IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011

Push Planning for Object Placement on Cluttered Table Surfaces

Akansel Cosgun Tucker Hermans Victor Emeli Mike Stilman

***Abstract*— We present a novel planning algorithm for the problem of placing objects on a cluttered surface such as a table, counter or floor. The planner (1) selects a placement for the target object and (2) constructs a sequence of manipulation actions that create space for the object. When no continuous space is large enough for direct placement, the planner leverages means-end analysis and dynamic simulation to find a sequence of linear pushes that clears the necessary space. Our heuristic for determining candidate placement poses for the target object is used to guide the manipulation search. We show successful results for our algorithm in simulation.**

I. Introduction

Natural environments often contain cluttered surfaces. Robots designed to operate in unstructured domains alongside humans must not only manipulate specified objects but also manipulate the environment itself. In this paper we examine the problem of placing objects onto cluttered surfaces. We focus on designing a non-prehensile manipulation strategy that applies linear pushes to objects that reside on the surface to create space for a target object. We refer to this problem as *footprint clearing*, where the footprint is the two dimensional boundary of the object to be placed when viewed from above. The term *clutter* represents any object or group of objects present on the surface.

This paper has two contributions. First, we propose a novel planner that selects a sequence of pushing actions to clear a space large enough to place the target object. Second, we introduce a heuristic that guides search by finding candidate placements that are likely to yield simpler manipulation plans. Our research is complementary to perception of surfaces and objects [1–3] and control of robot arms both for object placement and non-prehensile manipulation [4–10].

First, we examine the relevant literature in Section II, including the perception of clutter, non-prehensile manipulation, object placement, and related planning work on obstacle clearing. Section III defines the problem domain and Section IV describes our proposed algorithm in detail. Section V discusses experimental results produced in simulation with variations of domain complexity. Section VI places our planner in the broader context of clutter manipulation.

II. Related Work

Existing work on automated pushing [4–10] does not focus on pushing objects as a prerequisite to object placement. Research in perception has distinguished surfaces from objects residing on them [1–3] and recently [3] placed objects on cluttered surfaces assuming suitably large free space. We create free space by pushing and rearrangement planning.

Akansel Cosgun, Tucker Hermans, Victor Emeli, and Mike Stilman are with the Center for Robotics and Intelligent Machines and The School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA.

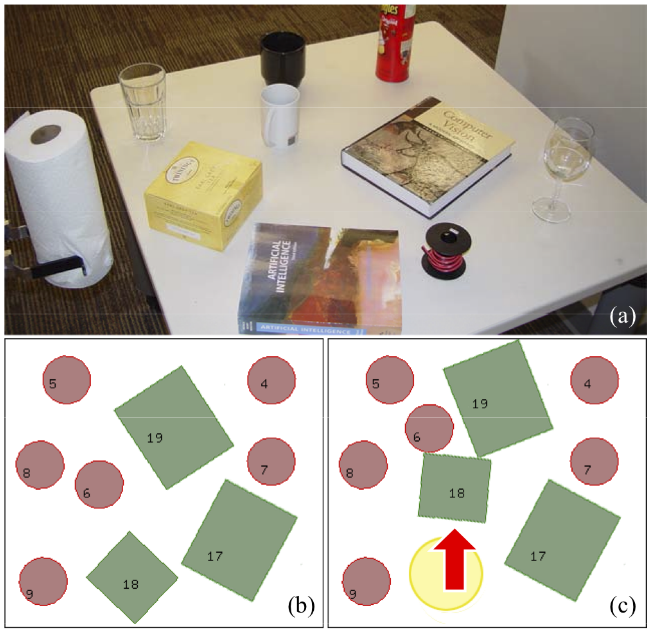


Fig. 1: Example tabletop push planning scenario.

A. Tabletop Object Pushing

Mason and Lynch developed the dynamics and control of pushing [4, 11]. Since then, pushing of tabletop objects has been performed with a variety of goals. Tabletop pushing has been used for segmentation in [5–7, 12]. Omrčen learns a pushing rule for flat objects that aligns them slightly off the table edge and facilitates grasping [9]. Dogar [10] proposes "push-grasping" in cluttered tabletop environments where the set of possible grasps is constrained. This approach pushes an object so that it rolls into the hand of the robot leading to successful grasps that avoid collisions with the clutter. Berenson [13] also studies grasp planning in cluttered environments with a focus on collision avoidance. To our knowledge, no work has considered pushing strategies that allow collision in order to create free space. Since the submission of this work, Dogar presented similar work to ours to solve the related problem of pushing objects to facilitate grasping of a target object [**?**]. As illustrated in Fig.1, our research is complementary to existing studies and would allow simpler strategies for grasping and placement.

