CS4758: Robot Construction Worker

Alycia Gailey, biomedical engineering, graduate student: asg47@cornell.edu
Alex Slover, computer science, junior: ais46@cornell.edu

***Abstract:* Progress has been made in the design and development of robots that can perform tasks that would be repetitive and time-consuming for humans. Examples include sweeping floors, stacking items, and moving items from one location to another in a factory setting. Another such challenge would be assembling building materials into a specified architectural design, which involves challenges of localization and classification. We aim to teach a robot how to recognize the presence of toy blocks scattered about on the ground at random locations, and then stack those blocks into a specific structure specified by the user. To detect and locate blocks we use both the RGB visual and point cloud data from a Kinect device. Various clustering algorithms are used to determine the exact positions of the blocks. Finally, we input the structural design from a text file in the form of a set of 3D coordinates at which to place each block, all coordinates being relative to a point of origin. Our experimental results show that the robot is able to not only detect the 3D location of blocks, but also distinguish larger blocks from smaller blocks and build the design we specify. We conclude that robotic construction has considerable potential to assemble even complex designs without human assistance.**

Keywords: clustering, construction, machine vision

I. Introduction

Programming robots to perform time-consuming and menial tasks has great rewards, freeing time for humans to work on more sophisticated tasks which do not involve difficult labor. However, for a robot to move objects from one location to another, the robot must not only recognize the location of objects, but also be able to distinguish one type of object from another. For example, if a robot must move books from a desk to a bookshelf, the person may not want the robot to grab a cereal box and place it on the shelf with the books, or worse yet, a cup of coffee. The ability to distinguish one type of object from another is important and challenging. Another challenge is in programming the robot to know where in 3D space to place the objects. We address both of these challenges in this paper.

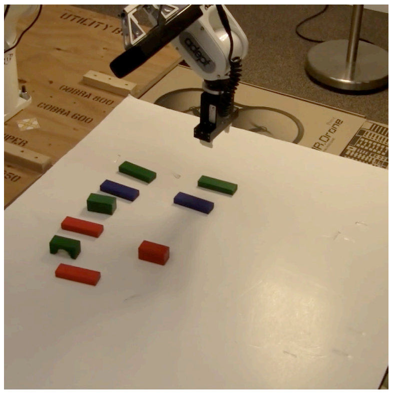
Figure 1: Initial position of blocks, after localization. For clarity, the images in this paper use the construction on a white background, though this isn't necessary.

We place toy blocks on the ground below a Kinect camera and collect point-cloud image data of the blocks. The Kinect gives both point cloud data indicating coordinates of the blocks in 3-space and RGB visual data, with a 640x480 resolution. The RGB and depth values of each pixel are sufficient to determine whether or not that pixel belongs to a block. Clustering algorithms are used to group pixels together according to which block they belong to. From here, the average X- and Y- values can be taken to find the block's center, as this can be assumed to be the mean of all values, since the blocks are rectangular. In addition, the size of each cluster determines the type of block (small, medium, or large).

Once the robot knows the location and size of each block, it reads a set of build instructions from a text file. Each instruction has configuration-space coordinates (position relative to a point of origin, and rotation), and which type of block should be used.

This is sufficient to define a structure, even in practical settings: users might specify that they prefer larger (and presumably heavier) items on the bottom, and the flexibility means that almost any structure composed of blocks can be built.

II. Related Work

Having robots move objects from one location to another is a well-understood application, with a great deal of existing work and commercial implementations. However, much of this work takes place in factory-like settings where the type and initial position of the object is known exactly. More recent work has focused on combining these placement utilities with computer vision and classification algorithms that do not require a specific object to be placed at a known position. This is the work we are hoping to extend.

Machine vision is one of the most heavily studied areas in robotics because of how crucial it is for a robot to perceive its environment in order to perform tasks without human assistance.

