Two-level RRT Planning for Robotic Push Manipulation

Claudio Zito, Rustam Stolkin, Marek Kopicki and Jeremy L. Wyatt

Abstract— RRT planning is a well developed paradigm for solving motion planning problems where the search occurs directly in the configuration space of the object to be moved. In manipulation however, the object to be moved is indirectly controlled by contact with a robot manipulator. This is a different problem and one that must be solved if RRT planning is to be applied to robot manipulation of objects. The simplest version of this problem is one in which the robot has a single finger. The planning problem is thus to search for a sequence of pushes that will move the object from a start to a goal configuration. Our main innovation is to split the planning problem into a RRT planner operating in the configuration space of the object, and a local planner that generates sequences of actions in the joint space that will move the object between a pair of nodes in the RRT. We show that this two level strategy enables us to find successful pushing plans. We also show the effect of varying the number of pushing actions that are randomly selected at each stage.

I. INTRODUCTION

In this paper we show how to extend RRT planning to the case where we wish to achieve motions of an object in a configuration space by manipulating it with a robot arm. This touches on the general problem of how to plan actions in one space (e.g. the joint space of an arm) which have an effect on another space (e.g. the configuration space of an object) where there is no inverse model of the effects. In this paper we present a two level planner based on RRT which is able to solve this problem. We begin by reviewing the literature in our domain of application, which is that of push manipulation. We then describe our algorithmic approach, and present results from a simulation of the problem.

A. Push manipulation in robotics

Pushing operations are encountered frequently in robotics, but have received comparatively little attention in the research community. In one sense, pushing is perhaps the most primitive kind of manipulation, but push manipulations are interesting and challenging in that (especially in 3D problems) they provide a large number of unstable positions of the workpiece, and the relationship between applied pushes and the resulting workpiece motions are complex and hard to predict and control. Executing a successful pushing operation necessitates a planning task which is therefore non-trivial, and this is what is investigated in the present paper.

Pushing may have significant practical applications: a robot may be able to push an object which is too heavy for it to lift; robot vehicles without manipulators may move objects by pushing them [1]; pushing may be of importance for manipulation at the meso, micro or nano-robot scales [3].

Pushing is also important in that push contacts are fundamental to more complex tasks such as grasping [12]. When a two fingered gripper or a multi-fingered hand approaches a grasp configuration, uncertainty (object geometry and pose are subject to sensing accuracy, fingertip poses are subject to proprioceptive accuracy) means that one finger will typically contact the workpiece before the others, resulting in a single finger pushing phase before a stable grasp is achieved. Furthermore, any grasp is achieved as a combination of pushing forces from the grasping fingers, and in-hand dexterous manipulation motions are essentially the (non-linear) superposition of the effects on a workpiece of pushing motions due to each of the contacting fingers.

Mason [11], was the first to identify pushing operations as fundamental to manipulation, especially grasping. Mason developed a detailed analysis of the mechanics of pushed, sliding objects and determined conditions required for various 2D motions of a pushed object. In [16] Peshkin and Sanderson attempted to put quantitative bounds on the rate at which these predicted motions occur. Lynch [10] developed a method for finding the set of all possible motions of a sliding object, in response to an applied push. More recently, Cappelleri et al. [3] have experimented with push manipulation of 2D sliding objects at the micro-manipulation scale. However, all of this work is restricted to planar sliding motions of what are effectively 2D objects. There is little literature addressing the more complex problems of push manipulations on real 3D bodies, which are free to tip, topple or roll, as well as slide, and for which achieving a desired tipping or toppling effect on the workpiece may actually be a critical objective of the manipulative operation.

Our previous work [7] deals with the problems of learning forwards models to predict the workpiece motions that will result from applied pushes and, [4] [13], the problems of visually tracking the poses of pushed objects. In this paper, we address the problem of planning a sequence of pushes which will deliver a rigid body to a desired location and orientation.

B. Planning manipulations for pushed objects

In robotics, as described in [9], motion planning was originally concerned with problems such as how to move a piano from one room to another in a house without colliding with obstacles or bounding walls. The state-of-the-art algorithms to solve such problems are based on randomly
sampling points or poses in the free configuration space of the object being moved, e.g. [5].

