6D Object Pose Estimation and 2D Object Outline Extraction for Robotic Grasping

Daniel Sawyer
Computer Science
University of South Florida
danielsawyer@mail.usf.edu

Tian Tan
Mechanical Engineering
University of South Florida
tiantan@mail.usf.edu

Abstract

In order for a robot to dynamically grasp an object, it requires a multitude of different algorithms working in unison for this to be achieved. From all the literature we reviewed, there is not a one-shot system in place that meets all of those requirements. Briefly, the requirements are as follows: First, you must detect an object and localize it in the image. Then you must run semantic segmentation on that localization in order to get the label and mask. After calibrating the image sensors, you then run the RBGD images along with the mask, localization, and label through a second neural network which will return the 6-DOF pose that can be used to grasp the object. Finally, there is grasp refinement which requires a camera mounted on the hand of the robot in order to fine-tune the grasp for more difficult objects such as spheres and asymmetric objects. We use Detectron[4] based on Mask R-CNN[5] for the first network and then feed the data into DenseFusion 6-DOF PoseNet[9] for a single-system integration of all these methods. For future work, we plan to design a single network that can take an RGB or RGBD image in one end and return localization, semantic segmentation, and 6-DOF Pose in a single shot.

1. Introduction

Individuals with diminished physical capabilities must often rely on assistants to perform Activities of Daily Living (ADL). Comparing to having a human assistant, assistive robots are more affordable and using assistive robot makes the user feel they have retrieved capabilities to live by themselves rather than have to live to depend on other people. However, controlling a blind robot to perform activities of daily living is challenging even for healthy users. The solution for this is to give the robot as much autonomy as possible and simplify the control as much as possible. In this project, we are focusing on building a simple hand-eye coordination system that enables the robot to see the scene and pick up a recognized object in the scene while the user only needs to specify the object of interest. More specifically, the vision system will have two components, one is the object pose estimation (OPE) system which infers the object location and orientation in the scene and leads the robot to pre-grasp pose, another is the vision for grasp tuning (VGT) system which provides visual cues for self-adjustment in final grasping. The OPE system takes the RGBD image of the scene in front of the robot as input. The RGBD image will be collected from Microsoft Kinect which is fixed to the robot base. Due to the limitation of depth camera in the grasping stage, where the object is very close to the robot hand, only RGB image will be used for grasp tunning. This RGB image will be collected from an RGB-camera fixed to the robot hand. The hardware set-up is shown in Figure 1.

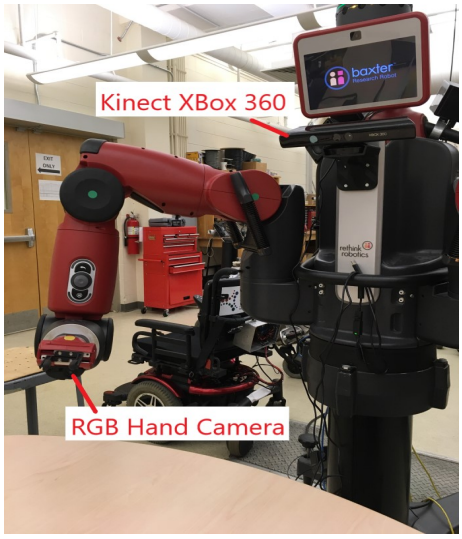


Figure 1. Hardware set-up

2. Background

In this project, we are building a robotic hand-eye coordination system to perform object grasping. The most pop-

ular approach to solve this problem is the end-to-end learning approach like[8]. This paper presents a neural network directly outputs the most likely successful grasp based on monocular images. It has a good grasp success rate but is expensive to train the network and the user has no control over which object to pick. Unlike this type of approach, our method uses two vision systems for reaching the pre-grasp pose and executing grasp separately. One vision system is the 6D object pose estimation system(OPE) which is implemented based on the following work. DenseFusion[9] is a 6-DOF pose estimation network based off of PoseNet and PointNet. Its input requires semantic segmentation and object localization. For these, we use Detectron[4] which consists of a network that is the combination of Faster R-CNN[7] and Mask R-CNN[5]. These are very both very fast networks by today's standard and we should be able to process this data in real time. The other system is the vision for grasp tuning system(VGT) which extract the object outline using Detectron/Mask R-CNN [5]and Canny edge detection[6].

