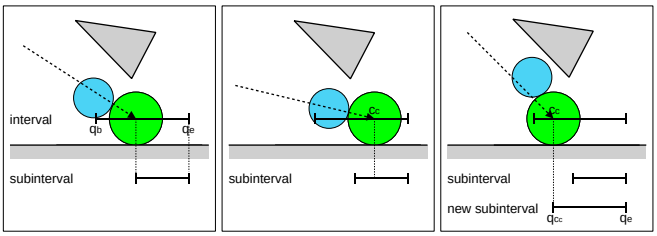

Figure 14: The subintervals are associated to a vertex. (a) The dotted arrow shows how O arrived on the interval. From that position, P is guaranteed to be able to push O to the end of the interval (q_e), hence the subinterval that is associated to a vertex v. (b) If, at a later moment, O arrives in an already explored part of the interval, no new vertex needs to be added to T_s. (c) If O arrives in a not previously explored part of the interval, the subinterval is extended.

5 Compliant exploration

If the local planner is not able to reach the random configuration q_r but instead hits an obstacle γ_i at configuration c_c, this collision point is used as a starting point for compliant exploration. Compliant exploration is a procedure to capture the topological structure of the set of compliant configurations that can be reached starting at c_c, i.e. the part of compliant space that can be reached. The results of this exploration are compliant configurations that are added to T_s and T_g. Starting at c_c, we initiate the compliant exploration in two directions: clockwise and counterclockwise; for one of these a preceding contact transit is necessary. After that, the two are similar.

To capture the topology of compliant space and to distinguish between explored and not yet explored parts of compliant space, intervals are used (see Definition 2.4). A vertex v, compliant to obstacle γ_i stores the compliant position $q_{c_c} \in R^{r_o}_{\gamma_i}$ of O along with the interval I_{γ_i}. The interval implicitly defines the position of P. Because of Lemma 2.1 we know that, once a configuration that belongs to an interval has been reached, P is guaranteed to be able to push O to the end of that interval. Therefore, v does not represent a single position, but rather the continuous subset $[q_{c_c}, q_e(I_{\gamma_i})]$ of I_{γ_i}. This subset is denoted by the *subinterval* of v. An example of a subinterval is shown in Fig. 14a. If v is used in a query, P is free to choose a position between the bounding obstacles of the associated interval I_{γ_i}.

5.1 Forward compliant exploration

If the RRT algorithm generates a compliant configuration $c_c = (q_{c_c}, \alpha_{c_c})$ on obstacle γ_i, the corresponding interval I_{γ_i} is identified by determining the bounding obstacles. Now three cases can occur. The first case occurs if I_{γ_i} has not been encountered before. A new vertex v is created and added to T_s. The subinterval associated to v is $[q_{c_c}, q_e(I_{\gamma_i})]$. This case is shown in Fig. 14a. In the second case, there is already a vertex v' in T_s associated with I_{γ_i}, and q_{c_c} is inside the subinterval of v'. All compliant positions reachable from c_c were already reachable from v', thus c_c can be discarded and no new vertex is added to T_s (Fig. 14b). The third case occurs when there is already a vertex v' in T_s associated with I_{γ_i}, but q_{c_c} is outside the subinterval of v' (Fig. 14c). A new vertex v is added to T_s that is a copy of v' but with an associated subinterval $[q_{c_c}, q_e(I_{\gamma_i})]$. Vertex v' is now redundant and is removed from T_s.

If a new vertex v has been added to T_s, we check whether, at position $q_e(I_{\gamma_i})$ the next interval can be reached (possibly after a contact transit). Since P and O always maintain contact, there is at most one such interval and it will be associated to the same compliant component. If another interval can be reached, the procedure is repeated.

There are a number of situations in which compliant exploration ends. As stated before, we can encounter an already explored part of an interval. In that case, compliant exploration ends. Also, at the start of a new interval, the contact transit to a position in the push range may fail (Fig. 15a). If P is not able to push O any further because it gets stuck between two obstacles or is forced outside the push range, exploration ends (Fig. 15b). If O reaches the position where exploration started (e.g. it was pushed entirely around a compliant component) exploration also ends (Fig. 15c).

