Collision-Free Push Planning for Manipulation of A Box with A Roomba Robot

Amir Yazdani*, Jacob Harris†, Heath J. French‡, Yangyang Zhao§
*Department of Mechanical Engineering, University of Utah, Salt Lake City, Utah, Email: mojtaba.yazdani@utah.edu
†Department of Mechanical Engineering, University of Utah, Salt Lake City, Utah, Email: jacob84401@gmail.com
‡School of Computing, University of Utah, Salt Lake City, Utah, Email: heath.french@utah.edu
§Department of Mechanical Engineering, University of Utah, Salt Lake City, Utah, Email: yyang.zhao360@gmail.com

***Abstract*—The capacity to live independently is a major concern for the elderly. Technology geared toward helping the elderly live independently is growing. This project for ME EN 6225 proposes a push planning algorithm for the simplified problem in which a mobile robot pushes a walker to the elderly when they want to get up from bed. The proposed algorithm uses a combination of A* and RRT planning algorithm to solve manipulation of a box (walker). Its performance has been evaluated through simulation studies on different environments as presented in the result figures.**

I. Introduction

Mobility is one of the most important factors that contributes to quality of life. Physical degradation, because of age or illness, often decreases mobility and increases falls, leading to a downhill spiral of physical impairments including lack of independence. Each year, about 35% of individuals over age 65 experience one or more falls. Falls are the third leading cause of chronic disability worldwide6. The bedroom is the place where most falls occur within the home [1]. In the hospital, patient falls occur most frequently at the bedside, and some of the areas researchers have reported that the rate of bedside falls account for 50% of all falls [2] [3].
Currently, there are some technologies such as bed rails [4] or smart walkers [5] to help elderlies with getting up from bed or sitting on a coach. Bed rail are cheap affordable but they are passive and still human power is required to use them to get up. Smart walker are expensive and commercial ones are not smart enough to be used daily by elderlies.
In this project, authors tried to address the the concerns with home falls with a low-cost, intelligent mobile robot that will provide mobility aids, such as a walker, in the event that it determines that risk of fall is high.. The idea is shown in Fig. 1.
As a course project for ME EN 6225, a simplified version of the aforementioned bigger problem was defined in a simulation-based approach. In the proposed project, the problem includes the manipulation of a walker around an obstacle-filled area using a Roomba robot from a start position to a goal position. For simulation studies, all objects will be projected to the 2D plane. The Roomba as a circle, the walker as a box, and obstacles as simple polygons.
Several assumptions for the problem have been made:

- Only pushing has been used for manipulating the box.

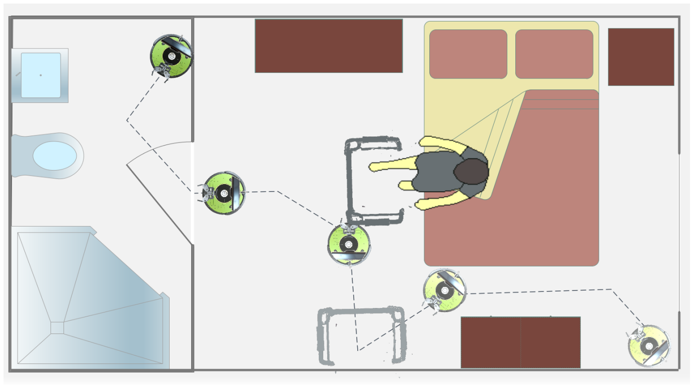


Fig. 1: When fall risk is high, such as when a person wants to get out of bed, robot brings a walker to him

- Start and goal position of the box and start position of the robot are known.
- Obstacles and their position are known.

The proposed approach to solve the above problem includes a high-level planner for push planning for the box and a lower-level planner for motion planning of the robot. Based on what has been learned up to the point of proposal, a modified A* planning algorithm as a high-level planner in combination with RRT algorithm were proposed as the solution.
The main contribution of this project is the development of a collision-free push planning algorithm for a box by a Roomba robot which includes a higher-level A* planner and a lower-level RRT planner. Simulation studies have been done in Python for different environments and with different start and goal positions for the robot and box. Results reveal the ability of the algorithm to do a collision-free manipulation for the box.

