DOBBY: A Desk Organizing Bot BassicallY

Aayush Mudgal
Columbia University
New York, NY, 10027
am4590@columbia.edu

Mehul Kumar
Columbia University
New York, NY, 10027
mk3916@columbia.edu

Abstract—Humans are remarkably competent at processing cluttered spaces. Usually, once we have identified an object in the clutter, we use our arms and fingers to pick, grasp, push, slide, stack, etc, i.e. to manipulate the object in 3D space. In this paper, we introduce Dobby - our conceptualization of an object arrangement bot. It can identify and arrange work-spaces to reduce clutter. Our approach is extensively focused on integrating vision with grasping, utilizing mobility if available. For the same, we have adapted algorithms and developed our own heuristics to get to the final requirement. We also provide a survey of tools used.

Keywords - Humanoid; Robots; Motion Planning; Grasp planning; Computer Vision; Move-It; Fetch; ROS;

1 Introduction

Dobby [1] - a desk organizing bot basically, is a robot that clears work-spaces. In particular, our goal is to create an autonomous machine that can evaluate a scene and reduce clutter. This entity is built on top of existing frameworks like Moveit [SC13], Graspit [MA04] and agile grasping [Gua+16] that allows basic functionality required to work on a robot without dealing with all the complex structure and planning problems. We have also designed and described the algorithms for clearing the work-space. For demonstration purposes, we use Fetch [RBH15] as our Dobby model.

We decided to go for a breadth-first approach that allowed us to explore, understand and use numerous supporting libraries and packages. The task to reduce clutter is mechanical enough for an autonomous robot to achieve without assistance or training, while broad enough to fully use capabilities of industrial robots available today. A lot of the support programs are written as separate packages. Using these open source libraries, we built a program for a fetch robot to autonomously see, find and move objects in a 2D space as per strategies for packing the work-space. Since we use a fetch robot [RBH15], our code is grasp oriented and optimized for a mobile base with high (seven) degrees-of-freedom and at least one robotic arm.

The basic outline of the current project covers the following topics

- Identify objects in the work-space
- Plan movement of graspable objects into a compact space
- Clean up the working space using different algorithms

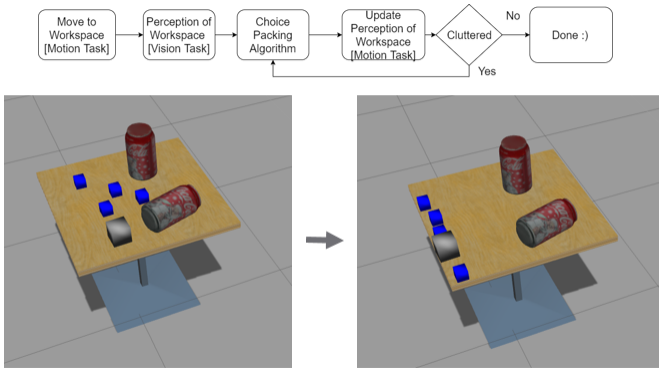

Figure 1. Task Progression for organizing a 2D-space autonomously

1. Link to GitHub Repository

The concept itself is a lot more flexible and broader than the scope of a course project, we propose the following ideas as addendum to our work.

- **Organizing 3D spaces**
  Going beyond 2D work-spaces will require more efficient algorithms as the degrees of freedom increase exponentially with dimensions of search space. But the generality of the concepts allows constraints to be mapped to a larger space using statistical optimization and filtering techniques, which are being developed now [**3d**].
- **Transform objects**
  Stack, rotate, transform objects for improved packing in 3 or less dimensions.
- **Irregular / connected objects**
  Better vision algorithms and sophisticated choice of camera angle and location can help us distinguish and/or change grasps to handle such a case
- **Feedback from vision cues**
  Vision could help in providing a feedback in case of failed grasps

2 Related Previous Work

Though lot of work has been done in the field of object detection [Ren+15], and grasping [MA04], and manipulation [Dis+01], [DWB06]. There has recently been a surge of interest in the field of autonomous grasping [Amo+14]. On detailed discussions with Iretiayo Akinola [2], we came up with this proposal of autonomously grasping objects to clear the clutter in a domestic work-space. This involved using existing algorithms for object detection, grasping and manipulation in a modified manner so as to full fill the task of better work-space arrangement. We found that this application of a humanoid robot is quite interesting, as everyone faces the issue of cluttered work-spaces, and having a robot to autonomously help with that is very motivating.

3 Perception of Work-Space

This involves finding the boundaries of the work-space i.e. the area where the objects are placed, and also identifying the objects in the work-space. We utilize a simple hough transformed based line segment finding algorithm [DH72] to find the rectangular boundaries of the table, using the image captured by the head camera of the robot. This can be further extend it to more complex ways of finding the work-space based on the RGB-D images of the scene. To identify the objects on the work-space, we perform image segmentation on the parts of the image that lies within the boundaries of the table.