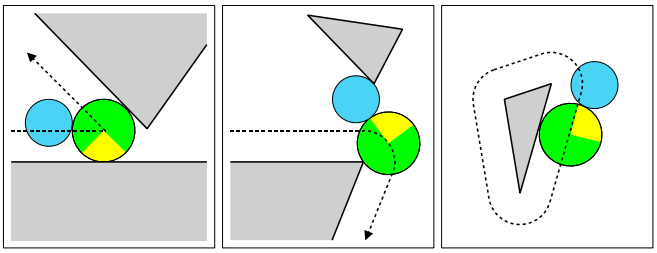

Figure 15: Compliant exploration ends. (a) O hits another obstacle and P is not able to reach the new push range. (b) P is forced outside the push range by an obstacle. (c) O has been pushed completely around a compliant component.

5.2 Reverse compliant exploration

As stated before, if a connection is made from T_g, we need a reverse version of the compliant exploration algorithm because all edges need to be directed to the goal. Suppose that compliant configuration c'_c (line 11 of Algorithm 1) is connected to T_g. Then c'_c is the starting point for reverse compliant exploration. Since the edges in T_g are directed to the goal position, the paths that result from compliant exploration need to be directed to c'_c. Stated differently, we want to know from which part of compliant space c'_c is reachable. Since reverse exploration is used, the reverse exploration direction is counter clockwise for the paths that are directed clockwise (and vice versa). Again P is revolved 180 degrees.

Reverse exploration in interval I_{γ_i} starts at some compliant position $q_{c'_c} \in R^{r_o}_{\gamma_i}$ and is directed to $q_b(I_{\gamma_i})$. Again, because of Lemma 2.1 we are guaranteed to be able to reach $q_b(I_{\gamma_i})$. In contrast to forward exploration, using reverse exploration, two intervals may be reachable from I_{γ_i}. Both have to be explored and corresponding vertices and edges have to be created and added to T_g. Fig. 16a shows such a situation. Object O has reached the start of an interval at $q_b(I_{\gamma_i})$ and P has the choice between two directions around γ_j, both are shown in Figs. 16b+c.

5.3 Geometric Aspects

Compliant exploration is a purely geometric process. After a compliant configuration c_c has been found by the local planner, we first need to detect to which interval c_c belongs. An interval is identified by the bounding obstacles. To find the bounding obstacles at c_c, circular

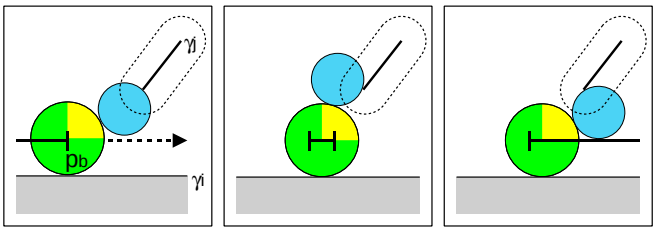

Figure 16: Reverse exploration in the direction of the arrow. (a) At this point P has the choice between two intervals (either going clockwise or counterclockwise around γ_j). (b) If P chooses the counterclockwise direction, a small interval is reached. (c) If P chooses the path through the corridor between the two obstacles, a larger interval is reached.

ray shooting in CS^{r_p} can be used. These rays represent the center of P. One ray is shot from the center of the current position of P in the direction of $\text{PR}_b^+/\text{PR}_b^-$ and another one in the direction of $\text{PR}_e^+/\text{PR}_e^-$. Recall that, if P is outside the push range, a preceding contact transit may be necessary.

After determining the interval I_{γ_i} to which c_c belongs, we are certain to be able to reach $q_e(I_{\gamma_i})$. To find $q_e(I_{\gamma_i})$, a number of *events* can be distinguished that trigger the end of an interval. To determine these events, we use 6 structures (shown in Fig. 17):

- CS^{r_p}
- CS^{r_o}
- $CS^{2r_p+r_o}$
- $CS^{2r_o+r_p}$
- $LPCW$: the path of the lowest endpoint of the push range when moving clockwisely around a compliant component
- $LPCCW$: the path of the lowest endpoint of the push range when moving counter clockwisely around a compliant component

We will now describe the events for compliant exploration in the clockwise direction using the structures. Compliant exploration in the counter clockwise direction is analogous.

1. Since an interval is associated with an obstacle, the interval ends by definition when O collides with a different obstacle. The collisions of O with another obstacle are easily determined by checking mutual intersections of CS^{r_o} (Fig. 18a).
2. There is a change in the bounding obstacles. We subdivide the events that cause such a change in 3 different types.
   (a) If an obstacle enters the push range, an event occurs. An obstacle can enter the push range in two ways, either via PR_b^+ or PR_e^+. The first can be found by calculating intersections between $CS^{2r_o+r_p}$ and CS^{r_p}. Such an intersection is shown in Fig. 18b. The second, shown in Fig. 18c can be found by finding the intersections between $LPCW$ and CS^{r_p}.