The report is organized as follows: A review on related works on fall prevention robots, care-providing robots, pushing and manipulation of objects by robots, and a related motion planning algorithm have been presented in section 1. Section 2, illustrates the approach and methods that have been used in the the solution algorithm for the problem. Modified A* and RRT are introduced in detail. Simulation studies and results for different environments are presented and discussed in section 4. Section 5 will cover analysis of the proposed algorithm. The conclusion, discussion and future directions for the project are presented in Section 6.

II. Related Works

Several researches have been done using robots for care-providing.

In field of pushing, Mason and Lynch [6] presented a model of the dynamics of pushing by a manipulator. Salganicoff et al. [7] created a forward model of object for pushing which used vision feedback. Later, learning algorithms were used to learn dynamic model of pushing objects and predicting result of pushing on unkown objects [8], [9]. Several researches have tried to learn the best contant location for a successful push [10]. They focus on extracting the shape of the object and determining contact locations that are good for pushing. Hermans et al. [11] provided an data-driven method for predicting proper contact locations for pushing unknown objects using vision. They also proposed a method for extracting the 2D shape of an object and generate points for performing pushing tests. They were pushing towards the centroid of the object from a given point using the feedback controller. Several researches also focused on delivery of an object from one position to another position. Quingguo et. al. [12] focused on finding appropriate pushing actions and developing a push planner which will complete the task using these actions. The planner they developed is based on a set of assumptions and a simplified model of two-agent point-contact push. They used that model for an off-line preprocessing step to identify push primitives. They showed that Under two-agent point-contact push, an object could exhibit a very complicated behavior and outcome of a push will depend on the pressure distribution, the geometry of the object, and the contact conditions.

III. Approaches and Methodes

As previously mentioned, the proposed algorithm uses A* as the high-level planner and RRT as a low-level planner. They are discussed in detail as follows:

A. A As The High-level Planner*

The standard A* graph search algorithm requires a known graph to search over. The search over this graph is complete and optimal with respect to the path length. This situation involves pushing a box through an environment where only the location of the box, robot, goal, and obstacles are known. A graph must be created in order to perform the search. Each state in the graph is a state that the box is able to achieve. The box is unable to move of its own accord and therefore requires an action, or push, by the robot to transition it into a new state. The robot transitions the box into a new state by pushing the box a small distance epsilon in a direction perpendicular to the box face. The location at which the push occurs is potentially all points on the parameter of the box. If all of the continuous push points were considered, the graph would be infinite and the A* search over this graph would no longer complete and would have infinite time and space complexity. To resolve this problem, a discrete set of push points should be chosen. For the project, four push points were selected at the center of each side of the box.

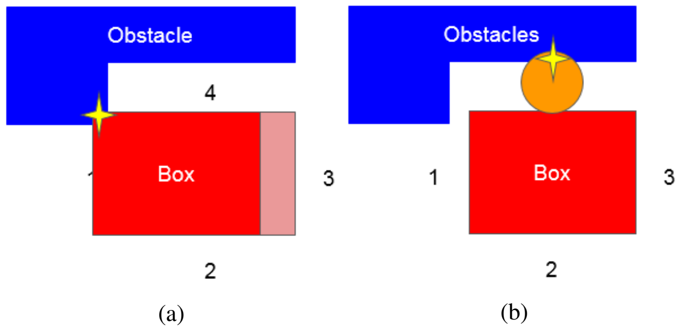


Fig. 2: Tests to pass before adding a node to the graph: a) first test, b) second test

It is not always possible for the robot to perform a given push action. This is because there might be obstacles blocking the robots access to a particular push point. If this is the case, the resulting state that a push would cause can never be achieved and should not be included in the boxs graph.
It is also possible that a given box movement may cause collision between the box and the environment, or a collision between the robot and the environment. Here, an assumption is made that the boxs width and height are greater than the boxs radius. It is also assumed that the push points are located inwardly enough on the box such that the robot will always be shadowed by the box during a push. This assumption guarantees that the robot will not come in collision with the environment during a pushing operation. If the box itself collides with the environment, the movement is not possible and the state in collision is not included in the search graph. Thus it can be seen that for a state to be added to the graph during graph creation, it should pass two tests:

1) Robot can achieve state necessary to enact the pushing action on the box.
2) A given push should not cause the box to be in collision with the environment.