B. Planning for Obstacle Clearing

Our approach to footprint clearing is related to two planning domains: *Rearrangement Planning* [14, 15] and *Navigation Among Movable Objects* (NAMO) [16–19]. However, the problem is not readily solved by existing methods due to the under-specified goal configuration, obstacle collisions and dynamic non-prehensile manipulation.

[20] showed the problem of pushing objects for rearrangement to be PSPACE-hard and gave an $O(n^3)$ algorithm to solve it for one movable object in an environment with n corners. Alami [21] broadened this domain to find sequences of manipulation actions for rearrangement of a single object. In contrast to early developments, our work focuses on domains that require the manipulation of multiple objects.

Ben-Shahar [14] approached rearrangement planning of multiple objects by planning independent motions of objects from start to goal configurations and sequencing these plans through a "permutation net." Ota [15, 22] has expanded this approach to the use of LRTA* to generate motion plans and heuristics for intermediate configurations. Both existing planners in this domain have focused on robots that interact with one object at a time. We allow objects to collide and plan through physics-based simulation.

Alternative approaches to handling movable obstacles are evaluated in the domain of NAMO [16]. Given a single navigation or manipulation task, the robot plans to make free space by manipulating unspecified objects in its environment. Table-clearing is even less constrained than NAMO [17, 19] in that the final configurations of all the objects are unspecified. Similar to rearrangement planners, NAMO planners have also focused on sequential single-object interactions. Despite significant differences, some of the concepts in this work are closely related to [18] which applies means-end analysis in order to efficiently compute plans for displacing objets with multiple interactions. We present a similar process of reverse search in our novel domain.

III. Problem Description

Consider a rectangular surface such as a tabletop, T, defined by opposite bounding corners $((x_{min}, y_{min}), (x_{max}, y_{max}))$, and a set of object shapes $O = \{o_1, o_2, \ldots, o_n\}$. The first $n-1$ shapes describe the objects residing on the tabletop and o_n defines the shape of the target object to be placed. Let $q = (P_1, P_2, \ldots, P_n)$ define the poses of all the objects, where $P_j = \{(x, y) : x \in \mathbb{R}, y \in \mathbb{R}, (x, y) \in o_j\}$ is the set of all points occupied by object shape o_j. A linear pushing action is defined as $u_{j,k} = (o_j, \phi_k, d_{j,k}))$, where the push is applied to object o_j in the direction ϕ_k for a distance $d_{j,k} = d(o_j, \phi_k, q) > 0$ with a constant velocity $\epsilon > 0$. The planner dynamic interactions between objects is governed by a function f, determined by the dynamic simulator, where $\dot{q} = f(O, q, u)$.

Apart from their shape, position and orientation, all objects have a boolean flag *indirectly_pushable*, which defines whether the object can be pushed indirectly by other objects or only directly by the manipulator. This distinction could be desirable for certain objects (i.e. tall, fragile) which may need extra care when being pushed.

Placing the object onto a crowded tabletop requires an agent to find a sequence of pushing actions $U = (u^1, u^2, \ldots, u^t)$ that will result in a state q_{end}, where three conditions are satisfied:

1) No objects intersect: $\bigcup_{P_i, P_j \in q_{end}, i \neq j} (P_i \cap P_j) = \emptyset$
2) Objects are within table borders: $x_{min} < x < x_{max}$, $y_{min} < y < y_{max}$ $\forall$ (x, y) in $P \in q_{end}$.
3) Objects are stationary: $\dot{q} = 0$.

Note that the search domain is infinite if there are no restrictions on the actions. We can limit the domain to allow only one push per object in a plan U, such that $o_i \neq o_j$ for every pair $(u_{i,k}, u_{j,l}) = ((o_i, \phi_k, d_{i,k}), (o_j, \phi_l, d_{j,l}))$ in U. Even in this domain, an exhaustive search won't be effective. Given a set of n objects and g pushing angles, the branching factor would be ng and total nodes in the tree would be $(ng)^n$. For $n = g = 10$, an exhaustive search tree produces 10^{20} nodes, a space too large for standard hardware. This formulation creates maximum of n pushes in any final plan. We allow k pushes per object, where k is the number of objects initially overlapping the goal footprint, resulting in a maximum number of kn. By combining a larger action space with informed heuristics, we aim to simplify the solution.

