Assembling Objects With a Robotic Arm and Supports

Daniel Cabrini Hauagge
dc579

***Abstract*—In this work I investigate the problem of assembling pairs of objects using a robotic arm and obstacles as a support. The problem is decomposed into a sequence of grasping problems that take the objects from a random position into their fully assembled configuration using an obstacle as a support to hold one of the objects while the robotic hand moves the other object into it's final position. To determine what is a good grasp of the assembly I use SVM and hand labeled examples. I simplify the problem by assuming that the objects can be approximated as 2D objects. I show simulated results of final assemblies that demonstrate the viability of my approach.**

I. Introduction

Although robots have been used in assembly lines for decades, the environment in which they operate is highly structured and the tasks are fixed, so a hard coded sequence of movements can be successfully employed. As robots move into homes and start to interact with unstructured enviroments one of their tasks will be to assemble daily objects (*e.g.* while cleaning up a desk a robot might need to put the cap onto a pen or put papers into a file cabinet). The task in challenging at many levels, a robot will need to be able to identify a class of objects (*e.g.* plates, cups, chairs), reason about how they should be located inside the house, and be able to manipulate them successfully. In this work I address the problem of assembling pairs of objects, one novelty of this work is the use of the environment to assist in the assembly process.

II. Previous Work

The work done by [1] addresses the problem of learning the relation between the components of an object (e.g. on a microwave oven how the door moves with relation to the microwave or a drawer comes out of a cabinet). They formulate the problem as that of finding a minimum spanning tree in a graph, where the vertices are the components of the object and the edges model how one component moves with respect to the other. The cost of each edge (there are multiple edges linking the same vertices) models the complexity of the joint (higher cost for higher complexity) and how well the model can explain observations. In [2] another aspect of the problem is studied, that of devising strategies that allow a robot to dermine the different moving parts of an object. The problem is formulated as an instance of policy learning, where higher rewards are given to actions that allow the robot to gain the largest amount of information (measured in the number of degrees of freedom that can be discovered).

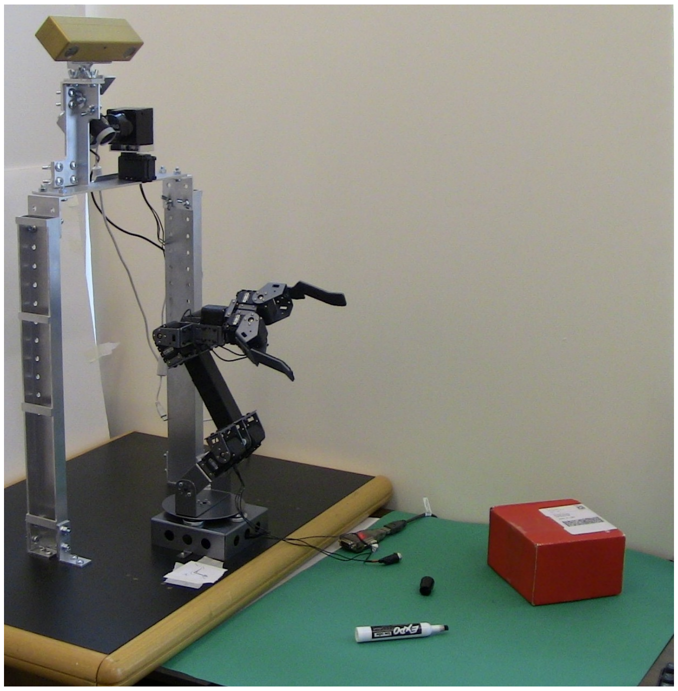


Figure 1: Setup used for this project. An AX12 robotic arm and a webcam.

III. Setup

For this project I employ an AX12 robotic arm and a webcam capturing color images at a resolution of 1280×800 pixels (see Figure 1). All the code was written in C++ using ROS and OpenCV. I used SVM^{light} for grasp rating. I included code for background subtraction that I've written for my *M.Sc.* project (this code is under the `thirdparty` directory inside `applications/demo_ax12_assembly`).

A. Calibration

As we assume that objects lye on a plane our calibration is greatly simplified. We need to know how to map points on the plane where the objects lye to points in the image. We can compute this correspondance with bilinear interpolation with four pairs of points (p_i, x_i), where $p_i \in \mathbb{R}^2$ is a point in the image and $x_i \in \mathbb{R}^3$ is the corresponding point in cartesian coords in the arm reference frame. Given a new point p in image coordinates we can estimate α and β such that $p = \alpha(\beta p_1 + (1-\beta)p_2) + (\alpha - 1)(\beta p_3 + (1-\beta)p_4)$ by solving a quadratic equation. Once α and β have been estimated we can obtain the corresponding point in cartesian coordintes by applying the same interpolation, now with the x_i's $x = \alpha(\beta x_1 + (1-\beta)x_2) + (\alpha - 1)(\beta x_3 + (1-\beta)x_4)$.