Fig.2(a) and (b) demonstrates when a test 1 and test 2 failure would occur. A single test failure will cause the resulting box state to be dismissed as a graph state candidate. To reduce computational time in the event of a test failure, the most efficient test should be performed first. The test of a box collision with the environment is relatively non-complex when compared to testing if the robot planner can achieve a push point. This test, in many cases, involves a full RRT. Therefore the box collision test should be performed first.
With a method in place for graph creation, the order of graph creation and graph search should be established. Common convention would dictate that an entire graph should be created then searched over using a graph search algorithm. For this application, the graph creation is the most time intensive operation. Therefore it is desired to only create graph vertices when they will be actively involved in the search. The knowledge

of whether a vertex will be utilized only becomes available during the expansion phase of the A* graph search algorithm. While determining if the robot can achieve a push point location, a robot plan is created. To avoid unnecessary re-calculation of the robot plan, the plan is stored in conjunction with each node in the A* tree. Once the box goal is achieved, the robot plans are concatenated together in the back path. A* utilizes a summation of the backward path cost and a heuristic forward cost. This summation of costs is used when choosing the next node to expand. The heuristic used was the L2 norm or Euclidean distance.

$$Cost_{total} = h(s) + g(s) \tag{1}$$

$$h(s) = \sqrt{(x_1 - x_{1goal})^2 + (x_2 - x_{2goal})^2} \tag{2}$$

$$g(s) = \sum_{i=0}^{\textit{size of backpath}} cost(i) \tag{3}$$

The total value was calculated and added into the priority queue only after the graph expansion tests were passed. A check was completed to determine if a state had been previously visited and also to determine if a state was already in the queue at a higher cost. If the state existed at a higher cost, it was replaced with the lower cost node.

Algorithm 1 Push planning algorithm

```
1: $n0 \leftarrow$ *Initial Node*
2: $queue \leftarrow n0$
3: **while** $queue \neq empty$ **do**
4:     $n_i = queue.pop()$
5:     $visited.append(n_i)$
6:     **if** $n_i$ *is whithin of goal* **then**
7:         **return** *backpath*
8:     **for** $a$ *inBoxActions* **do**
9:         $s' = transitionfun(a, n_i, boxtesp)$
10:        **if** *s not in collision* **then**
11:            **if** *push point is reachable* **then**
12:                $G =$ *robot path cost* $+$ *previous plan cost*
13:                $H = Heuristic$
14:                $Cost_{total} = G + H$
15:                **if** *s not in visited* **then**
16:                    $queue \leftarrow s'$
17:                **if** *s is in visited with higher cost* **then**
18:                    $replace(s')$
19:     **return** *no plan available*
20:
```

B. RRT as the low-level planner

The available box actions are dependent on the robots capacity to perform the movement. A plan must be generated for every possible action that the A* graph search algorithm tries to use to branch. The plans produced for the robot can vary in complexity. This means that a cost must also be connected to each plan for A* to properly determine which state should be evaluated next. The robot planner is separated into two different phases. First, the planner determines if the robot is simply moving around the box to a different side and if no obstacles are near the box. This is done by creating a Big Box which is greater than the diameter of the robot on all sides and is located at the same location as the box. If this box is not in collision, then the shortest path around the box is created using another Medium Box that is greater than the radius of the robot on all sides. The corner points making up this Medium Box are known to be safe for the robot to reach. If the conditions for the simple plan are not met, then the planner uses RRT-connect to find a path to its goal.
The cost of the paths generated by the robot path planner are based on the step-size chosen for RRT-connect. For an RRT based plan, the number of actions required in the plan is returned. For a simple plan around the box, the total length of all the movements is taken and then divided by the step-size. Since the cost for turning our robot in these paths is not included, RRT-connect was specifically chosen since it would reduce the number of times that turning would be required.

C. Environment Modeling

In order to perform computational simulations that are both clear and close to the real world, environment files have been generated to describe the 2-dimensional space. There are four elements initialized in environment files. These are the robot, box, obstacles, and boundaries.
The robot is represented as a simple circle and is defined by a radius and a center point. Similarly, the box is represented by its center point, width and height. The position of the start location for the box and robot are defined. The location of the box goal is also defined.
As a simplification of real world objects, the obstacles are modeled as convex polygons defined by a sequence of vertices. Boundaries have been generated by assigning minimum and maximum values both for x-axis and y-axis. This is essential, as it will be used to prevent the robot from pushing the box into unknown fields.