In order to achieve generality and efficiency, we decompose the task of placing an object in a cluttered environment into two stages: The first stage determines the goal placement pose P_n for the target object o_n. The second stage finds the set of push motions U, that satisfy the three requirements listed above. To bias towards shorter plans, we iterate stages 1 and 2 gradually allowing more complex plans.

IV. Proposed Algorithm

A. Goal Configuration Calculation

For a given tabletop configuration, infinite goal configurations satisfy the high level task requirement of placing object o_n somewhere on T. We wish to clear as little space as necessary. Therefore, our heuristic favors poses with less overlap with current objects. It determines a pose $\hat{P}_n$ by sampling among the least overlapping poses.

Given an overhead segmentation of the objects and the surface, we produce a binary image where object pixels have value 0 and surface 1. For M discrete orientations we produce a rectangular binary mask of the footprint of the object to be placed. This kernel has value 1 for object pixels and 0 for the surrounding pixels. When convolved with the binary image, this kernel produces a score at each pixel resulting in a grayscale image as shown in Figure 2b.

As a post-processing step for each orientation we examine only the object poses where the entire object footprint is contained within the boundaries of the table. We then determine the location of locally-maximal points within this interior. Regions of points with scores above 10% of the global maximum value are kept as modes in the distribution. Figures 2(c) and (d) show examples of these modes for two different orientations of the same object. The red points correspond to candidate locations.

Note that this heuristic produces a discrete representation of the configuration space of target object o_n. Each pose P_n^i in configuration space has a score S_i between 0 and the area of o_n. In the case where a solution that requires no pushes exists, the maximally scored pose P_n^* represents this solution and no further planning is required. In the common case where no such solution is found, subsequent configurations are produced by probabilistically sampling from the list proportional to the pose score. Pose P_n^i is sampled with

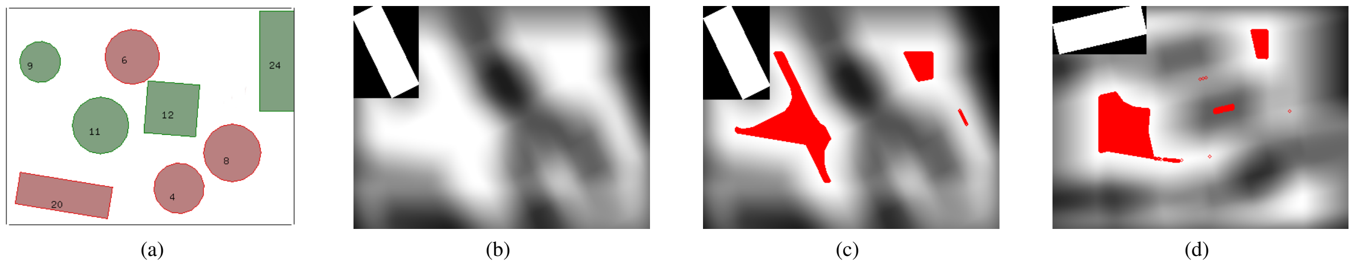


Fig. 2: (a) Current environment. (b) Footprint kernel and result of convolution with the environment. (c) The same convolution result with modes highlighted in red. (d) Footprint kernel, convolution results, and highlighted modes for another orientation of the same object.

probability $p(P_n^i) = \frac{S_i}{\sum S_j}$. Given a pose sample, we present a solution for push planning that clears the object footprint in Sections IV-B and IV-C.

B. Footprint Clearing Planner

Given a candidate goal pose $\hat{P}_n$ we introduce a planning algorithm that clears the footprint defined by $(\hat{P}_n, o_n)$. First, we explain the algorithm for the case where only one object o_1 overlaps with the placement footprint o_n. Next, we extend this approach to multiple overlapping objects in Section IV-C. The planner uses Breadth First Search (BFS) to search over the possible pushing actions $U_{o_1} = \{u_{1,k} = (o_1, \phi_k, d_{1,k})\}$. It determines the pushing distance $d_{1,k}$, via the following push termination conditions: (1) Any object collides with the table border or makes contact with $o \in O$ deemed not *object_pushable*. (2) $o_1 \cap o_n = \emptyset$. The interactions during the motion appear as a black box, which returns the resulting q_{new} and the first blocking object o_j to the planner.

