Detecting Position and Orientation of Objects for Robotic Manipulation using Computer Vision:

Final Project Report

Morgan Kuphal & Ashvin Pidaparti

CSCI 5551: Robotics

Professor Junaed Sattar

University of Minnesota - Twin Cities

12 May 2020

Abstract

While robots are able to automate many repetitive human tasks, they are fundamentally limited in their ability to interact with dynamic environments. The goal of this project is to overcome this limitation such that a robot could grab blocks of various shapes, placed at random locations and orientations within its workspace, and stack them at a single position. In order to perform this task autonomously, a robotic arm captures and interprets a digital image of its workspace. This image can then be processed using computer vision algorithms to determine each object's position and orientation in reference to some common frame between the digital image and the robot's real world workspace. Using this data, the transform from each detected object to the real world location of each object can be calculated. Finally, the poses the robot must assume to complete the desired object manipulation can be calculated and the robot can execute the manipulation procedure. Due to unforeseen circumstances limiting access to a real robot, this project was simulated on the UR5 collaborative robotic arm using the RoboDK simulation software. This report discusses how this is achieved and details real world applications and potential improvements of the computer vision techniques employed.

Introduction

Robots are easily able to perform repetitive tasks without manual input, but face limitations when compared to humans in dynamic environments. This barrier prevents robots from being used in repetitive tasks to which they might otherwise be well suited, such as sorting objects which appear at random locations and orientations in a workspace. While humans are easily able to visually identify objects at various locations and orientations, this task is more complex for a robot. Robots must process image data captured by digital cameras in order to determine the locations and orientations of each object. Processing this visual data into usable information is a nontrivial programming task that requires multiple computer vision techniques and algorithms. In this project, all computer vision algorithms were implemented in Python 3.8 with limited use of existing computer vision code libraries.

Unlike a human's intrinsic ability to determine how the body must move in order to grab an object after visually locating it, a robot must be programmed to calculate the transform between the visual data and the real world. After each object is detected, these image locations must be mapped to real world locations that can then be used to calculate the joint poses the robotic arm must assume to grab each object. This is achieved by defining a common frame of reference between the digital image and the real world at a fixed and easily detectable feature in the digital image. Then, offsets from this frame and another easily detectable fixed feature in the image can be measured. Since the real world distance between these easily detectable features is known, the ratio between image pixels and real world distance can be calculated. Once this ratio is known, the translational transform between the common frame of reference and each object can be calculated such that the robot can determine how it must move to perform the desired manipulation.

The main focus of this report is discussing the computer vision algorithms that were used to detect each object and calculate the transforms from the shared frame of reference to each object location. In addition, the RoboDK simulation script is discussed, which uses the location and orientation data to simulate grabbing each object that appears in the robot's workspace and stacking them at a defined position. Finally, real world applications and potential improvements of these computer vision techniques are discussed.

Due to the lack of access to the physical UR5 robot, this project was simulated. In order to simulate a workspace, a black board was used with three white squares outlining the intended workspace of the robot. These white squares against the black background constituted easily detectable features which could be used to establish a common frame of reference and the pixel to distance ratio between the image of the workspace and the real world. Randomly placed square and rectangular blocks were used as sample objects to be grabbed and stacked by the robot. Images of this simulated workspace with the two different types of objects that were used are displayed below (*figures 1 & 2*).