3. Approach

The base framework of our project is the Robotic Operating System(ROS)[3]. It is a communication system that manages the communications between distributed nodes via the anonymous publish/subscribe mechanism. We use ROS to pass camera messages to the vision system and then deliver the output to the robot control system. For controlling the robot we use Moveit![2] an open source motion planning framework. It enables us to plan collision-aware motion with a given robot end-effector goal pose, the pre-grasp pose in our case. From pre-grasp to grasp the robot is controlled using the robot SDK functions. Before we can calculate pre-grasp pose from the recognized object 6D pose and its shape we need to first calibrate our cameras. The intrinsic parameters of the cameras are found in ROS camera messages which are given by the manufacturer. To get extrinsic transformation between the Kinect and the robot we use the Kinect extrinsic calibration from Emarolab[1]. Using this method we can establish the transformation between the camera and the robot base with only one pose-fixed planar pattern as shown in Figure 2. The calibration pattern is composed of three QR code we used the left-top corner one labeled as ar_marker_8. The calibration procedure is to get the 6D pose of the QR code relative to each camera, then with the given transformation from the hand camera to the robot base, the transformation between Kinect and the robot base can be established.

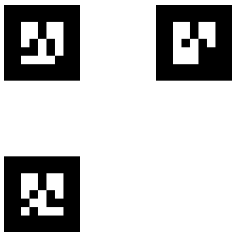


Figure 2. Kinect camera extrinsic calibration pattern

The DenseFusion[9] CNN uses RGBD data along with semantic segmentation masks in order to estimate the 6-DOF pose of an object. The architecture of the network can be seen in the Figure 3. It takes quite a bit of work to get to that point and includes three separate pipes. We are using Detectron with the newest implementation of Mask R-CNN for our first pipe, which runs pretty close to real time. The Detectron network runs an HTTP server that allows incoming POST and GET requests with the network loaded on the GPU awaiting an image in order to run inference on it. Since the network is already loaded on the GPU, it allows for inference to be very fast. Once the POST has completed, it runs the RGB image through the network, saves the data to a zip file and returns the HTTP address for that zip file in the POST request. The second pipe is the DenseFusion network which is also running an HTTP server and has its network loaded on another GPU waiting for POST requests which send the RGBD data and returns a string with the 6D pose estimation for each object that was detected from the first pipe. The final pipe runs from the robot which takes the RGBD images from the kinect or realsense, sends them to the DenseFusion network, receives the 6-DOF pose and moves the arm into the pre-grasp position.

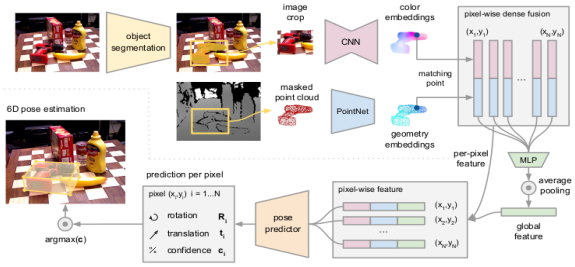


Figure 3. DenseFustion Network Architecture

At the grasping stage, the hand camera is used to guide the gripper to grasp at an appropriate location on the object. Our approach to accomplishing this is to find the 2D object outline and map the gripper into the same image, then we can predict and evaluate the contact region based on which we adjust the grasping motion. To detect the object outline, using only edge detection there will be too many background edges and some object outline will not be detected due to background color similarity as shown in Figure 4.