Instead of a brute force search over all possible pushes, our algorithm begins by considering just the set of pushing actions U_{o_1} on overlapping object o_1. Suppose every push $u_{1,k}$ terminates following rule (1) defined above. Following each push of object o_1, we detect any object o_j that blocks the motion of o_1. After exhausting all single pushes of o_1, the planner resets to the initial configuration q_{init} and searches over the pushing actions U_{o_j} of each blocking object o_j. Given the resulting configuration from each push of o_j the planner backtracks to o_1 and searches over the pushes U_{o_1}, testing if any clears o_1 from the placement footprint. This approach guides the search by means-end analysis. The planner attempts to remove the obstructing object o_j from the path of object o_1. Since objects may constrain the movement of object o_j the planner recursively follows this procedure until it clears the footprint or reaches a maximum depth L_{MAX}. Pseudocode for the planner is given in Algorithm 1.

To clarify the algorithm, consider a simple example. Figure 3a illustrates an initial tabletop configuration and the accompanying search tree generated by the planner. The yellow rectangle o_5 represents the object placement footprint. The object cannot be placed since o_4 overlaps the placement footprint. At the first stage of the algorithm, the planner attempts all possible actions U_{o_4} on o_4 checking if any single push clears object o_4 from region o_5. The results of one such pushing action is shown in Figure 3b. Since a collision occurs between o_4 and o_6, the planner resets the environment to the

Algorithm 1 *pushPlanner*$(T, o_1, q_{init}, P_n, L_{max})$

```
 1: $Q$.enqueue$((q_{init}, o_1, \emptyset))$
 2: loop
 3:     if $Q$.empty() then
 4:         return Failure
 5:     end if
 6:     $(q, o, U) \leftarrow Q$.dequeue()
 7:     if $U$.size $> L_{max}$ then
 8:         return Failure
 9:     end if
10:     for $u_i \in U_o$ do
11:         loadConfiguration$(q)$
12:         $(q_{new}, o_{BLOCK}) \leftarrow$ executePush$(u_i)$
13:         $U_{new} = U$.append$(u_i)$
14:         if goalTest$(T, P_n, U_{new})$ then
15:             return $U_{new}$
16:         end if
17:         $Q$.enqueue$((q_{new}, o_{BLOCK}, U_{new}))$
18:     end for
19: end loop
```

initial configuration and the search continues by attempting push actions U_{o_6} on object o_6. Figure 3c shows the results of one such push from U_{o_6}. These pushes correspond to the search over Level 2 in the search tree. Following this push, the planner again searches over U_{o_4} checking if any push clears o_4 from the placement footprint.

Since no push of o_4 clears o_5, we examine the collision of o_6 with o_7 in Level 2. The planner begins its search at Level 3 attempting pushes U_{o_7}. Figure 3(d) shows one such push, where o_7 reaches the table edge. The planner backtracks to the previous level and attempts pushes U_{o_6}. For each push in U_{o_6} the planner backtracks again and attempts U_{o_4}. Figure 3(d) shows the resulting plan clearing the footprint.

In the following section we extend this algorithm to multiple objects overlapping the placement region. We also explain how the placement heuristic creates shorter plans.

C. Iterative-Deepening Breadth-First Search

In the case of multiple overlapping objects, our planner proceeds as follows. An overlapping footprint is selected by rejection sampling from the probability distribution in Sec. IV-A. Each object that overlaps the footprint is a *sub-goal*. For each sub-goal, the planner applies Algorithm 1. If a goal test succeeds then the planner continues with the next