We aim to address the challenge commonly faced in which a robot is programmed to look among multiple objects on the ground for a specific object of interest and place that object into a structure of a predefined design.

III. Approach

For a robot to be able to take a collection of scattered objects in front of it and build a specific structure out of them, it must have multiple capabilities.

1) Detect the 3D location of the objects amidst a background
2) Distinguish one type of object from another. For example, classify objects by size and color
3) Know an object's orientation
4) Know where on the structure to place the next object
5) Know how to place an object onto the structure of the correct orientation

Visual and Point Cloud Data Analysis

In the past, addressing the first challenge would have required an expensive LIDAR scanner, or inferring depth from 2D images. The former approach requires unwieldy and expensive equipment, the latter has accuracy problems despite a great deal of effort to make it feasible. Fortunately, the Kinect sensor is able to provide both visual and depth data accurately and cheaply. For each point in its 640x480 grid, the Kinect provides both 3-space coordinates relative to itself, and RGB values from 0 to 255. For our application, the blocks have bright colors which are easily distinguishable from the background in either RGB or HSV space, though with the variety of color segmentation algorithms available this is not strictly necessary. The point cloud data gives the 3D location of objects directly, while the visual data gives pixel values, which can be matched to their corresponding point in the point cloud to find their position.

The second challenge is addressed by analyzing the visual data. Because we know the colors of the blocks to be used in advance, and know that there are only four types, we use a simple threshold-cutoff algorithm for classifying a pixel by type based on its RGB values as explained in Table 1. This effectively segments the image and greatly simplifies the clustering step.

Table 1: Threshold R, G, B values for recognition of pixels belonging to red, green, blue and yellow blocks. By default, pixels are classified as background pixels.

Pixel Classification	Red	Green	Blue
Red block	>100	<100	<100
Green block	<100	>R	<90
Blue block	<100	<100	>70
Yellow block	>G	>100	<100

On the initial pass, pixels are not always assigned the correct color because of noise, smearing, and light glare. Thus, a median filter is applied to the segmented image, where each pixel is given the most common RGB classification of its neighbors. Figure 1 shows the image processing steps where the leftmost image is the original and the middle image is the output of the median filter after color reassignment. Even after the median filter is applied, there are small segments of color due to noise, these are removed by filtering out all clusters below a certain size.

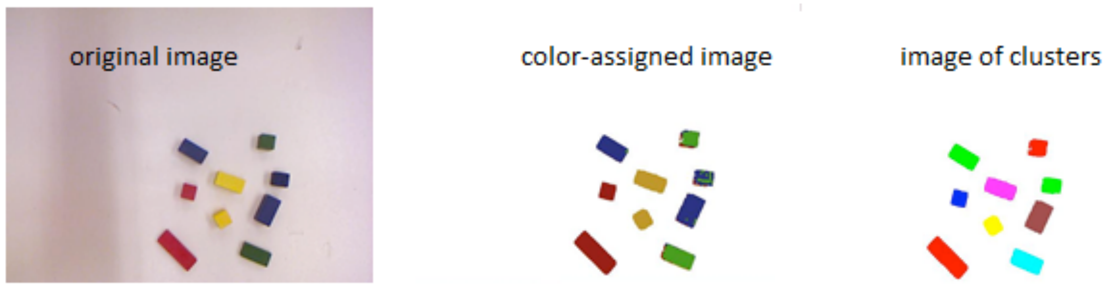

Figure 2: (left) The original image taken by the Kinect camera. (middle) The image created by reassigning RGB values to each pixel depending on whatever color classification the RGB values of each pixel most closely resembles. For example, pixels with RGB values most closely resembling red are assigned a value of pure red. (right) This image represents which pixels belong to which cluster. The red pixels all below to the cluster #1; the green pixels all belong to cluster #2 and so on

Clustering

Once pixels are assigned a color or classified as background, they must be grouped into clusters such that each cluster belongs to a different block. A matrix is made representing which cluster each pixel belongs to. Each entry is 0 for a background pixel, and nonzero if the pixel belongs to a cluster where the nonzero value represents the cluster number. To assign a cluster number to each pixel, a nested loop examines each pixel and ignores ones belonging to the background. When the first colored pixel is found, it is assigned a value of 1. When another colored pixel is found, we check if it is adjacent to the pixel that already is assigned a nonzero value. If so, then it must be within the same cluster, so it is assigned the same nonzero. If it is not adjacent, then it is assigned a new cluster value, and so on for each pixel.