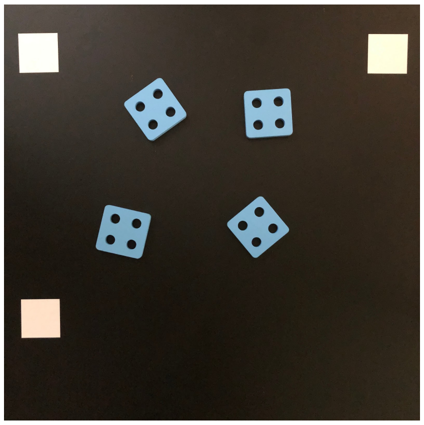
Figure 1: Square Blocks

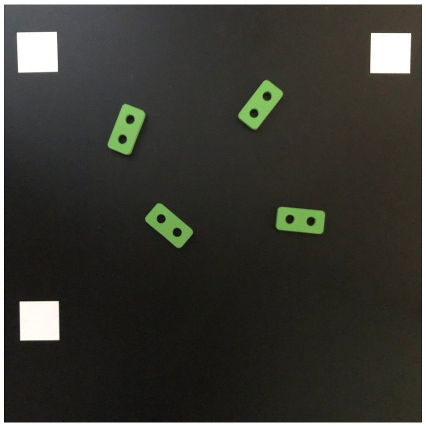
Figure 2: Rectangular Blocks

Methodology

First, the image of the workspace is converted from red, green and blue (RGB) representation into hue, saturation and value (HSV) representation. This means that instead of each channel in the digital image matrix representing the amount of red, green, or blue light at each pixel, each channel represents the color, color intensity, and overall brightness. By analysing each of these three channels separately, different features in the image become more pronounced and easier to detect. The procedure for converting from RGB to HSV values was implemented using pseudocode written by Max Agoston (Agoston, 2005). A sample HSV conversion on the square blocks in the workspace image is shown below (*figures 3, 4 & 5*).

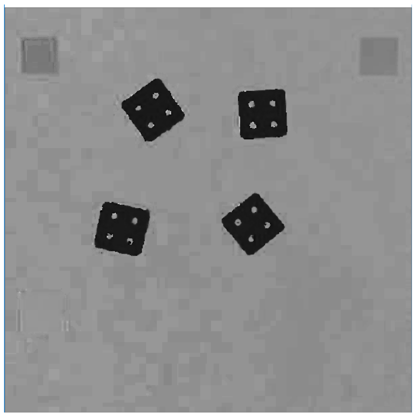
Figure 3: Hue Channel

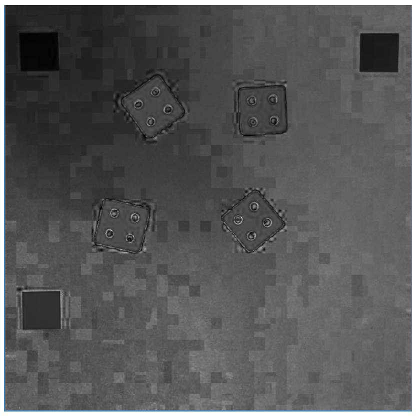
Figure 4: Saturation Channel

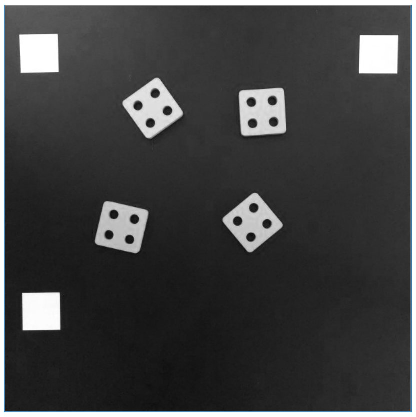
Figure 5: Value Channel

In order to detect the objects, the hue channel is thresholded to produce a binary image that has only white pixels where the hue value is within a certain threshold and all other pixels are set to black (*figure 6*). In order to detect the white squares to establish a common frame of reference and pixel to distance ratio, the value channel is used and the image is thresholded to produce a binary image that contains only the white squares while all other pixels are set to black (*figure 7*).

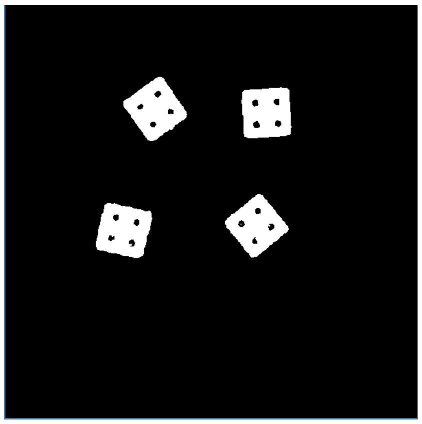
Figure 6: Square Blocks Binary Image

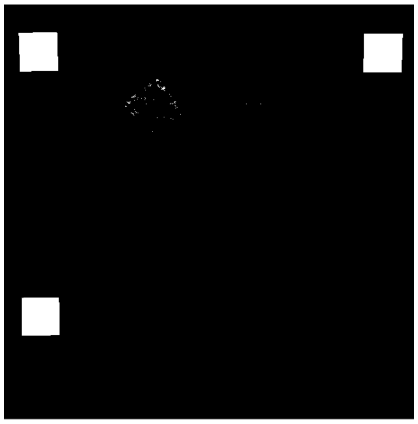
Figure 7: White Squares Binary Image

Next, a blob detection algorithm was developed to extract blobs from the binary images. In this project, a blob is defined as a list of adjacent pixel locations which represent a single feature or object. In order to generate this list of pixels which make up a blob, the blob detection algorithm crawls the image from left to right and top to bottom until reaching a white pixel. Upon reaching a white pixel, the algorithm will crawl around the blob to all adjacent white pixels which make up the perimeter of the blob and append each coordinate location to a list of perimeter coordinates (*figure 8*). Using the list of perimeter coordinates, a list of all the coordinates contained within the perimeter can then be generated *(figure 9*). Any blobs below a certain size are ignored to prevent misinterpreting image noise caused by imperfect image thresholding as blobs.