D. Actions and Transition Function

As mentioned previously, the robot is only able to interact with the box at four push points. The push of the robot causes the box to change state. The amount of change in the box location is determined by the push distance epsilon. This change will occur in either the x or the y direction depending on the push point. The transition function returns the new state of the box and is a function of the current location of the box, the push point, and epsilon. The transition is deterministic

E. Collision Checking

Prior to collision checking, forward kinematic functions generate the forward kinematic for the robot and the box (both box and big box). These forward kinematic results will be used in collision checking. There are three different types of collision checking functions required for the algorithm.

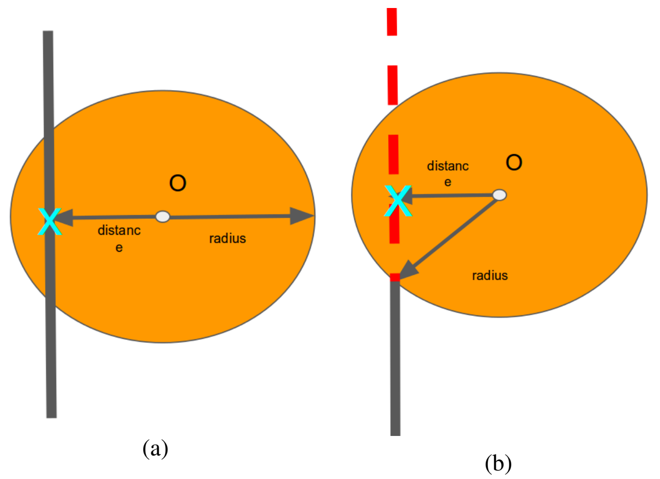


Fig. 3: Different situations in collision checking between circle and polygons

The first collision checking is of the polygon-polygon type. This type includes box-environment and big-box-environment collision checking functions. In these functions, collision checking performed between the box and each obstacle can be parsed into two minor parts. The first part is to check whether the vertices of the box are inside the obstacle (convex polygon). The other part is to check if an intersection exists between obstacle and box line segments. A collision-free node can only be extended when neither of these parts get classified as in collision.

The second collision checking type is for circle-polygon collision, which includes robot-box and robot-environment collision checking functions. For both of these functions, the first step is to determine if the robot center is inside its colliding counterpart. The robot is not represented as a discrete set of points and therefore the previously used line segment intersection collision check cannot be used. Instead, the closest distance between the circle center and each obstacle is checked to see if it is less than the radius of the robot. This collision check can have two possible configurations. Fig.3 shows these two possible configurations. Part (a) shows the robot in collision as described above; the distance between the robot center and the line is less than the radius.

Part b shows an instance where the robot will be not be in collision even when the distance from the continuous line is less than the radius. To account for this configuration, the location of the intersection between the line in question and the line perpendicular which runs through the robot center should be examined. Only if this location is between the vertices of the obstacle line, will the robot be considered in collision.

IV. Simulation Studies and Results

To check the performance of the proposed algorithm, different environments were developed with different numbers of obstacles and different configurations of obstacles. In these environments, the robot tries to reach the box from a robot start point and push the box to the boxs goal position. Each environment has distinct initialization locations and goal locations. A graph of the environment, visited nodes, and the resulted plan and run time for each study are provided as the reported results. Additional results in video form can be viewed at: https://goo.gl/iFGsvm

A. *Environment 1*

Environment 1 is the same environment used in HW2, which includes four polygons as obstacles, as is shown in Fig.4(a). In this figure, robot start position, box start position, and box goal position are represented by the yellow circle, red rectangle, and green rectangle, respectively.

Fig.4(b) shows the result of running the algorithm on environment 1. Yellow circles are the plan for robot, green rectangles are the plan for box, and red rectangles are visited nodes that were not included in the final path.

It can be seen in the figures that the robot successfully reaches the first planned push point of the box using the RRT-connect algorithm and it moves the box to its goal position. Result for different start and goal positions are available as video clips.

B. *Environment 2*

Environment 2 is a maze-shaped environment with three big rectangular obstacles. Similar simulation study has been done for environment 2 and results shows that the algorithm returns a path that can achieve the goal with good performance. Part (c) and (d) of Fig.4 represent environment 2 and the resulting plan, respectively.