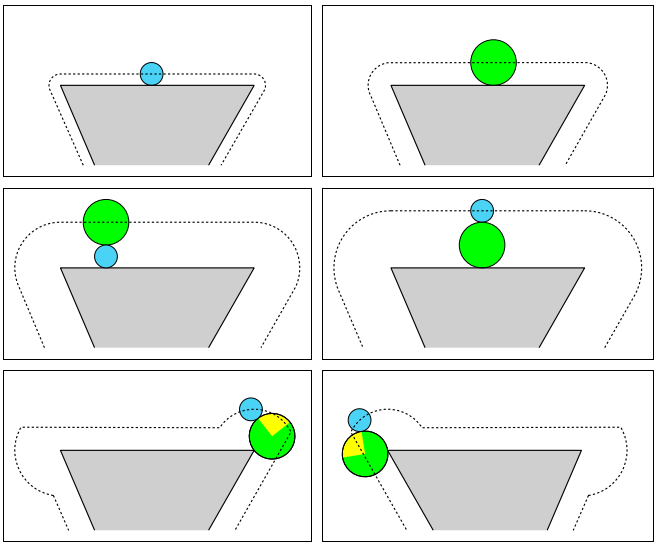

Figure 17: The various structures necessary to determine the events. All consist of circular arcs and straight lines. (a) CS^{r_p}. (b) CS^{r_o}. (c) $CS^{2r_p+r_o}$. (d) $CS^{2r_o+r_p}$. (e) The path of the lowest position of the push range clockwise $LPCW$. (f) The path of the lowest position of the push range counterclockwise $LPCCW$.

(b) An obstacle can also leave the push range. An obstacle γ_j leaves the push range if its distance to the center of O is such that P fits between O and γ_j. This event can be determined by finding intersections between CS^{r_o} and $CS^{2r_p+r_o}$ (Fig. 18d).

(c) If O moves without an obstacle entering or leaving the push range, an event can also occur. These can also be determined using the intersections of 2a. In Fig. 18e an obstacle forces P outside the push range and an interval ends. Fig. 18f shows an example of the start of a new interval. Since P is able to reach PR_b^+, a new interval originates.

Using the events in the order in which they are encountered, it is easy to determine the chain of intervals that are reachable from the compliant start position in both the forward and reverse exploration direction. If O can be pushed completely around a single obstacle without any event occurring, an interval also ends. Note that an event denotes the end of *an* interval, not necessarily I_{γ_i}. Storing the end points of the elements of the structures in Kd-trees [5] allows for efficient determination of the events.

To find the events in case the friction μ_1 between O and Γ is larger than 0, only $CS^{2r_o+r_p}$ needs to be adapted analogously. Instead of determining the intervals after a compliant configuration has been found, preprocessing the events and intervals and storing them allows for even faster compliant exploration.

6 Probabilistic completeness

The basic RRT algorithm (that only grows a tree from the start configuration) is known to be probabilistically complete [14]. The advantage of the bidirectional version of the RRT is that

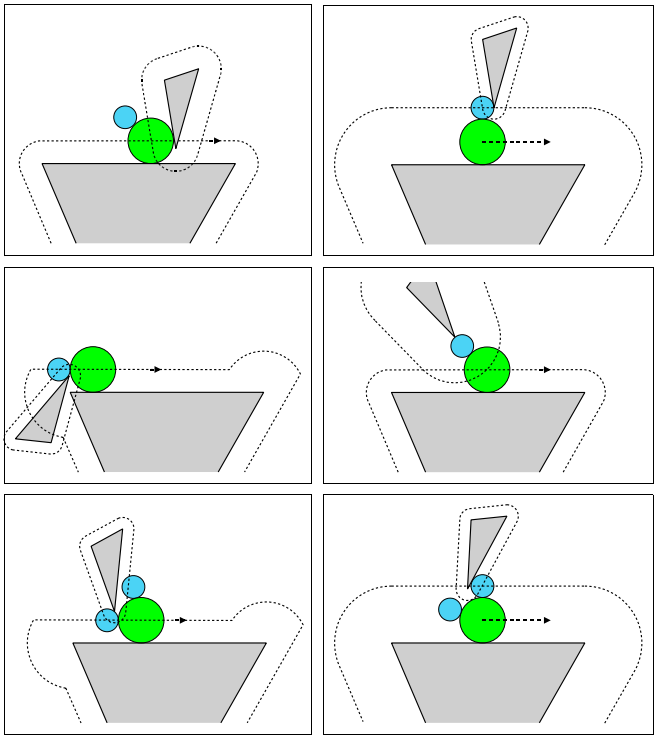

Figure 18: The various events that trigger the start and end of intervals.

it helps escaping narrow passages. If the single-tree-RRT is used for pushing through non-compliant space, the position of P at a vertex is dictated by the direction of the connection of the vertex to the tree. Because of the random nature of the RRT every pushable path through non compliant space will eventually be found.