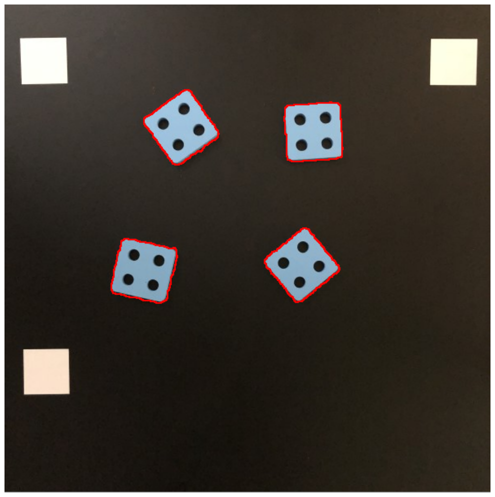
Figure 8: Square Blocks Blob Perimeters

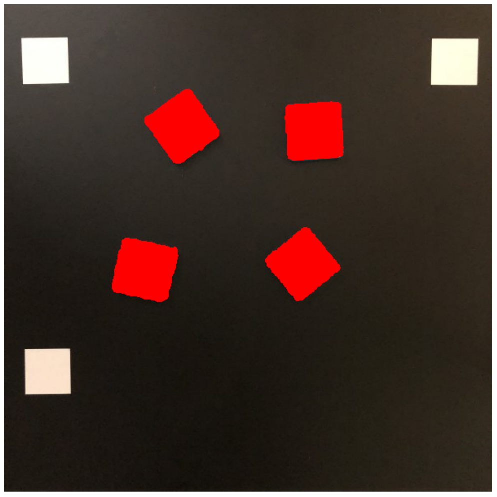
Figure 9: Square Blocks Blob Coordinates

Upon determining the location of a blob, the blob orientation must be calculated such that the robot gripper can be rotated so it can grab the corresponding object and lift it securely. The orientation of a rectangular object is known to be the angle of the line segment connecting two adjacent corners of the rectangle. Using the Harris Corner Detection algorithm, the corners of a blob can be identified and the object's orientation can be determined. The Harris Corner Detection algorithm consists of convolving the subject image with the Soble operator in the X and Y directions to extract the edges in the X and Y directions after a Gaussian blur is applied to mitigate image noise (Sánchez, 2018). These two datasets are then combined according to equation 1 as shown below. Finally, each pixel is assigned a score equal to a free variable *k* multiplied by the trace of the moment matrix *M* subtracted from the determinant of the moment matrix as seen in equation 2 (Sánchez, 2018). These scores are normalized, and each pixel above a given threshold is accepted as a corner.

$$\text{Equation 1:} \quad E(u,\ v) = \sum_{x,y} w(x,\ y)[\ I(x+u,\ y+v) - I(x,\ y)\ ]^2$$

$$\text{Equation 2:} \quad R = \lambda_1\lambda_2 - k * (\lambda_1 + \lambda_2)^2 = det(M) - k * tr(M)^2$$

In this implementation of the Harris Corner Detection algorithm, pixels within a distance of 25 pixels of another corner were rejected. To produce corners for individual blobs, the region of interest was assigned to be the bounding rectangle of the blob in question, and this process was repeated for each individual blob. Once the corners have been located, two adjacent corners must be extracted from that list. This was accomplished by assigning two pairs of corners, calculating the angle of the segments connecting each pair, and comparing the angles for similarity; if the angles were within ± 10 degrees, it could be concluded that the corners in each pair were adjacent. Shown below is the accurately located and ordered corners used to draw outlines around each of the objects in the workspace (*figure 10*). To orient the gripper appropriately, the angle of the segment connecting one pair of adjacent corners was returned.