C. *Environment 3*

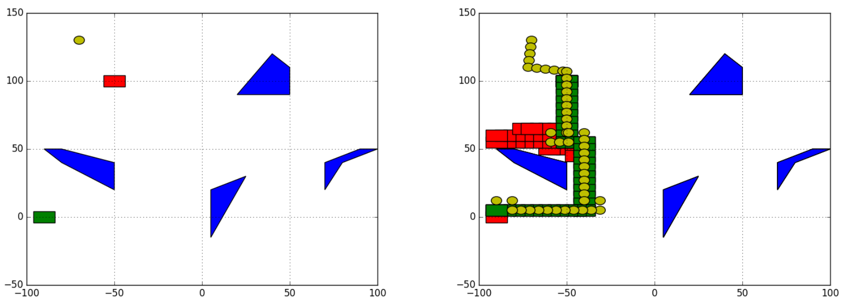
Environment 3 is also a maze-shaped environment with diagonal-edge polygons. This environment represents a good challenge for the planning algorithm. As seen in Fig.4(f), the algorithm successfully pushes the box around obstacles while minimizing the total cost.

D. *Environment 4*

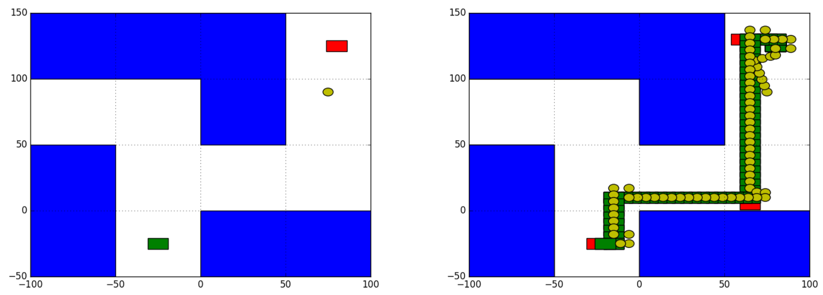
Environment 4 is the most challenging environment for the RRT planner as the box is always near obstacles, causing the RRT planner to be used exclusively. It includes some rectangular obstacles and a single triangular obstacle. The box must be pushed in a specific order to navigate through the passages capable of fitting the box. Only the robot can pass through the narrow passages to access different push points. As it is shown in Fig.4(h), the robot successfully determines and executes the necessary push order.

V. Analysis

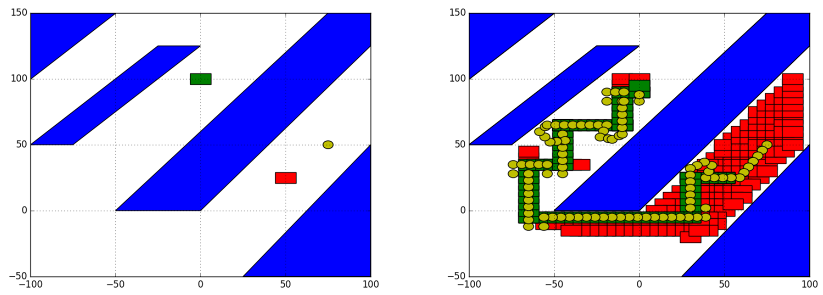
As mentioned in the approach section, the proposed algorithm is a combination of A* and RRT algorithms. Here, theoretical and practical analyses on performance of the algorithms is provided.



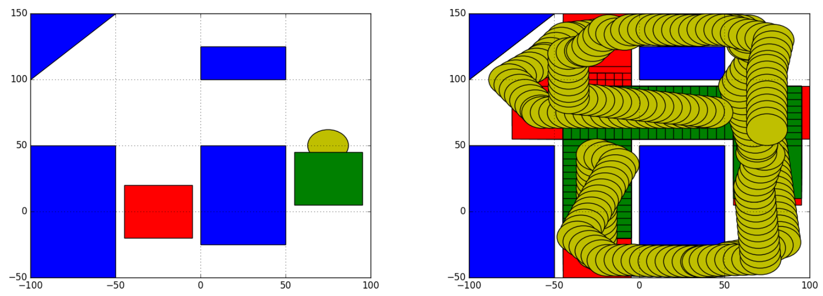
(a) Environment 1



(b) Environment 2



(c) Environment 3



(d) Environment 4

Fig. 4: Simulation results on different environments. In left column, robot initial poses are yellow, box initial poses are red, box goal poses are green. In right column, robot path are yellow, box bath are green and visited nodes for box are red.

A. *Completeness*

The A* portion of the algorithm is complete because it will find a path to the goal if it exists and exit if it has visited all possible states. In this application, the A* algorithm is reliant upon the RRT of the robot to add states to its graph. RRT is only probabilistically complete with an infinite number of samples. Here, the samples are limited for efficiency reasons. With RRT, the algorithm as a whole is not complete.