This alone is not sufficient to assign all pixels to the correct clusters because when scanning left-to-right and top-to-bottom, as a pixel that belongs to the same block but in a completely different part of the block may be assigned a completely different nonzero value. Therefore, after the first loop assigns nonzero values to all the colored pixels, an inner loop is run repeatedly to ensure that pixels have the same cluster value as their neighbors, as defined by a distance metric. As this is run repeatedly, the clusters grow larger, and it is run until convergence.

At this point, the clusters have been assigned, and a final pass is made to eliminate clusters below a certain size as noise. An example of the results can be seen on the right of Figure 1. Once the pixels are all classified into the appropriate cluster, the size and location of each cluster is computed, with a nested loop that counts the number of pixels in each cluster, and determines the midpoint from the average of all coordinates (in both directions), which is acceptable because the blocks are rectangular.

We attempt to also determine the orientation of the blocks from the visual data, which becomes especially important when the width and length of the object are very different, because the robot should not grab an object along the longest direction. We attempted to determine orientation of the blocks with respect to the horizontal axis from the visual data by finding the top-right and bottom-right sides of a block (the max-X, max-Y and max-X, min-Y points of a cluster) and extracting the angle from the cotangent of the difference in coordinates, but accuracy was too poor to be of any real use, probably because the noise-filtering and segmentation techniques distort the true position of the corners too much.

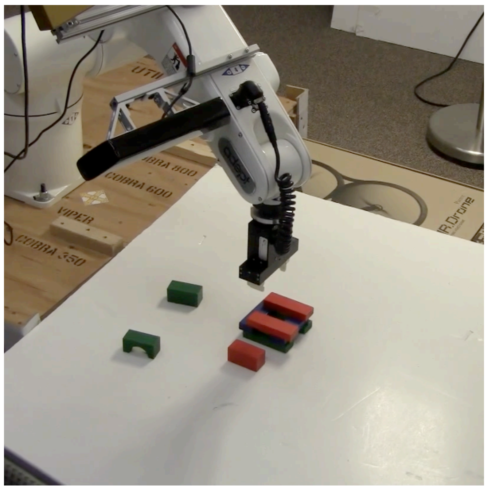
Figure 3: The partially completed structure. The robot knew to use large blocks for the initial steps and small blocks for the next ones.

For the point cloud, we use the PCL library to collect and manipulate the data. After removing the floor plane and additional noise from the point cloud with PCL primitives, with is left is a set of disjoint regions, one for each block. We wish to find the center of each of these blocks, which is very amenable to using k-means clustering.

Initially, we attempted to guess the number of clusters from the data using the 'gap statistic' method proposed by Tibshirani, but results were poor: when blocks where placed close together and at the same orientation, precision dropped to around 80%, even when weighting was applied to only consider the (local) X and Y axes. We speculate this may be due to the fact that, because the blocks are often rectangular, the optimal weight for the three axes varies depending on the blocks orientation- if the long axis is along X, Y must be weighted more heavily so that two adjacent rectangular blocks are not treated as a square. A two-pass method where the orientation of blocks was calculated first and then used to tune the weights for finding the gap statistic may have been able to solve this problem, but ultimately we chose to use the number of clusters from the visual data as the input, since it was so reliable.