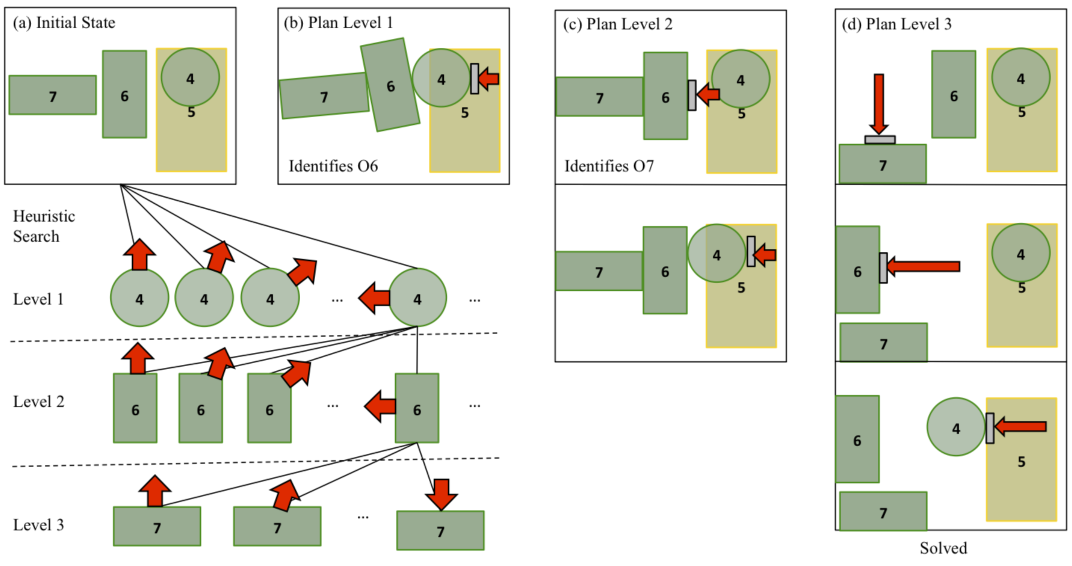


Fig. 3: An illustrative example that demonstrates the push planning algorithm.

Algorithm 2 *goalTest*(T, P_n, U)

```
1:  startObjCnt=size(getOverlappingObjects($(T, o_n)$))
2:  while $U$.notEmpty() do
3:      $u_{next} = U.pop()$
4:      executePush($u_{next}$)
5:      curObjCnt=size(getOverlappingObjects($(T, o_n)$))
6:      if startObjCnt < curObjCnt then
7:          return success
8:      end if
9:  end while
10: return failure
```

overlapping object starting from q'_{init}, where the sub-goal is satisfied. Algorithm 2 shows how the planner attempts prior pushing actions to evaluate if a sub-goal has been reached. A sub-goal succeeds if the number of overlapping objects has decreased, regardless of which object has been cleared.

For any initial configuration there are likely to be many solutions. Our planner prefers solutions with fewer pushes. We extend our algorithm to use Iterative-Deepening Breadth-First Search (IDBFS). We add an outer loop around Algorithm 1 which iteratively increases the maximum allowed tree depth from 0 to L_{max}. IDBFS allows our BFS planner to run on a number of different placement footprints in an attempt to find shorter plans which will require fewer pushes. Algorithm 3 explains the full IDBFS Push Planner, which takes into account multiple objects overlapping the placement footprint.

While the planner could exhaustively search over the placement candidates, our algorithm samples a finite number of candidate poses C_{max} at each depth of the IDBFS. Two factors motivate this sampling: efficiency and similarity of candidates. Even with the thresholding described in Section IV-A, the heuristic can still propose tens of thousands of candidate poses. However, neighboring poses are unlikely to produce distinct planning results. Sampling allows the planner to efficiently visit a diverse set of object placement

Algorithm 3 *IDBFSPushPlanner*$(T, L_{max}, C_{MAX}, o_n)$

```
1:  for $L = 0$ to $L_{MAX}$ do
2:      for $C = 0$ to $C_{MAX}$ do
3:          $\hat{P}_n \leftarrow$ getCandidateFootprint$(T, o_n)$
4:          $O_o \leftarrow$ getOverlappingObjects$(T, o_n)$
5:          $\hat{U} \leftarrow \emptyset$
6:          while $O_o$.notEmpty() do
7:              $o_1 \leftarrow O_o$.pop()
8:              $U' \leftarrow$ pushPlanner$(T, q_{init}, o_n, \hat{P}_n, L)$
9:              if $U'$ fails then
10:                 BREAK
11:             end if
12:             $\hat{U}$.push$(U')$
13:         end while
14:         if $\hat{U}$ succeeds then
15:             return $\hat{U}$
16:         end if
17:     end for
18: end for
19: return failure
```

poses, favoring those with higher scores.

Note that this variety of possible goal configurations motivates IDBFS. Instead of forcing the planner to find a long solution at a potentially difficult footprint goal configuration, we choose to abandon the current placement candidate and restart from a new, potentially easier, goal footprint. If a number of candidate locations fail, then it is more likely to reach the goal at a deeper level and the planner is allowed to search deeper. Our approach is inspired by Iterative-Deepening Depth-First Search (IDDFS) [23], which maintains the minimal solution length of BFS while taking advantage of Depth-First-Search's memory efficiency.