2. LinkedIn Profile: Iretiayo Akinola

4 Desk organizing bot

Dobby is a desk organizing bot. In this proof of concept, we programmed a Fetch robot in a simulator to analyze and arrange a 2D space autonomously. In the present setting the work-spaces consists of a table, and the objects are randomly placed on the table. The aim of the packing algorithm is to increase the amount of free space. Or in other words, the packing algorithms minimizes the total area surrounding all the objects. This could also be viewed as minimizing the area enclosed by a rubber released from infinity which capsulates the work-space. Following is the detailed discussion of the workflow, divided into namely the algorithms for packing.

4.1 Packing Algorithms

These algorithms and heuristics are a subset of solutions to 'Bin-packing' [KV12] and 'Knapsack problem'. We have focused on algorithms that require low computation and depend on information about the work-spaces. Such algorithms fit the bill for robots since a robot can gather huge amount of information using its sensors. This approach is different from using increasingly computation intensive learning methods.

- **Randomized 2-bin algorithm**
  In this algorithm, we pick a random object and try to place it to the designated region of interest. This region of interest is initially an area the size of the largest object in the work-spaces, with some tolerance factor to handle error and uncertainty in measurement. As objects are added, the region is updated. We discovered that stacking of simple objects (cubes) is easier than we expected, as it only requires to adjust the height from which the grasped object should be dropped, taking care it is never too high, otherwise the cube would topple off.

- **Greedy algorithm**
  Greedy algorithm is the same In this algorithm, we pick a random object and try to place it to the designated region of interest. This region of interest is initially an area the size of the largest object in the work-spaces, with some tolerance factor added in practice. As objects are added, the region is updated. We discovered that stacking objects is easier in practice than we expected, only requiring us to toggle the z-coordinate in region of interest, taking care it is never too high.

4.2 Dobby Workflow

This section describes the work-flow of the program i.e. the sequence of instructions and modular actions that form the desk arrangement solution. This

Result: output : Object to be grasped
```
# objs : List of objects and positions
 returned by vision processor
# roi : Region of interest i.e. the part
 of work-spaces left to arrange
```
objs, roi = input;
output = [];
if !$objs.isEmpty()$ **then**
  o = random_object(objs);
  if $o.grasps() \neq None$ **then**
    $output = o.grasps[0]$;
  end
  else
    $updated\_obj = obj.remove(o)$;
    $output =$
     $Randomized\_2bin\_partition(updated\_obj)$;
  end
end
return output;

Algorithm 1: Randomized 2 Bin Partition

Result: output : Object to be grasped
```
# objs : List of objects and positions
 returned by vision processor
# roi : Region of interest ie the part
 of work-spaces left to arrange
```
objs, roi = input;
output = [];
if !$objs.isEmpty()$ **then**
  o = largest_movable_object(objs);
  if $o.grasps() \neq None$ **then**
    $output = o.grasps[0]$;
  end
  else
    $updated\_obj = obj.remove(o)$;
    $output =$
     $Randomized\_2bin\_partition(updated\_obj)$;
  end
end
return output;

Algorithm 2: Greedy Algorithm

- **Build a map with gazebo**
  This requires looking around the surroundings and creating a depth map based on head camera input. Motion planning is a well studied problem and we use *PlanningPlaceInterface* and poses from moveit_python package for mapping the surroundings. This phase is crucial in real world example, but has been removed from the demo. For the sake of clarity of goal, we start with a table in an empty environment with clutter, which we designed and then manipulated in gazebo simulator.
- **Initialize Dobby**
  Initialization involves two major steps. The First is loading the robot model. Improving these models corresponds directly to robot performance. Next is calibration. Although calibration is tightly coupled with robot model, calibrate_robot command in fetch_ros packages allows correcting errors in frames and transforms caused by accumulation of errors or the randomness involved in a high degrees-of-freedom setting. Calibration is also suggested in cases where robot runs into non-fatal errors.
- **Setup Clients**
  A lot of components need to be initialized before the robot/simulation can work. ROS packages provide the underlying functionality for most challenges involved. The packaged used are listed and described below
  - rospy
  - moveit
  - video_stream_opencv
  - graspit
  - agile

  In a physical environment like the robotics lab, we can use the tools from the Humanoid Robotics course, specifically HW1. Using the *fetch_navigation* package and SLAM, we can load the map of the region and run the simulation in real world setting. However, the navigation has not been tested extensively, hence not included in the demo.
  - MoveBaseClient()
  - FollowTrajectoryClient("torso_controller", ["torso_lift_joint"])
  - PointHeadClient()
  - GraspingClient(
  - CameraClient("/head_camera/rgb/image_raw")
- **Take position**
  Based on the position of the table, move the robot and initialize the robot
- **Assess work-spaces**
  Identify objects through the pointcloud returned by camera sensors. Create object queue with original and target location. Choose an algorithm for packing, by default set to randomized. Pass object queue to the algorithm
- **Figure out grasp**
  Once we know the object of interest, the object properties and the grasps are passed to the Moveit's Pickplace Interface. The robot tries to pick the object. It is not always necessary that the Pickplace interface would return in a success.