IV. Algorithms

Given an image I, the goal of our algorithm is to determine e sequence of pairs (p_{i1}, p_{i2}), where $p_{i1}, p_{i2} \in \mathbb{R}^3$ represent the coordinates of the fingers of the robotic arm, that bring the objects into their assembled configuration. We restrict our problem to objects that lie on a plane, this simplifies calibration of the arm with respect to the camera (as was described in III-A), dispenses the use of sensors that capture 3D data, and allows us to reason about contacts between objects in terms of their contours. The overall pipeline can be seen in Figure 2, in the following subsections we go over each of the stages in more detail.

A. Background subtraction

We assume that the background is of flat color, to remove it from the image we collect a set of images B_i and masks M_i that indicate where the background is (the green sheet in Figure 1 does not fill the camera's field of view, hence the need for a mask). We then compute a 3D histogram (one dimension for each color component) of the pixels in B_i indicated by M_i. We then fit a gaussian distribution to the 3D histogram H and replace the values in it's bins with value of the gaussian. To remove the background form a new image we lookup the value store in H for the color of the corresponding pixel, if it is greater that than a certain value we declare the pixel to correspond to foreground. The result of background subtraction can be seen in Figure 3.

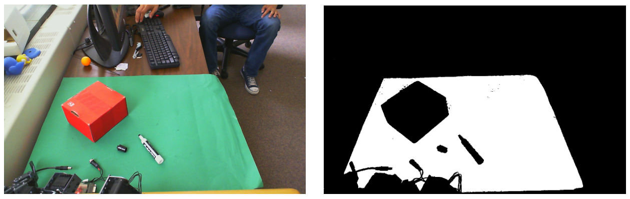

(a) Original image. (b) Result after subtracting the background.

Figure 3: Result of background subtraction.

As constrained as this setup might seem, we beleive that with the use of a 3D sensor we could substitute the constant color assumption with the assumption that objects are lying on a plane.

B. Connected component extraction

After subtracting the background from the image we detect the connected in the mask that was produced and filter out components that are too small or too big. Here we assume that in their initial configuration the objects are not touching each other, otherwise their connected components would be merged.

C. Pairing of objects and determination of desired final configuration

The next step in the pipeline is determining which objects go together and what should be their final configuration. This step currently requires human intervention. For pairing the objects the user clicks on the two connected components that should go together. To specify the final configuration the user clicks on two pairs of points, where each pair has one point in each of the two contours, each pair indicating that the corresponding points overlap in the final configuration. The connected component corresponding to the support is indicated previously and supplied to the system during initialization.

D. Recursive grasp planning

For this stage of the algorithm the contours of the connected components are extracted and the normal to each point is obtained by computing the gradient on the original mask and annotating the corresponding point with that information. The algorithm will determine if objects intersect each other by applying the transforms to the points on the contour and verifying if intersections between the contours occurr.

Once we know what is the desired final configuration of the parts we can reformulate the problem as a grasping problem (where a sequence of grasps might be needed) with 3 contact points, two representing the arm fingers and the third representing the obstacle which will be used as a support. First our algorithm moves one of the objects so that the final configuration is achieved, if there are no intersections between objects and obstacles then we proceed (we allow for intersections between objects to allow, for instance, the tip of the pen to be inside the cap when the cap is on), if there are intersections then we backtrack and start by moving the other object, if both attempts fail then the assembly fails[1]. The next step is to find the best graps for the assembled object, one constraint here that is not common to grasping problems is that the two arm fingers must be on one of the objects and the obstacle contact point must be on the other. The algorithms goes over all possible grasps and ranks them. Given the best grasp the next step is to determine if the grasp is viable. A grasp is viable if the obstacle contact point touches is in contact with the obstacle. If the grasp is viable then we can execute the grasp by moving the object that is in contact with the fingers towads the object that is in contact with the obstacle. If a grasp is viable the algorithm first tries to move the object in contact with the obstacle so that they are touching, this is done with a recursive call, where the desired final configuration now is that which puts the obstale and the object that should touch the obstacle in contact (with the constraint that the obstacle, while being viewd as another object, still cannot be moved). If the recursive call is successful, then we can move the object held by the fingers towards the object touching the obstacle so that we achieve the final configuration. If the recursive call is not successfull, then we try the next best grasp and so on. A diagram of an execution of the algorithm can be seen in Figure IV-D.

[1]There is obvious room for improvement here, we could move both objects away from the obstacle so there is room to assemble them.