The above also means that every possible compliant configuration reachable from non compliant space will eventually be found, because a compliant section in a path is always preceded by a non-compliant section. Also since eventually every vertex will be generated, every hockey stick curve will be considered, and thus all possibilities to leave a compliant section will be tried.

7 Experiments

We implemented our algorithm in Visual C++. Obstacles were preprocessed to create the list of events, since these are equal for every query. In this preprocessing phase, we first created the various structures (CS^{r_o}, CS^{r_p} etc.). All these structures consist of circular arcs and straight lines. Their intersections were used to create the list of events associated with every obstacle, as described in Section 5. We did not use a Kd-tree to store the intersections, but rather identified them in a brute force manner. Using the events, the list of intervals for every obstacle was created. During the execution of a query, the list of intervals was used for the compliant exploration procedure if a compliant configuration was found. Hockey stick curves were collision checked by taking small steps along the curve.

We conducted several experiments on a Pentium 2.4GHz system with 1GB of memory. The results were compared with a standard RRT algorithm, that does not use compliance. This algorithm simply follows the description given in Section 3.1, with the addition that contact transits were incorporated to ensure that connections were valid. The results are shown in Table 1. All results were averaged over 10 runs.

First, we looked at the (simple) examples of Fig. 2. As explained before, without compliance no path can be found in Fig. 2a. With compliance a solution can easily be found. After preprocessing (which takes about 0.02s) queries can be executed quickly in 0.0025s on average.

In Fig. 2b, even after the creation of hundreds of configurations, the basic RRT algorithm could not find a valid path. This is due to the fact that a very specific sequence of random samples is necessary. With compliance, it is easy to reach the goal once a compliant configuration has been found. Preprocessing takes about 0.007s after which queries can be executed in 0.004s on average.

In the experiment with the environment shown in Fig. 2c, the basic RRT algorithm is able to find a path. However because the object has to pass a narrow passage, it takes more time to reach the destination then when using compliance. After preprocessing which takes about 0.01s, our algorithm is able to find a path in 0.0005s on average as opposed to 0.01s for the RRT algorithm. Also the RRT needs many more vertices to find a solution 14 versus 141.

Finally, we conducted experiments with the scene shown in Fig. 19 consisting of 19 obstacle line segments. Preprocessing took 0.07s and queries were executed in 0.02s on average. Using only an RRT, a query took 0.27s. Notice also the difference in number of vertices: 115 for the compliant algorithm versus 506 for the RRT algorithm.

Scene	Type	Prep. time	Avg. query time	Avg. # vertices
Fig. 2a	Compliance	0.02s	0.0025s	40
Fig. 2b	Compliance	0.007s	0.004s	30
Fig. 2c	Compliance	0.01s	0.0005s	14
Fig. 19	Compliance	0.07s	0.02s	115
Fig. 2c	RRT only	-	0.01s	141
Fig. 19	RRT only	-	0.27s	506

Table 1: Results of the experiments.

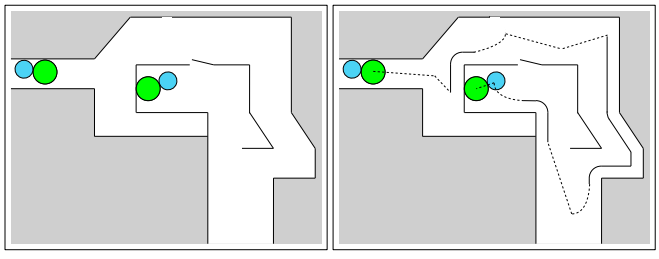

Figure 19: (a) Experimentation scene. At the left the start configuration is shown, in the center the goal configuration. (b) An example of a final path, the dotted lines represent non-compliant path segments, the solid lines represent compliant path segments.

The number of mutual intersections of the various structures increases quadratically with

the number of obstacles in worst case. However, in practical environments we encountered a linear increase in the number of intersections and thus, if a Kd-tree is used, a linear increase in preprocessing time.