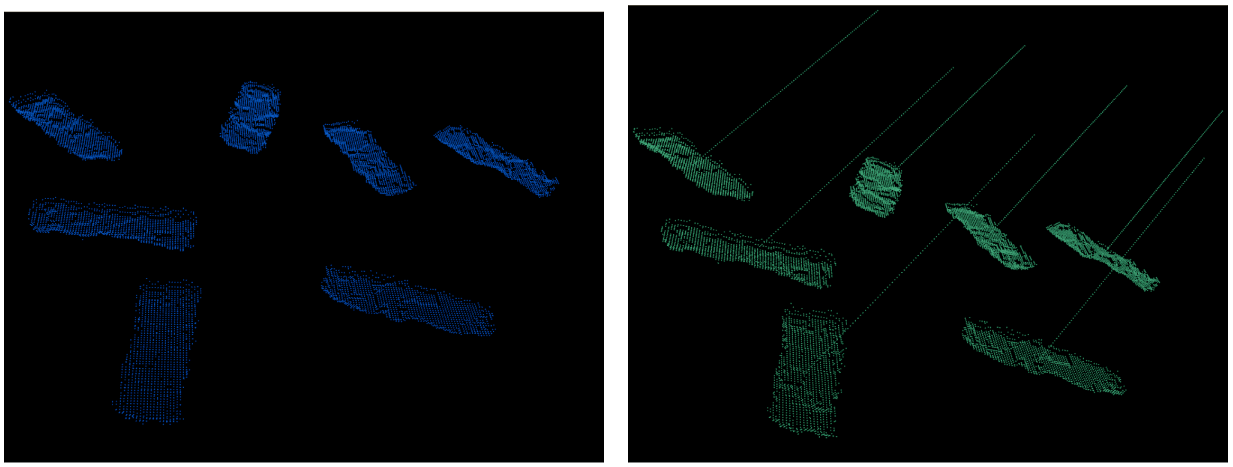
Figure 4: (left) The point cloud data of the blocks after the plane of the floor has been removed. (right) The same data with the positions of the clusters, as determined by k-means, clearly marked (zoom in to see)

Because k-means automatically finds the centroids of the clusters, these points are just transformed to global coordinates and combined with the visual data to determine the true positions. The two data sources agree to within 0.1 mm most of the time, so we abandoned plans to use linear regression to decide how to weight them, and just use naïve averaging.

Classification of Blocks by Size

In this simplified setting, there are three different sizes of blocks, whose sizes, both in absolute terms and in terms of the approximate number of pixels they occupy in the visual data, is known exactly. Thus, a sophisticated learning algorithm for classifying blocks by size is not needed, and the size of the cluster with threshold values for small, medium, and large blocks is used instead.

Importing the Architectural Design

Code already exists for the Viper manipulator arm to grab and place objects at specified global coordinates. As such, after we determine positions of the blocks in global space it is trivial to instruct the robot to grab them. The architectural design is coded as a series of 3D destination coordinates, orientation values, and the size of the desired block. The raw format of the data is:

X,Y,Z,Yaw,Pitch,Roll,Size {1, 2, 3}

The Process of Building a Structure

The robot is moved to a "home position" and a snapshot with point cloud data from the Kinect is taken. From this, the robot localizes the positions of all the blocks, using the combination of image segmentation and k-means described above. It then reads through the set of build instructions one at a time, finding a block of the correct size, picking it up, and placing it at the specified position and orientation. If a block of the correct size cannot be found, it will abort the construction process.

IV. Perception Results

Much of the evaluation of the efficacy of our algorithms at locating blocks was done manually because it was difficult to get hard data as to the accuracy of the system's guesses about position, since this would require measuring the exact position of the blocks with respect to the robot. As a result, most of our tests involved comparing the pixel values of the guesses from the clustering algorithm to human estimations of the true center of the blocks, and noting how close the robot moved to the "correct" position. The k-means and color segmentation code usually agreed to within 0.1 mm.