Our approach is designed to be efficient and applicable to practical domains. It is not complete. The planner only considers linear pushes instead of general trajectories. Furthermore, in the case of multiple objects overlapping the goal footprint, the ordering of objects to be pushed may affect its ability to find a solution. Since the overlapping objects are likely to contact each other when executing pushes, resolving the collision conflicts produces results similar to starting with a different overlapping object. Therefore we start a new search with a new placement footprint candidate instead of trying a different ordering of pushes if a sub-goal fails.

V. Experiments

We implemented our planner in simulation with the open source 2D physics engine Box2D [24]. Box2D incorporates contact, friction and restitution as well as collision detection. We modeled tabletop objects as rigid bodies of convex polygonal objects with equal densities. Push actions were applied with a rectangular gripper object o_g of dimension 2cm $\times$ 8cm. To apply $u_{i,k} = (o_i, \phi_k, d_{i,k})$, first o_g is placed centered at the center of mass of o_i with orientation ϕ_k and gradually moved along $(\phi_k - \pi)$ direction while checking collision between o_i and o_g. At the configuration where $P_i \cap P_g = \emptyset$, a final test between o_g and all other objects in O verifies whether the gripper can be placed without collision. If the configuration is collision free, the pushing action is feasible and o_g is moved in ϕ_k direction at a constant velocity.

We evaluated the performance of the algorithm under varying clutter percentages and ratios of *indirectly_pushable* objects. Clutter percentage is the ratio of the area occupied by the objects to the total surface area. We compared three scenarios: (1) all objects are *indirectly_pushable*, (2) half of the objects are *indirectly_pushable*, and (3) no objects are *indirectly_pushable*. The table size was 80cm $\times$ 60cm for all experiments, the initial object shapes $O_{init} = \{o_1, o_2, o_3\}$ and configuration $q_{init} = (P_1, P_2, P_3)$ of the first three objects were sampled randomly. At every iteration, the simulator randomly generated a circle of radius 7cm, square of side 14.4cm or rectangle of dimensions 8.8cm$\times$26cm with equal probability. The chosen object was scaled by a factor uniformly sampled from interval $[0.7, 1.3]$. The simulator provided the resulting table configuration q_{init} as input to the planning algorithm for placement. If the algorithm found a plan for placement, the simulator executed the pushes and placed the object before selecting a new object. When the algorithm failed, the simulator reset the environment to the initial configuration and repeated the procedure.

To keep the execution time of the algorithm reasonable, the maximum allowed tree level was set to 4, the maximum number of goal configuration samples per allowed tree level was 20 and the push angle resolution was $\pi/12$, resulting in 24 push directions. So in the worst case, 100 candidate footprint positions were tried. If a plan was not found for all the candidate positions, the algorithm terminated with no solution. We performed 1600 trial runs each for scenarios 1,2 and 3. Example configurations are given in Figure 4 and statistical results are shown in Figure 5.

The push planner had 100% success in cases with table clutter less than 40%. The average plan contained zero pushes for clutter percentages less than 20%, which implies all objects could be placed in empty spaces. The success rate rises higher with the percentage of *indirectly_pushable* objects. However, for very crowded tables (clutter percentage >70%) the planner rarely finds a feasible plan.

The total number of pushes searched reflects the complexity of the algorithm, since most of the computation comes from calculating the dynamics and running collision checks for the pushing action. The run time first increases rapidly, then suddenly drops. The compute time doesn't diverge, since at the peaks most of the cases hit the maximum allowed tree depth. Moreover, the search space actually declines for more crowded tables, since fewer feasible gripper locations exist, severely limiting the branching factor, eventually reaching zero. Note that we only analyze the successful cases for results in Figures 5b and 5c.

The average number of moves in a plan increases with the clutter percentage, until some point for the no-*indirectly_pushable* and half-*indirectly_pushable* cases. We attribute this to the fact that for a significantly crowded table with several *indirectly_pushable* objects, the planner either can't find a solution or finds a simple solution with a few pushes when the footprint object is small by chance. For the all-*indirectly_pushable* case, the solution continues to include more pushes as the table clutter increases. This is a result of allowing an object to move others, which increases diversity in table configuration and creates opportunities for different solutions. In contrast, for scenarios 2 and 3, many objects must be pushed one by one, which forces a more complex solution. Additional constraints arise since a more cluttered table restricts gripper placement to few locations.