However, such methods present two main problems for the domain of push manipulation. Firstly, there is no guarantee that poses suggested by the planner are actually reachable by applying pushes. Secondly, we require the additional information of what pushes should be applied to move the workpiece along the sequence of planned poses. Critically, there is often no way of knowing what the outcome of a push will be, except by trialling it in a predictor (here for proof of principle we use a physics simulator [15], however learned predictors are may also be advantageous [7]), for which no inverse model may be known. Hence, planners which span the space of possible motions with an interconnected graph-like structure of paths are not appropriate, because they are unlikely to be executable. Instead, what is needed is a planner, which uses predictive forwards models (of the workpiece motions caused by pushes) to span the working space with a branching tree-like structure of possible workpiece motions.

RRTs are popular techniques for path planning with kinodynamic constraints in high-dimensional spaces. Unlike other methods (e.g. PRMs [5]) they provide a branching tree-like search of the motion space, and can be intuitively considered as a Monte-Carlo approach to biasing search into the largest Voronoi regions of the space which remains to be explored. We adopt an RRT approach because of these useful qualities, however we find it necessary to employ a two level planner, where an additional technique is needed to construct the sequences of pushes which are necessary to reach the nodes requested by the RRT branching algorithm.

C. Our approach

We break the problem into two levels: firstly, planning motions of the workpiece (using a standard RRT planner) and, secondly, planning the actions of a robotic finger to achieve those workpiece motions (using a finger push planner that employs a randomised depth first search procedure). Crucially the searches for the solutions to each of these problems are interleaved. This division is necessary to maintain the desirable property of an RRT: that its search is biased so that the next node is likely to inhabit the largest unexplored region of the Voronoi diagram of the configuration space of the workpiece being manipulated. At each step the RRT generates a new candidate node. The finger push planner then randomly generates \( N \) pushes, and one of these is selected which moves the workpiece in a direction which maximises the rate at which the distance to this candidate node is reduced. That push is executed in the finger push planner until the distance to the candidate node starts to rise again, at which point \( N \) new pushes are generated, and one is selected. This process is repeated until the finger push planner has produced a sequence of finger motions that bring the object within some threshold distance of the candidate node. A new node is then added to the RRT for the object pose at the termination point of the final push (note that this will typically be slightly different than the original candidate node). The whole process then repeats with the RRT planner generating a new candidate node. The entire procedure terminates when a sequence of finger pushes have been found that carry the object to within a specified threshold distance of its target pose.

The resulting algorithm has some useful distinguishing properties. It uses only a forwards model to compute an action plan for robotic pushing manipulations, for which inverse models are typically unknown. Furthermore, it is robust against getting trapped in local minima. RRT can be seen as a “random walking” approach to explore the continuous state space of workpiece poses. Because of the complex relationship between these poses and the pushes that give rise to them, it is very easy for a conventional RRT approach to get stuck in a local minima of poses near to the current pose of the workpiece. Interleaving the additional technique of the randomised depth first search for pushes, avoids these local minima, and enables the RRT to continue growing new nodes towards the goal position using substantial step sizes.

II. PLANNING PUSHES TO REACH A GOAL POSE

Our method breaks this process down into two key components. First we will describe a global path planner, which uses an RRT algorithm to explore the configuration space of the workpiece by growing a tree of nodes towards the desired goal pose. Secondly, we will describe a local push planner, which uses a randomised depth first search procedure for finding locally appropriate sequences of pushes to reach from a previous node towards the next candidate node suggested by the RRT.

A. Global path planner

RRT planners iteratively expand the search tree by applying control inputs which lead the system towards randomly selected points. The RRT planner considers these selected points as new candidate nodes and tries to extend the closest vertex (node) of its existing search tree towards them. The important point is that the RRT planner does not directly compute the actions that are required in order to extend its tree, but instead selects new candidate nodes (in our case workpiece poses) which must be moved towards by applying an appropriate set of control inputs.