B. *Optimality*

With regards to the cost A* receives from the robot planner, A* is optimal. This is because A* will expand with priority to the next total lowest cost path. The heuristic portion of the cost is an underestimate of the total cost and is therefore admissible. The lower level RRT planner is not as optimal with respect to path length. This in turn causes the algorithm as a whole to be suboptimal.

TABLE I: Efficiency analysis

Case Study	Avg. frontier size	Avg. run time (sec)
Env 1	50.8	31.6
Env 2	62.3	31.2
Env 3	27.1	25.9
Env 4	8.3	160.2

C. *Efficiency*

To increase efficiency, the robot planner avoids using RRT-connect to find paths as often as possible. This significantly reduces the amount of time that is required to determine if A* related actions are feasible. When obstacles are constantly next to the box, the robot will use RRT-connect to get to other locations on a regular basis, thus increasing the runtime.
The efficiency of the planner is only slightly affected when the maximum number of samples for RRT-connect is increased. This is because the majority of the time, the robot is only moving from one side of the box to another and only needs a few samples. Generally, the full number of samples are only used when a path is not reachable.
Table (1) shows run times and frontier sizes for the different environments. These results are achieved while running the algorithm on a laptop with Core i7 CPU and 8GB of RAM.

VI. DISCUSSION

A. *Lesson Learned*

This project has provided an environment for two major lessons to be learned. This first lesson came through combining two different algorithms as one. The combination requires a deeper consideration of the finer points of each algorithm.
The second lesson came during modification of collision checking to handle a circular object. Initially, only the distance from the circle center to the line was considered. It was later discovered that because the equation for the line was continuous, a collision could be detected, even if the robot were far from the obstacle. Not taking the time to fully understand this collision check caused many hours of debugging.

B. *Future Work*

There are several areas in which the project can be expanded. The addition of more push points and pushes not directed through the boxs center of mass, would allow the box to change orientation. This change in orientation would allow the planner to be able to create plans for more complicated environments. Allowing the robot to pull the box, in addition to pushing, would increase the planners options for achieving the goal.
It might also be better to replace A* with a stochastic policy. The robot is commonly navigating in the same environment. Having a policy would decrease the time needed for replanning.
Finally, the transition function was assumed to be deterministic. This assumption is not true. A stochastic analysis should be performed and modifications made for either a policy or replanning.

REFERENCES

[1] Tinetti, Mary E., et al. ”*Risk factors for serious injury during falls by older persons in the community*.” Journal of the American geriatrics society 43.11 (1995): 1214-1221.

[2] Agostini, Joseph V., and Mary E. Tinetti. ”*Drugs and falls: rethinking the approach to medication risk in older adults*.” Journal of the American Geriatrics Society 50.10 (2002): 1744-1745.

[3] Hitcho, Eileen B., et al. ”*Characteristics and circumstances of falls in a hospital setting*.” Journal of general internal medicine 19.7 (2004): 732-739.

[4] Healey, Frances, et al. ”*The effect of bedrails on falls and injury: a systematic review of clinical studies.*” Age and ageing 37.4 (2008): 368-378.

[5] Grondin, Simon L., and Qingguo Li. ”*Intelligent control of a smart walker and its performance evaluation.*” Rehabilitation Robotics (ICORR), 2013 IEEE International Conference on. IEEE, 2013.

[6] Lynch, Kevin M., and Matthew T. Mason. ”*Stable pushing: Mechanics, controllability, and planning.*” The International Journal of Robotics Research 15.6 (1996): 533-556.

[7] Salganicoff, Marcos, et al. *A vision-based learning method for pushing manipulation*. University of Pennsylvania, 1993.

[8] Katz, Dov, and Oliver Brock. ”*Manipulating articulated objects with interactive perception*.” Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on. IEEE, 2008.

[9] Ruiz-Ugalde, Federico, Gordon Cheng, and Michael Beetz. ”*Fast adaptation for effect-aware pushing.*” Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on. IEEE, 2011.

[10] Kopicki, Marek, et al. ”*Learning to predict how rigid objects behave under simple manipulation.*” Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011.

[11] Hermans, Tucker, et al. ”*Learning contact locations for pushing and orienting unknown objects.*” 2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids). IEEE, 2013.

[12] Li, Qingguo, and Shahram Payandeh. ”*Manipulation of convex objects via two-agent point-contact push.*” The international journal of robotics research 26.4 (2007): 377-403.