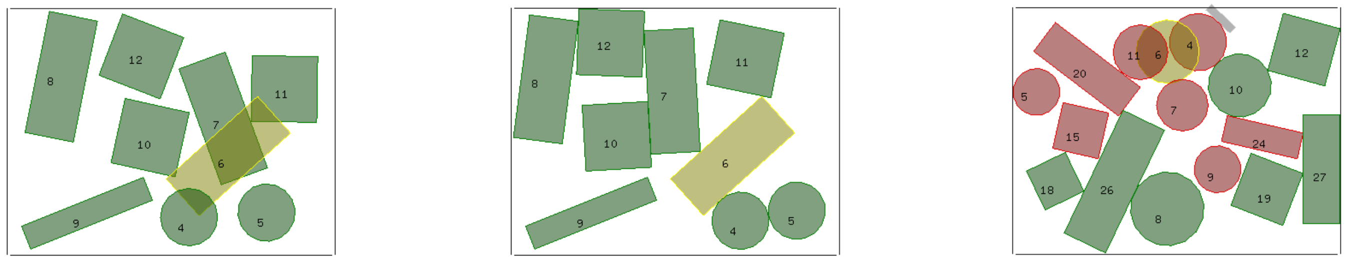
VI. Conclusion and Future Work

We have presented a novel algorithm for clearing surfaces such as tabletops to facilitate the placement of objects. Exploiting the candidate pose generating heuristic assists in finding shorter solutions by providing a variety of candidate locations to the planner. Applications to a physical robot will benefit from this bias towards simpler solutions. Plans with a greater number of push actions during real world operation induce a greater chance of divergence from the simulated physics used by the planner. By reducing the number of actions this error can be reduced and the robot can more robustly perform the task.

Object placement is a compelling task for robotics research that bridges a gap between pushing strategies such as those in [9, 10], tabletop clutter perception [3, 5, 12], and picking and placing [2, 25]. While our work has focused on the use of planning to enable placing of objects in environments, the same approach could be useful in enabling object grasping. As noted earlier [**?**, 9, 10] use pushing to assist in grasping objects in constrained settings. By setting the goal footprint to be an area surrounding the object to be lifted, and treating that object as an immovable obstacle, our planner could then be used to clear space to make room for grasping an object in a cluttered environment.

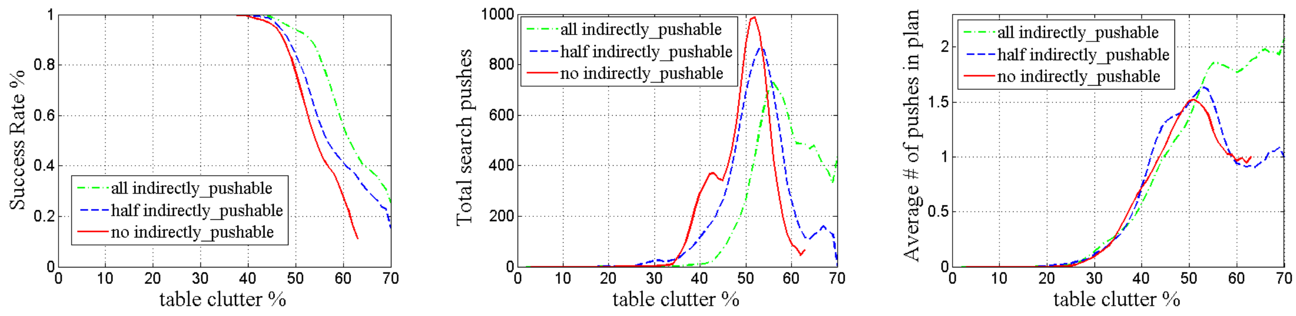
VII. Acknowledgments

This research was partially supported by NSF grant IIS-1017076. It was conducted as part of the course *CS 4649/7649 Robot Intelligence: Planning* at Georgia Tech.



(a) (b) (c)

Fig. 4: (a) A successful run from the all-*indirectly_pushable* scenario. o_6 is the candidate goal footprint, table clutter is 39.5% . (b) Table configuration after plan execution on (a); 24 total pushes searched in 4.8s resulting in a plan of 4 pushes. (c) A failed run from the half-*indirectly_pushable* scenario. Clutter was 51.7%, total number of pushes searched was 2,505 requiring 43.6s of execution.