As described in [8], this kind of planning can generally be viewed as a search in a metric space, \( X \), for a continuous path from a given initial configuration, \( x_{\text{init}} \), to a target configuration, \( x_{\text{goal}} \). In conventional motion planning, a state transition function of a form \( \dot{x} = f(x,u) \) is enough to encode the kinematic and dynamic constraints of the problem. The vector \( u \) is typically selected from a set of possible control inputs, \( U \). In the simplest case, \( u \) can be analytically computed and specifies a particular direction in the metric space in which we extend the RRT. The vector \( \dot{x} \) represents the derivative of the state \( x \) with respect to time. A new state \( x_{\text{new}} \) which will become a new node of the search tree is simply computed by integration of \( \dot{x} \) over a fixed time period \( \Delta t \).
In push manipulation however, deriving a sequence of control inputs for the robotic manipulator is a complex problem per se. The workpiece is caused to move by (typically unstable) contacts with the robot’s end effector, and computing the choice of the appropriate control inputs is dependent on the ability to predict how the workpiece will behave under a variety of different possible pushing actions. In the case of pushing, we are not usually able to explicitly compute the correct action to achieve a desired workpiece motion, although we can make use of physics engines (or even learned forward models) to predict the outcome of a possible action. Hence, an intuitive approach is to try some pushing actions in different directions until we find one which causes the workpiece to move in a desirable direction.

More specifically, in our scenario the space \( \mathcal{X} \) represents the set of all possible configurations of the workpiece in \( \mathbb{R}^3 \) with respect to a global frame of reference \( \mathcal{O} \). Any point \( x \in \mathcal{X} \) is expressed as \( x = [R_O p_Q] \), where \( R_O \in \mathbb{R}^{3 \times 3} \) is a rotation matrix and \( p_Q \in \mathbb{R}^3 \) is a translation vector, both over the \( x, y, \) and \( z \) axis with respect to \( \mathcal{O} \). Hereafter we also refer to this set as the configuration space.

Unlike with more conventional uses of RRTs, we need to specify the control inputs of the system in a different space that should not be confused with the configuration space of the workpiece. The motion planning for the robotic manipulator occurs in the joint space, \( \mathcal{J} \), which defines the set of all possible configurations that the robot arm can assume. In particular, we want to select the change in the joint space, \( \Delta u \), which produces a desirable change in the workpiece configuration space, \( \Delta x \), such that the reduction of the distance between the current workpiece pose and the next candidate node, selected by the RRT algorithm, is maximised as follows:

\[
\bar{u} = \arg \min_{u \in \mathcal{U}} \rho(x_{rand} - f(x, u, \Delta t))
\]

where \( \mathcal{U} \) is a set of randomly selected actions; \( x_{rand} \) is the next candidate node, randomly selected by the RRT algorithm for expansion of its tree; \( f(\cdot) \) is the state transition function which applies the action \( u \) in the current state \( x \) for a fixed time interval \( \Delta t \); and \( \rho \) is a metric distance in \( \mathcal{X} \) which denotes the distance between two configuration states \( x, x' \in \mathcal{X} \) in terms of rotational and translational displacement.

The rotational or angular displacement between two workpiece poses is evaluated using the quaternion representation. Let \( q, q' \) be points on the 3D unit sphere about the origin in 4D quaternion space, which represent the rotation matrices \( R, R' \in \mathbb{R}^{3 \times 3} \) respectively, then it is possible to evaluate the difference in orientation between the two poses as:

\[
||q - q'||_{rot} = \frac{2}{\pi} \min(\cos^{-1}(qq'), \cos^{-1}(q(-q')))
\]

The rotation space is isomorphic to the 3D unit sphere in quaternion space, with diametrically opposite points identified [12]. The distance is computed as the smaller of the two possible angles between the images of the two rotations on this sphere. The distance thus is never greater than \( \frac{\pi}{2} \). For simplicity we normalise the value of the angular distance into the range \([0, 1]\).