Thus, for error handling in this case, state of the robot is reset (if it fails 3 times) by tucking its arms and repeating the steps again. The object and the and grasps are passed to the Moveit grasp planner. Let the robot try to pick the object.

- **Place object**
  Use the region of proposal defined by the algorithm to select a place. Usually, it would be next to previous objects, with a tolerance gap between the objects. If stacking is allowed, the neighborhood can also be z-dimension, i.e. objects on top of previously placed objects.
- **Update the work-space**
  Update the region of interest. If stacking isn't allowed, unmark area already used.
- **Repeat until clean**
  Until all the objects lie inside the region, reassess the workspace and repeat previous three steps, in order.
- **Error handling**
  Since the agile planner [**??**] is stochastic and still under work, it has bugs that cause it to collide or miss the grasp. We faced numerous such issues while debugging. The robot has safety measures built-in which can also be triggered through rospy. After slight brushes or operating for a long time, the robot can be recalibrated, given it hasn't shutdown on unexpected events. If the system reports that there are no objects in the scene, the first step that is taken is to tuck the robots arms, and slightly change the field of view. This helps at times the robot's arms themselves occludes its view.
- Know your limits Identify if the space is unmanageable. This includes objects beyond capability, too small or too large
-

4.3 Tools Used

We use

4.3.1 ROS Framework

ROS [Qui+09] - Robot Operating System is an adaptable framework for writing robot software. This is, to some level, an analogue to android for mobile devices. Since the robots have much diverse structures and features, flexibility and scalability are major requirements. The ROS packages used in the project are described briefly here, to put the complexity into context.

- Gazebo
- MoveIt
- Tele-op
- fetch_ros This is the parent package that provides most functionality for the Fetch robot using ROS. This includes the calibration, kinematics and other fundamental functionality.
- AGILE grasp
- RVIZ interface

4.3.2 video_stream_opencv

4.3.3 Vision

Detected lines using filters and opencv in python Detected objects in a scene using opencv in python

4.3.4 Moveit

[SC13] MoveIt! is the state of the art software for mobile manipulation, as it incorporates the latest advances in motion planning and manipulation

5 Conclusions

5.1 Results

We tested our methods and algorithms against simple work-spaces. We limited the objects on the table to be simple cubes that are not very difficult to grasp. The work-space of the robot is limited to a table. In our experiments the objects in the work-space are assumed to be within the reach of the robot. We also enforce that the robot moves or tries to move only those object for which it thinks that it can move. In other words objects that are beyond the capacity of our robot, are left untouched by it as shown in Figure [2]

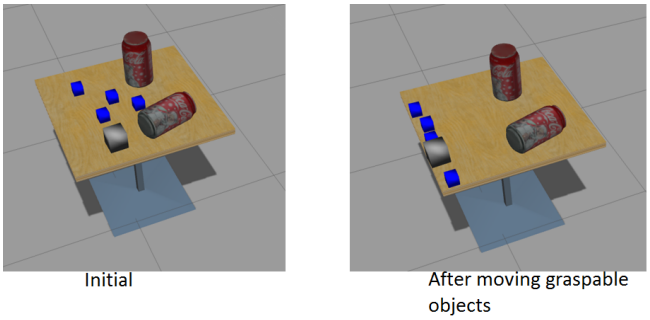

Figure 2. Cleaning work-space by moving graspable objects to one side

We were also able to stack cubes one over the another [3]. This involved figuring out the empty place where we wish to stack the objects. However because of the nature of objects and also due to error propagation while placing the objects, we could only reach to a height of 3 cubes at a time. The Figure [3] shows the initial state of the work-space, and the final state of the work-space after stacking cube one over the other. We observe that we are not able to stack very large number of cubes, due to the errors that that crept in due to the motion.