(a) (b) (c)

Fig. 5: Experimental results on all three test cases. x axes are the clutter percentage and only successful cases are plotted for (b) and (c). (a) shows success rate of the placement algorithm, (b) shows total number of searched pushes for successful cases (c) shows the average number pushes in the plan for successful cases.

References

[1] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3d point cloud based object maps for household environments. *Robot. Auton. Syst.*, 56(11):927–941, 2008.

[2] Advait Jain and Charles C. Kemp. EL-E: An Assistive Mobile Manipulator that Autonomously Fetches Objects from Flat Surfaces. *Autonomous Robots*, 2010.

[3] Martin J. Schuster, Jason Okerman, Hai Nguyen, James M. Rehg, and Charles C. Kemp. Perceiving Clutter and Surfaces for Object Placement in Indoor Environments. In *ICHR*, 2010.

[4] M T Mason. Mechanics and planning of manipulator pushing operations. *Int. J. Rob. Res.*, 5:53–71, September 1986.

[5] Paul M. Fitzpatrick and Giorgio Metta. Towards manipulation-driven vision. In *in IEEE/RSJ Conference on Intelligent Robots and Systems*, 2002.

[6] Dov Katz and Oliver Brock. A factorization approach to manipulation in unstructured environments. In *Int. Symp. on Robotics Research*, 2009.

[7] Jacqueline Kenney, Thomas Buckley, and Oliver Brock. Interactive segmentation for manipulation in unstructured environments. In *ICRA*, 2009.

[8] Matthew T. Mason, Siddhartha Srinivasa, and Andres S. Vazquez. Generality and simple hands. In *International Symposium of Robotics Research*, July 2009.

[9] Damir Omrcen, Christian Böge, Tamim Asfour, Ales Ude, and Rüdiger Dillmann. Autonomous acquisition of pushing actions to support object grasping with a humanoid robot. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, Paris, France, 2009.

[10] Mehmet Dogar and Siddhartha Srinivasa. Push-grasping with dexterous hands: Mechanics and a method. In *Proceedings of 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, 2010.

[11] Kevin M. Lynch and Matthew T. Mason. Controllability of pushing. In *IEEE Int. Conf. on Robotics and Automation*, pages 112–119, 1995.

[12] Dov Katz and Oliver Brock. Extracting planar kinematic models using interactive perception. volume 8, 2008.

[13] Dmitry Berenson and Siddhartha Srinivasa. Grasp synthesis in cluttered environments for dexterous hands. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids08)*, December 2008.

[14] Ohad Ben-Shahar and Ehud Rivlin. Practical pushing planning for rearrangement tasks. *Trans. on Robotics and Automation*, 14, 1998.

[15] J. Ota. Rearrangement of multiple movable objects. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 1962–1967, 2004.

[16] M. Stilman and J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *Proceedings of the 2004 IEEE International Conference on Humanoid Robotics (Humanoids'04)*, volume 1, pages 322 – 341, December 2004.

[17] Kei Okada, Atsushi Haneda, Hiroyuki Nakai, Masayuki Inaba, and Hirochika Inoue. Environment manipulation planner for humanoid robots using task graph that generates action sequence. In *In: Proceedings of 2004 International Conference on Intelligent Robots and Systems*, pages 1174–1179, 2004.

[18] Mike Stilman and James J Kuffner. Planning among movable obstacles with artificial constraints. *International Journal of Robotics Research*, 27(12):1296–1307, Novemeber 2008.

[19] M. Stilman, J. Schamburek, J. Kuffner, and T. Asfour. Manipulation planning among movable obstacles. In *IEEE Int'l Conf. on Robotics and Automation (ICRA'07)*, 2007.

[20] G. Wilfong. Motion planning in the presence of movable obstacles. In *Proceedings of the fourth annual symposium on Computational geometry*, SCG '88, pages 279–288, New York, NY, USA, 1988. ACM.

[21] R Alami, T Simeon, and J P Laumond. A geometrical approach to planing manipulation tasks, the case of discrete placements and grasps. In *International Symposium on Robotics Research*, 1989.

[22] J. Ota. Rearrangement Planning of Multiple Movable Objects by a Mobile Robot. *Advanced Robotics, 23*, 1(2):1–18, 2009.

[23] R.E. Korf. Depth-first iterative-deepening* 1:: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[24] Box2d. http://www.box2d.org/, November 2010.

[25] Aaron Edsinger and Charles C. Kemp. Manipulation in human environments. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, 2006.