We denote the translational displacement between two different positions \( p, p' \in \mathcal{X} \) as the Euclidian norm in \( \mathbb{R}^3 \):

\[
||p - p'||_2 = \sqrt{\sum_{i=1}^{3} (p_i - p'_i)^2}
\]

Now, given two workpiece poses \( x, x' \in \mathcal{X} \) we are able to evaluate the “distance” between them as a combination of the two aforementioned metrics as follows:

\[
\rho(x, x') = \frac{1}{2} ||Q(x) - Q(x')||_{rot} + \frac{1}{2L} ||p - p'||_2
\]

where \( p, p' \in \mathbb{R}^3 \) are the translational components of \( x, x' \) respectively; \( Q(x) \) is an operator which transforms the orientation of a pose \( x \) into the corresponding point on the surface of a 3D unit sphere in quaternion space; and \( L \) is a critical parameter of the workspace (e.g. the diameter of the region in which the robot manipulations take place). The purpose of \( L \) is to ensure that both rotational errors and translational errors occupy ranges between 0 and 1, so that the summation of (2) results in a meaningful cost function, in which rotational and translational displacements share equal

---

1Previously work by Brost [2] on push stability was used to do this for planning manipulation under uncertainty, here instead we use a physics simulation.
Algorithm 3 SELECT_ACTION

Input: $x, \dot{x}, x_{rand}, \Delta t$
Output: $x_{best}, u_{best}, \zeta_{best}$

$\zeta_{best} \leftarrow \infty$

$x_{best}, u_{best} \leftarrow \emptyset$

for all $u \rightarrow \mathcal{U}$ do

$x', \zeta' \leftarrow \text{SIMULATE_ACTION}(x, u, x_{rand}, \Delta t)$

if $\zeta' < \zeta_{best}$ then

$x_{best}, u_{best}, \zeta_{best} \leftarrow x', u, \zeta'$

end if

end for

end while

$\rho$ determined according to the metric $d$ from a pose at the closest vertex $x$ which will move a workpiece to the next candidate node $x_{near}$. Thus, define the beginning and end of a straight line path through $x$ near $x_{rand}$ as $x'$ (initialised as the vertex $x_{near}$) and the selected action $u$ for a fixed time interval $\Delta t$, $x'\prime \leftarrow G(x', u, \Delta t)$.

During the simulation, the local push planner behaves as a closed-loop controller and the actual workpiece’s configuration state is checked at each fixed time period $\Delta t$. After each period $\Delta t$, if the distance $\rho$ to the candidate node $x_{rand}$ has been reduced, then action $u$ is continued for an additional period $\Delta t$. Once the distance $\rho$ begins to rise again, rather than continue falling, the push is interrupted and the previous workpiece configuration is stored.

If the final stored configuration is identical to the initial configuration, $x_{near}$, then it means that this action is not useful and the planner discards it. The simulator then resets the initial position of the workpiece at $x_{near}$ and repeats the procedure by executing each of the remaining $N$ pushing actions. After all $N$ push trajectories have been simulated, the planner selects the most efficient push which minimises the distance $\rho$ to the candidate node $x_{rand}$. If no trajectory reduces $\rho$, then the planner stores this iteration as a failure and selects others $N$ random pushing actions. If a pre-set limiting number of failures is reached without reducing $\rho$, then the RRT planner will not extend the tree, and will instead randomly choose a new candidate node $x_{rand}$.

Once a candidate node, $x_{rand}$, and a useful pushing action $u$ have been found (where $u$ moves the workpiece nearer to $x_{rand}$, thus diminishing $\rho$), it is still unlikely that $u$ alone will bring the workpiece sufficiently close enough to $x_{rand}$ (satisfying a pre-defined maximum distance). It is more likely that a single action will move the object to an intermediate configuration which is still close to the initial one, $x_{near}$. If we settled for the single push $u$ alone, and this intermediate configuration was added to the RRT as a new node, we would lose the desirable behaviour of the RRT planner that yields a biased exploration towards unexplored regions. Applying a naive random action-selection behaviour to the RRT planner would transform its behaviour into a simple naive random tree, where randomly selected vertices are selected to be extended by random actions in order to generate new vertices. In this case, as described in [8], “Although one might expect the tree to randomly explore the space, there is actually a very strong bias toward places already explored”. Consequently most of the new generated nodes would remain in the same region as their parent nodes.