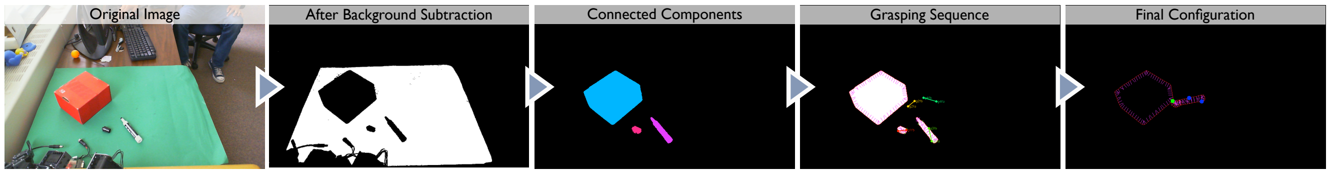


Figure 2: Full pipeline.

E. Learning to determine best grasping points

The algorithm described in IV-D goes over all possible 3-contact grasps that have one contact point in one of the connected components and the other two on the other connected component. To determine which are the best candidate grasps we use SVM to classify grasps into bad and good. The labeling is done manually and consists of clicking on 3 points around an object and labeling the grasp as good or bad (see Figure 5 for examples). The feature vector on which the model is trained encodes information about the contact points and their relation with each other, the components of the feature vectors are

$$n_i \cdot n_j \quad \forall i, j \in \{1,2,3\} \tag{1}$$

$$\frac{(p_i - p_j) \cdot n_i}{\|p_i - p_j\|} \quad \forall i \in \{1,2,3\} \tag{2}$$

$$\frac{(c - p_i) \cdot n_i}{\|c - p_i\|} \quad \forall i, j \in \{1,2,3\} \tag{3}$$

where n_i and p_i are the normal and location of the contour point i and $c = (p_1 + p_2 + p_3)/3$ is the centroid of the points (see Fig. 6). Eq. (1) encodes information about the angle between the pairs of normals, Eq. (2) encodes information how well aligned the points are, and Eq. (3) tells if a point's normal is pointing towards or away from the centroid. In total we have 9 components in the feature vector.

V. Results

For grasp selection I ran a test on a set of 36 images containing three different objects (a cap, a pen, and a screwdriver) and hand labeled 5 positive examples and 5 negative examples for a total of 360 labels. In a 10-fold cross validation the average acuracy was 89.17% with a standard deviation of 4.43%, the average recall was 89.45% with standard deviation of 7.0% and the average precision was 88.92% with a standard deviation of 6.63%.

An example of a successfull grasping sequence can be seen in Figure 7.

VI. Conclusions and Future Work

In this work we have examined the problem of assembling two objects using one arm and an obstacle as a support. We were able to reformulate the problem as a sequence of grasping problems and simulate successfully grasps with the algorithm. As future improvements to the system we would like to eliminate the need for human intervention when pairing the objects, a challenging problem since we need to identify generic instances of objects and be able to reason about how they relate spatially with each other. There is also room for improvement in the features used to learn good grasps. Due to noise in the contours we obtain very noisy estimates of the normals, which corrupt the feature vectors. To overcome this problem we would like to devise feature vectors that incorporate statistics such as averages and variance for the points. Another direction of investigation is to allow for a flexible number of contact points, right now we allow for only three and go over all possible grasps to search for the best one. This is obviously not a scalable strategy, as the number of contact points grows then number of possible grasps grows exponentially. Yet another extension to investigate would be to extend the algorithm to work for 3D data, which would require more sophisticated sensing.

Additional Work

I've committed code to the ROS repository into various modules.

- `applications/demo_ax12_assembly`
  Most of the code for my final project is contained in here, some of which might be useful to other projects, like the background subtraction code (OpenCV has a class for background subtraction but the problem that I found was that it continuously updates the model, so stationary objects will eventually be considered background), there's a class for background subtraction that can save it's state to file for later use and there's also a utility (`configBSROS`) to configure an instance of this class and then save the state to file.
- `control/ax12grasping`
  Contains the camera to arm calibration (for the 2D case).
- `applications/demo_ax12_camera`
  Utility that displays the stream coming from a camera (obtained from a topic published by the Brown University module probe) and publishes user clicked points (number of points published in each message is configurable) to a ROS topic, useful for debugging and prototyping.
- `export_packages`
  A script to take the packages listed in the file `export_list.txt` and upload them to the Google code repository.

References

[1] J. Sturm, C. Stachniss, V. Pradeep, C. Plagemann, K. Konolige, and W. Burgard, "Towards Understanding Articulated Objects,"

[2] O. B. Dov Katz, Yuri Pyuro, "Learning to manipulate articulated objects in unstructured environments using a grounded relational representation," in *Proceedings of Robotics: Science and Systems IV*, (Zurich, Switzerland), June 2008.