8 Conclusions

In this paper we have introduced a novel manipulation planning algorithm in which pushing is combined with compliant motions. We used the reduced problem of two disks as a first step to explore the potential of this combination. The resulting manipulation plans use compliance to extend the range of problems for which a solution can be created. Also in environments or in subsets of environments in which the density of the obstacles is high, compliance helps in lowering the complexity of the solution and to pass narrow passages.

We used the RRT algorithm to provide a natural balance between the number of compliant and non-compliant path segments. Since our algorithm supports both forward and reverse edges, it is also suited for a multi-tree approach [19]. If the environment is preprocessed then, given a compliant configuration, it is easy to check to which interval that configuration belongs and thus what part of the compliant space is reachable from it using geometric operations.

We have not allowed the pusher to lose contact with the object. Extending our algorithm by allowing these non-contact transits can be done by using existing motion planning techniques in which the pusher moves and the object is considered an obstacle. Configurations in which a contact transit is not possible (because of collision of the pusher), are candidates for such non-contact transits.

Acknowledgements

Part of this research has been funded by the Dutch BSIK/BRICKS project. The authors would like to thank Arno Kamphuis for fruitful discussions and suggestions.

References

[1] P. K. Agarwal, J.-C. Latombe, R. Motwani and P. Raghavan, Nonholonomic Path Planning for Pushing a Disk Among Obstacles, *Proc. IEEE Int. Conference on Robotics and Automation,* 1997

[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones and D. Vallejo, OBPRM: An Obstacle-Based PRM for 3D Workspaces, *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR), pp. 155-168,* 1998

[3] H. Arai and O. Khatib, Experiments with Dynamic Skills, *Proc. of 1994 Japan-USA Symposium on Flexible Automation, 81-84,* 1994

[4] B. Aronov, M. De Berg, A. F. Van der Stappen, P. Ŝvestka, and J. Vleugels. Motion Planning for Multiple Robots, *Discrete and Computational Geometry, 22:505-525,* 1999

[5] J. L. Bentley, Multidimensional Binary Search Trees used for Associative Searching, *Commun. ACM, 18:509-517,* 1975

[6] R. L. Burden, J. Douglas Faires, Numerical Analysis 7th ed., *Wadsworth Group,* 2001

[7] A. J. Briggs, An Efficient Algorithm for One-Step Planar Compliant Motion Planning with Uncertainty, *Algorithmica, 8(3):195-208,* 1992

[8] S.-W., Cheng, O. Cheong, H. Everett, and R.W. van Oostrum, Hierarchical Decompositions and Circular Ray Shooting in Simple Polygons, *Discrete and Computational Geometry, 32, pages 401–415*, 2004

[9] B. R. Donald, The Complexity of Planar Compliant Motion Planning under Uncertainty, *Proc. ACM Symp. on Computational Geometry,* 1988

[10] M. A. Erdmann, On Motion Planning with Uncertainty, *Technical Report AITR-810 MIT Artificial Intelligence Laboratory,* 1984

[11] K. Goldberg, Orienting Polygonal Parts without Sensors, *Algorithmica, 10(2):201-225,* 1993

[12] V. Koltun, Segment Intersection Searching Problems in General Settings, *Proceedings of the seventeenth annual symposium on Computational geometry 197-206,* 2001

[13] S. M. LaValle, Planning Algorithms, *Cambridge University Press* (or http://msl.cs.uiuc.edu/planning/), 2006

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

[15] T. Lozano-Peréz, M. T. Mason and R. H. Taylor, Automatic Synthesis of Fine-Motion Strategies for Robots *IEEE Trans. on Computers (C-32),108-120,* 1983

[16] M. T. Mason, Mechanics of Robotic Manipulation, *MIT Press,* 2001

[17] M. T. Mason, Mechanics and Planning of Manipulator Pushing Operations, *Int. Journal of Robotics Research, 5(3):53-71,* 1986

[18] M. T. Mason, K. M. Lynch, Dynamic Manipulation, *Proc. IEEE/RJS Int. Conference Intelligent Robots and Systems, pages 152-159,* 1993

[19] M. Strandberg, Augmenting RRT-planners with local trees, *Proc. IEEE Int. Conference on Robotics and Automation, pages 3258–3262,* 2004

[20] M. Sharir and S. Sifrony, Coordinated Motion Planning for Two Independent Robots, *Proc. of the fourth annual symposium on Computational Geometry, pages 319-328,* 1988