To avoid this overlapping or clustering of nodes, it is necessary that the local push planner be iterated to extend these ‘intermediate’ nodes until a configuration pose is found which comes close to the candidate pose, $x_{rand}$, which has been requested by the RRT planner (specified by some inverse kinematics solver, and a PRM based path planner[6] with optimisation is used to generate the trajectory in joint space. All the $N$ trajectories are simulated using a physics engine. Algorithms 3 and 4 define how the planner selects the next manipulative action by making use of a simulator for prediction. The physics simulator provides a prediction of the next workpiece configuration $x''$ which will result, given the current configuration $x'$ (initialised as the vertex $x_{near}$) and the selected action $u$ for a fixed time interval $\Delta t$, $x'' \leftarrow G(x', u, \Delta t)$.

B. Local path planner

Algorithm 2 shows our implementation of a local push planner which estimates a sequence of motor commands which will move a workpiece to the next candidate node $x_{rand}$ (which has been randomly generated by the RRT), from a pose at the closest vertex $x_{near}$ of the existing tree structure. The RRT planner supplies both inputs: the candidate node $x_{rand}$ and the closest vertex $x_{near}$ to it, determined according to the metric $\rho$ described above.

The local push planner now randomly selects $N$ possible finger trajectories in $\mathcal{J}$ which ensure that the end effector of the manipulator will collide with the workpiece. Each trajectory $N$ is generated as follows in the procedure SAMPLE_RANDOM_ACTIONS($x$). First a pair of points are randomly generated, each from a uniform distribution on two different surfaces of the object, neither of which is in contact with any other surface. These are then linked along a straight line path, which is extended away from the object in space by a fixed distance $d$ from each surface point. Thus a pair of points in configuration space have been defined that define the beginning and end of a straight line path through the object. These are then converted into joint space via an
maximum threshold distance). The intermediate nodes are encoded as part of a control sequence which defines a series of pushes (in our experiments typically 2-5 pushes) to extend the vertex \( x_{\text{near}} \) towards the new node \( x_{\text{rand}} \).

Once a series of pushes has been found that achieves a configuration \( x_{\text{best}} \) which comes suitably close to \( x_{\text{rand}} \), then the series of pushes is recorded and \( x_{\text{best}} \) is added as a new vertex to the RRT tree structure. This process continues until an RRT node is found that comes sufficiently close to the overall goal state of the manipulative operation.

III. RESULTS

A. Experiments

We have tested our push planning algorithm using a simulation environment based on the NVIDIA PhysX physics engine [15]. The test environment features a simulation model of a five axis manipulator (modelled after the Neuromics Katana 320 robot [14]), equipped with a single rigid finger with a spherical finger-tip, see Fig. 5.

In each experiment, the robot is tasked with pushing an L-shape object (referred to as a “polyflap” [17]) from a randomised starting pose, towards a randomly chosen goal pose. Polyflaps are interesting objects for testing push manipulation planners, because they can occupy a variety of different stable configurations, as well as a range of unstable modes. They are free to tip and topple, as well as slide and rotate upon a plane. In particular, we distinguish between the two flanges of each polyflap (shown as different colours in the figures), so that it is often necessary for the robot
Fig. 5. The image sequence shows a solution path computed by our planning algorithm. In this example, the planner selected 4 random push trajectories at each iteration. Image 01 shows the initial configuration of the experiment. The wire-framed polyflap to the right of the image is a ‘phantom’ to indicate the desired goal state. With respect to the initial configuration, the goal pose is translated by 28 cm and rotated by 90 degrees. Image 02 shows the trajectory in the free space of the arm in order to bring its end effector to the start position of the first push without colliding with the polyflap (employing an obstacle avoiding motion planner described in [6]). Images 01-04 show the first push. Images 03 and 04 show the execution of a critical push which makes the polyflap tip over, qualitatively changing its state by changing the face upon which it rests. Images 05-10 show a series of pushes which culminate in the polyflap resting in an unstable equilibrium pose along its folded edge. Note that this is a peculiarity of the physics engine test environment - in this case the physics engine considers this to be a stable pose where the polyflap does not tip if left alone. Images 12-13 show a sideways push. Images 14 and 15 show the final frontal push to align the polyflap’s pose to the desired one.