To solve this issue, we used Detectron/Mask R-CNN[5] to detect the object and get the image mask of that object as shown in Figure 5 then we perform edge detection on the masked image to get the object outline.

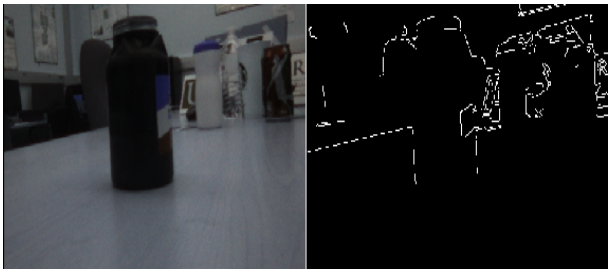


Figure 4. edge detection result without object recognition

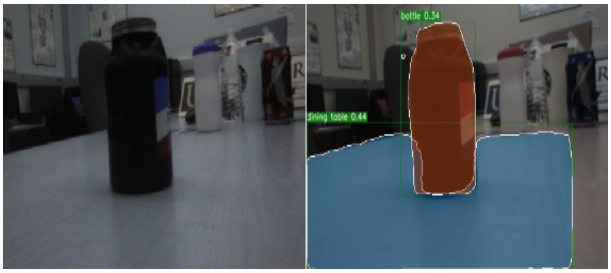


Figure 5. object detection and masking

Since the gripper pose is known to the camera and the camera intrinsic parameters are given, we can predict where the gripper will be in the image. The output of this vision system is then an image of object outline and the gripper grasp rectangle as shown in Figure 6. With this image, we are able to tune the gripper translation in x and y-direction and rotation about the z-axis(roll). Here frame-XYZ is the gripper frame, z is the forward direction, x is left and right, y is up and down. To adjust the gripper position in the y-direction we first perform edge normal analysis to get the normal at each edge pixel as shown in Figure 7 and Figure 8. The sum of edge normals in the contact region(edges in the grasp path) equals to zero when it is a valid contact region. Since the outline detection will have some error in practice, instead of choosing the zero-sum region we are looking for the minimum squared sum of the normals. To optimize the z-axis rotation of the gripper we minimize the angle between the normal of each gripper plate and the average normal of each side of the contact region. To tune the gripper x position we minimize the sum of vectors from the gripper center to the edge points of the contact region. In conclusion, the above grasp tuning procedure can be formulated as the following:

$$\triangle y = arg\min_{\triangle y} \parallel \sum_{i \in CR} \hat{n}_i \parallel^2$$

$$\triangle \theta = arg\min_{\triangle \theta} (\alpha_{k+1} + \beta_{k+1})$$

$$\triangle x = arg\min_{\triangle x} \sum_{i \in CR} (e_i - c_{k+1})$$

$$y_{k+1} = y_k + \triangle y$$

$$x_{k+1} = x_k + \triangle x$$

$$\theta_{k+1} = \theta_k + \triangle \theta$$

Where x_{k+1}, y_{k+1} and θ_{k+1} are the new x, y and roll of the gripper relative to the current gripper frame. α_i is the angle between left contact region normal and the left gripper plate normal and β_i is the angle on the right side. e_i is the position of the i_{th} pixel and c_{k+1} is the new gripper center position.

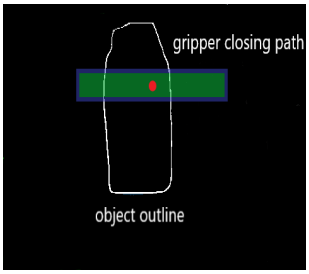


Figure 6. the outline of the object to be grasped and the gripper grasp rectangle

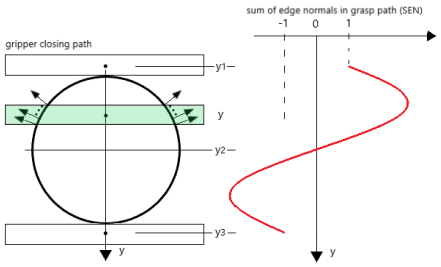


Figure 7. gripper y direction adjustment impact on the sum of edge normals in the grasp path