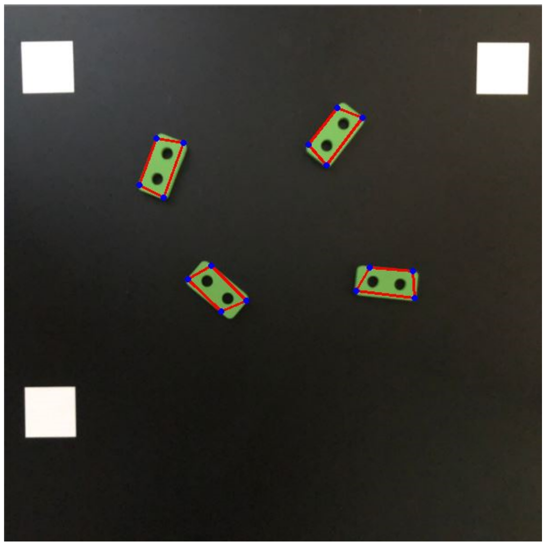

Figure 10: Rectangular Block Corners

After extracting all the object location and orientation data, it is saved to a JSON file such that the data can be transferred to the RoboDK simulation. A script implemented in Python is used to command the model of the UR5 robot. The script calculates the required joint poses for the robotic arm to grab each object by translating the prescribed X and Y distance and rotating about the Z axis in relation to the common frame of reference between the image and the real world. The script then commands the robot to move to a pose above each object, move linearly down to each object, close the gripper, move up, and place the object in a stack at the origin of the common frame of reference. The script commands the robot to stack the objects by moving to a height 9 cm higher each time a block is placed on the stack which is equal to the known block height.

Results

While the results of this simulation seem promising and the program appears to operate as expected, a true evaluation is not possible without a real world trial. It is difficult to gauge if the procedures described in this project provide accurate enough position data for the robot to properly grab and stack objects. An attempt was made to measure the actual locations and orientations of each object in the robot's workspace and compare to the locations and orientations generated by the computer vision algorithms. Every object center appeared to be at the right distance from the common frame of reference when measured using a tape measure, however, the measuring technique used had an approximate error of about ± 0.5 centimeters (cm). It is not known whether an accuracy of ± 0.5 cm would cause the robot to operate in an unexpected way.

In addition to being unable to fully evaluate the accuracy of the results, there may be other unforeseen hurdles in implementing this procedure on the actual UR5 robot. Some of these difficulties may be caused by encountering joint singularities when moving between poses, collisions with the environment, or inability for the gripper to properly grip the objects. Extensive troubleshooting is likely required for a real world implementation of the procedure described in this report.

Conclusion

As the design and development of novel robots progresses, their use cases in dynamic environments must also progress. Through implementing computer vision algorithms to determine the location and orientation of objects, the simulated UR5 robot appeared to successfully grip and stack those randomly placed objects. This success has significant real world implications, ranging from enabling automated warehouse operations to automating manufacturing lines dealing with parts of varying sizes in unpredictable locations.

Despite the apparent success of the computer vision techniques used, there are several ways this project could be improved and extended. As can be seen in the image showing the corners after detection (*figure 10*), it's clear that there is some error in the location of the corners. In future iterations, it would be preferable to use another, more robust method of corner detection. A method using the object perimeter as a convex hull has significant potential. In this method, each consecutive pair of points in the perimeter would have a line segment drawn between them, and, for each segment, the segment would be aligned with the X axis, rotating the blob (Chen, 2009). A bounding rectangle would be calculated around the new perimeter and the minimum of each of those bounding rectangles would be extracted, along with the angle at which the shape was rotated to to extract the bounding rectangle with the smallest area. Because the minimum area bounding rectangle will be aligned with the sides of a rectangular object, the angle of the rectangle will be the angle at which the gripper should be rotated to grab the object. In addition, applying the Sobel operator in the X and Y direction individually yields an edge detector in the respective direction (Chen, 2009). This edge detection method has significant potential to make contour detection more efficient because applying this edge detection method and thresholding would yield a binary image which only displays the edges of objects or features in the original image.

Works Cited

Agoston, Max K. (2005). Computer Graphics and Geometric Modeling: Implementation and Algorithms. London: Springer. pp. 300–306. ISBN 978-1-85233-818-3.

Chen, Jie, et al. "The Comparison and Application of Corner Detection Algorithms." *Journal of Multimedia*, vol. 4, no. 6, 2009, doi:10.4304/jmm.4.6.435-441.

Sánchez, Javier, et al. "An Analysis and Implementation of the Harris Corner Detector." *Image Processing On Line*, vol. 8, 2018, pp. 305–328., doi:10.5201/ipol.2018.229.