Fig. 6. The image sequence shows a series of pushes computed by our algorithm in which 8 different pushes were randomly selected at each iteration. Image 01 shows the initial configuration of the experiment from which the 5-axis robot executes the solution path. The wire-framed polyflap to the right of the image is a ‘phantom’ to indicate the desired goal state. With respect to the initial configuration, the goal pose is translated by 28 cm and rotated by 90 degrees. In this trial the algorithm has planned a distinctly different strategy from the one shown in Fig. 5. The manipulator first executes a sideways push, as shown in images 02 and 03, and then pushes against the vertical face of the polyflap to make it tip over (image 5). In images 06 and 07 the robot executes additional sideways pushes to correct the orientation. In images 08-10, the robot pushes the polyflap on its vertical face to move it onto the desired goal pose.
to ‘flip’ the polyflap in order to move it into the desired goal configuration. This choice of workpiece enables us to illustrate the truly 3D nature of the manipulations planned by our system, in contrast to the majority of related literature in which the workpieces are effectively 2D and are constrained to planar motions of sliding and rotating upon a flat surface.

Small five-axis manipulators, such as the Neuronics Katana, have a very limited envelope of effective operation. By coding this constraint into the planning algorithm, we see that the system plans pushes, both towards and away from the robot, such that the workpiece is never pushed out of the effective reach of the arm. This is a simple but very useful property of the system.

B. Examples

Figures 5 and 6 show two examples of experiments in which a sequence of pushes has been planned which successfully manipulates a polyflap into a desired goal configuration. In each image sequence, the wire framed polyflap (towards the right of each image) is a ‘phantom’ which is merely used to denote the desired goal configuration. The two faces of the polyflap are distinguished, one in pink and the other in grey, so that the reader can understand the orientation of the workpiece following each successive push by the robot.

A detailed explanation of the sequence of pushes and resulting workpiece motions is provided in the caption under each figure. Note (by observing the pink and grey faces of the polyflap) that the robot successfully re-orientates the workpiece with true 3D manipulations, by causing it to flip its resting base from one face to the other, in order to achieve the desired goal configuration.

C. Trends and evaluation data

For proof of principal, we have carried out 30 experiments in which the robot must plan and execute a series of multiple pushes. In all experiments, the robot successfully maneuvered the polyflap to within the specified threshold distance from the desired goal configuration.

During these experiments, we have varied the number \( N \) of randomly trialled pushing trajectories at each iteration (\( N \) is defined in section II, part B). We have performed ten experiments each for \( N \) equal to 4, 8 and 12 pushes respectively.

For each experiment, we have collected data on the total number of iterations, the total number of generated RRT nodes, and the rotational and translational distances and the combined cost function values each time a new RRT node has been generated. This data is charted in Figures 1-4. Charts 1-3 show how the error between the achieved workpiece pose and the goal pose decrease as successive nodes are added to the RRT tree. Chart 4 shows how the computational cost increases as additional nodes are added to the RRT tree. The convergent properties of the results suggest that an arbitrary degree of positional accuracy is achievable, with an expected trade-off between desired end-state accuracy and computational burden.

IV. Conclusion and Future Work

This paper has addressed the issue of how to plan a series of actions of a robot manipulator to achieve a transformation of an object. This is the only work we know of where complex 3D manipulations of non-planar objects have been planned and executed with pushing contacts. Previous work was restricted to 2D motions[3]. Our approach, by contrast, splits the planning problem into two parts: a global RRT planner, and a local planner which performs a randomized depth first search procedure in the action space of the robot. The push plans are iteratively refined by using a physics simulator to evaluate them. We have shown the ability of this two level approach to produce plans for a push manipulation scenario. Empirical experiments suggest convergence of the planned action sequences on arbitrarily accurate approximations to the desired goal state. We show how accuracy and computational expense vary with different parameter choices for the algorithm. Future work includes incorporating dynamic and state uncertainty into the local planning problem, and extending the approach to multi-contact manipulations, such as two finger grasps and in-hand manipulation.

V. Acknowledgements

We gratefully acknowledge funding from Fp7 IST projects CogX (215181) and GeRT (248273).

REFERENCES