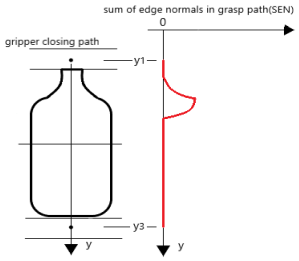


Figure 8. the sum of edge normals of bottle shaped outline vs. gripper y adjustment

4. Results

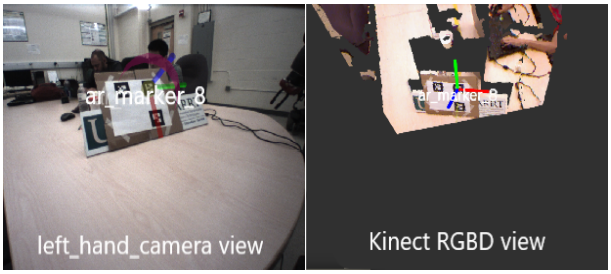
The intrinsic parameters of the two cameras are given by the manufacturer, K_h, K_r and K_d are intrinsic matrices of hand camera of the robot, rgb camera of Kinect and depth camera of Kinect.

$$K_h = \begin{pmatrix} 403.8373 & 0.0 & 603.6554 \\ 0.0 & 405.0408 & 390.1896 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$$

$$K_r = \begin{pmatrix} 525.0 & 0.0 & 319.5 \\ 0.0 & 525.0 & 239.5 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$$

$$K_d = \begin{pmatrix} 575.8157 & 0.0 & 319.5 \\ 0.0 & 575.8157 & 239.5 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$$

In order for a robot to infer the location of objects recognized by the camera, the camera has to be calibrated relative to the robot. The hand camera is a built-in camera of the robot, therefore its pose relative to the robot is known. Using the approach mentioned previously we did the Kinect camera extrinsic calibration. The results are shown in Figure 9 and Figure 10.



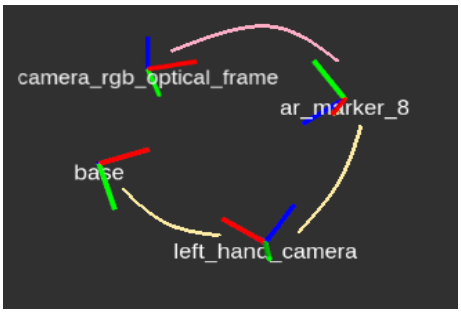


Figure 10. frame transformation visualization in extrinsic calibration

The following equation is used to calculate the transformation from robot base frame to the kinect camera frame ${}^{R}_{K}T$.

$${}^{R}_{K}T = {}^{R}_{H}T\,{}^{H}_{A}T\,{}^{A}_{K}T$$

Where ${}^{R}_{H}T$ is the transformation from robot base frame to the hand camera frame, ${}^{H}_{A}T$ is the transformation from hand camera to ar_marker_8, ${}^{A}_{K}T$ is the transformation from ar_marker_8 to Kinect.

For the Object Pose Estimation (OPE), we were unable to test any real world results due to time constraints. We were able to test it on the YCB Dataset with very good results. The real time integrated 6-DOF OPE system should be in a prototype state of real world implementation on the robot by the end of this week. We will have plenty of real world data results by the time the final report is due.

References

[1] Kinect extrinsic calibration. https://github.com/EmaroLab/.

[2] Moveit! https://moveit.ros.org/.

[3] Robotic operating system. http://www.ros.org/.

[4] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He. Detectron. `https://github.com/facebookresearch/detectron`, 2018.

[5] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[6] C. J. A computational approach to edge detection. 1986.

[7] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):11371149, Jun 2017.

[8] A. K. J. I. Sergey L., Peter P. and D. Q. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. 2018.

[9] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. 2019.