3. A video demonstration of the same is available at YouTube Video

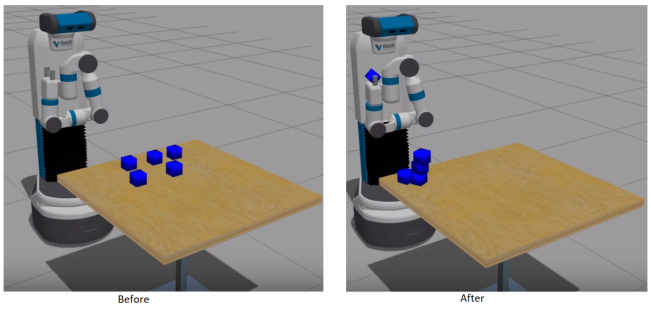

Figure 3. Stacking cubes one over the other

5.2 Future Work and Challenges Faced

In this section, we briefly describe the extensions and additions to this project. Given the capabilities and available code, a much complex application can be designed. We'll mention some suggestions that we wish to do in the future. The current system at times does not work as desired (or repeat itself) because of the probabilistic nature of path planning and grasping. This suggests that we should have a more constrained path planning operation, such that it minimizes the error due to path planning operation. Following are the list of ideas that we wish to work and improve upon.

- **Complex Work-spaces**
  Going to inaccessible and hazardous locations is where robots shine the most. Our design works for a small space, but for a more thorough and larger space will require caching and planning on objects not necessarily in the visual field. Adapting other computer vision techniques for more information is also relevant.
- **Improved Grasp Planner**
  Our current grasp planner is one of the publicly available ROS packages, and is highly randomized (probabilistic) and sometimes leads to collisions or missed grasps. Additional constraints to restrict certain risky movements, like error handling, updated pick and place code for accidentally fallen objects, etc, can be a fruitful pursuit.
- **Add heuristics / algorithms for efficient packing**
  We are currently using randomized and greedy arrangement algorithms. Preferably, there should be many such schemes available to a robot, which can then choose which fits it and the task better. For example, we can have an algorithms that results in minimum motion of the robotic hand, one which optimizes over the time taken for packing
- **Move around for better grasp planning**
  In the current setting the work-space of the robot is limited to the objects that are withing its reach. However this might not be the case when the work-space our huge. Thus we plan to have a systematic plan of the work-space first. Based on which it could plan the desired arrangement of the space. And could thus incrementally clear the work-space, by moving across the boundaries of the work-space
- **Transform, rotate objects**
  In the present system, the object were symmetric in the dimensions, hence any type of rotation of the object would result in the same update. However many day to day objects are not symmetric across all dimensions, thus for a better usage of the work-space area, such an object must be placed in an orientation where it is stable, and occupies the least area of the work-space
- **Integrating Vision for Error Recovery**
  Vision cues could also aid in identifying failed grasps, and thus could save time attempting to place failed grasps. This could be easily incorporated by threshold over the distance between the robot's fingers. The idea here is that the fingers of the robot would be nearly touching each other

ACKNOWLEDGEMENTS

We would like to thank Prof. Peter Allen for their encouragement, guidance, and teaching. We would also like to thank Iretiayo Akinola and Boyuan Chen for being always available at all time for help and guidance. Without their support, this project would not have been possible.

CONTRIBUTIONS

- **Aayush Mudgal [am4590]**
  : Responsible for the design and control of Dobby in gazebo. Formulated the code for work-space perception, grasping and placement of objects in the work-space based on the choice algorithm for packing. Contributed towards write-up.
- **Mehul Kumar [mk3916]**
  : Looked at ROS and the various packages and their integration. Contributed towards the write-up.

REFERENCES

[Amo+14] Heni Ben Amor et al. "Special issue on autonomous grasping and manipulation". In: *Auton. Robots* 36 (2014), pp. 1–3.

[DH72] Richard O Duda and Peter E Hart. "Use of the Hough transformation to detect lines and curves in pictures". In: *Communications of the ACM* 15.1 (1972), pp. 11–15.

[Dis+01] MWM Gamini Dissanayake et al. "A solution to the simultaneous localization and map building (SLAM) problem". In: *IEEE Transactions on robotics and automation* 17.3 (2001), pp. 229–241.

[DWB06] Hugh Durrant-Whyte and Tim Bailey. "Simultaneous localization and mapping: part I". In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110.

[Gua+16] Marcus Gualtieri et al. *High precision grasp pose detection in dense clutter*. 2016. eprint: arXiv:1603.01564.

[KV12] Bernhard Korte and Jens Vygen. "Bin-Packing". In: *Kombinatorische Optimierung*. Springer, 2012, pp. 499–516.

[MA04] Andrew T Miller and Peter K Allen. "Graspit! a versatile simulator for robotic grasping". In: *IEEE Robotics & Automation Magazine* 11.4 (2004), pp. 110–122.

[Qui+09] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe. 2009, p. 5.

[RBH15] Máximo A Roa, Dmitry Berenson, and Wes Huang. "Mobile manipulation: toward smart manufacturing [tc spotlight]". In: *IEEE Robotics & Automation Magazine* 22.4 (2015), pp. 14–15.

[Ren+15] Shaoqing Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[SC13] Ioan A Sucan and Sachin Chitta. "Moveit!" In: *Online at http://moveit. ros. org* (2013).