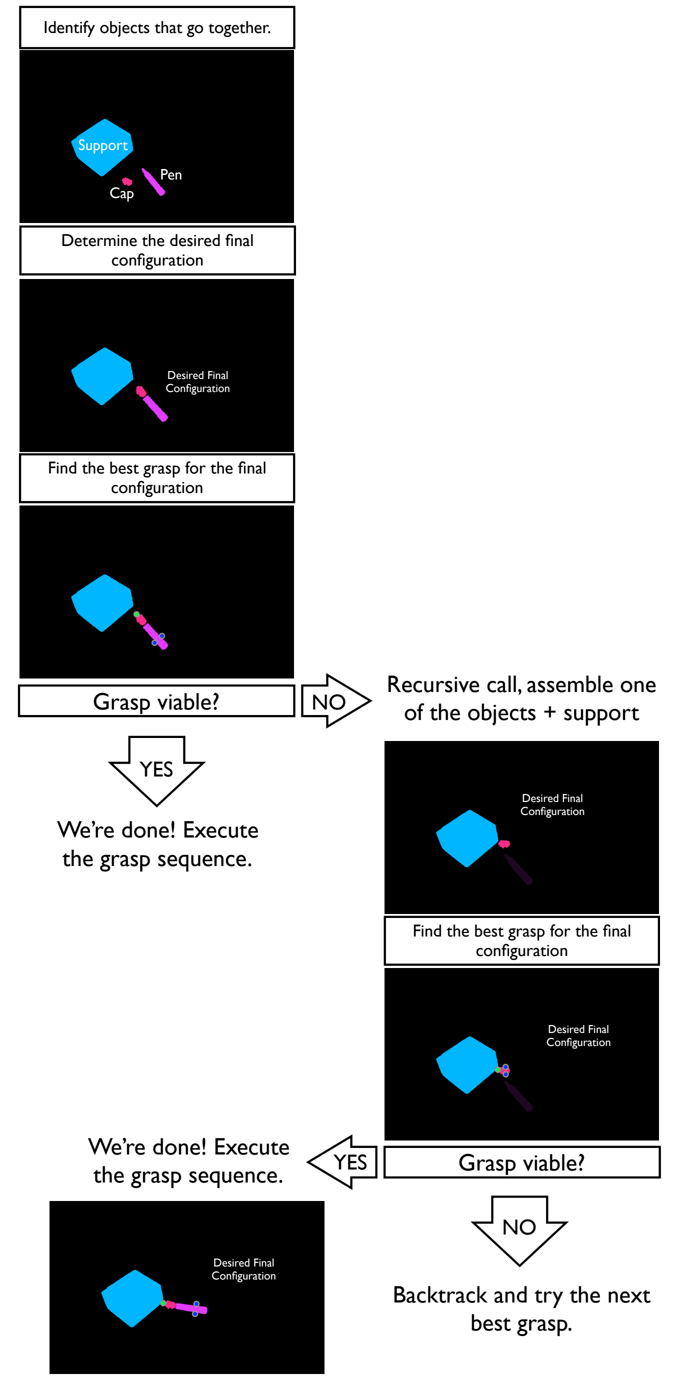

Identify objects that go together.
Support
Pen
Cap
Determine the desired final configuration
Desired Final Configuration
Find the best grasp for the final configuration
Grasp viable?
NO
Recursive call, assemble one of the objects + support
YES
We're done! Execute the grasp sequence.
Desired Final Configuration
Find the best grasp for the final configuration
Desired Final Configuration
We're done! Execute the grasp sequence.
YES
Grasp viable?
NO
Desired Final Configuration
Backtrack and try the next best grasp.

Figure 4: Diagram of the execution of the recursive grasp planning.

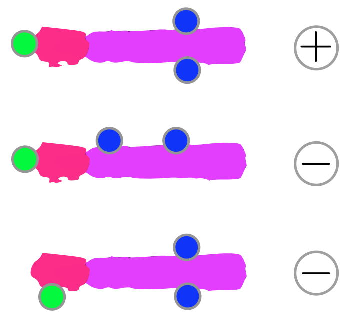


Figure 5: Examples of good and bad grasps.

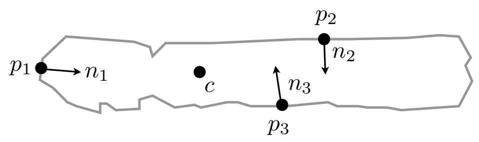
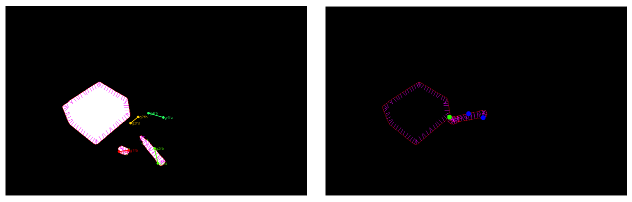


Figure 6: Example of grasp and variables used to compute feature vector.



(a) Grasping sequence. (b) Final configuration.

Figure 7: Example of successful grasping sequence.