V. Experimental Results

The goal of our experiments is to how well we can teach a robot to put together different architectural designs. Figure 2 shows an illustration of the robot arm we used in the Saxena lab. There are six degrees of freedom, five of which are joints. The joint j1 near the base allows the whole robot arm to turn left or right. Three other joints j2, j3, j5 allow the robot to reach up, down, far away or up close. J6 is the grasper, which opens and close while j4 determines the rotational orientation of the robot hand when it grasps and places objects. The Saxena lab has already written code for manipulating the joint angles in order for the robot to grasp and place at the desired location and orientation. We simply specified the location and orientation in our code after converting Kinect image coordinates to global coordinates, which are in units of mm with the base of the robot as the point of origin.

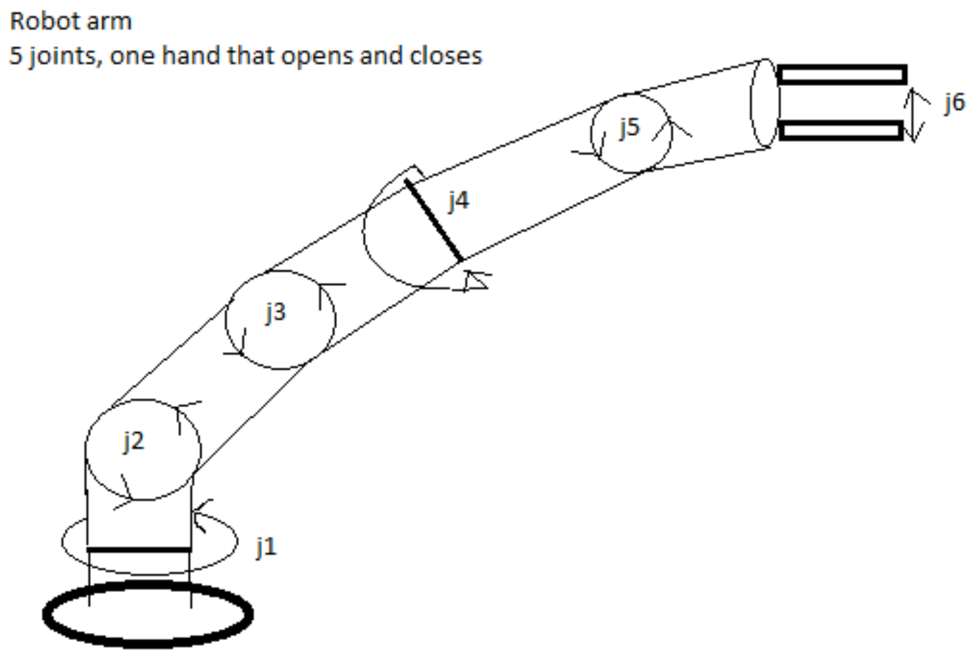

Figure 5: Diagram of the robot arm. J1, j2, j3, j4, j5 are the joints that allow the robot to move its hand towards objects. J6 is the 6th degree of freedom that allows the robot to grasp an object by opening and closing. As shown, j1 allows the arm to turn itself left to right. J2, j3, j5 allow the robot to bend downwards or upwards, and also determine whether the robot reaches far away or up close. J4 determines the orientation at which the robot hand grasps or places objects.

Although we are able to successfully instruct the robot to pick up blocks at random locations and build into a predefined architectural structure, we often found that the 3D coordinates were not always completely accurate in allowing the robot to neatly stack the blocks at the precise location that we intended relative to the other blocks. In particular, blocks vary in thickness so that if a block is thicker than the other blocks, we need an offset value for the z coordinate at which to place the block so that the block is not pressed into the other blocks below. Because of these setbacks, we had to do test runs a few times before we got the 3D locations and offsets just right.

Because it is difficult to input the exact correct spatial coordinates for every single block without making some mistakes, error is less likely to occur when the robot can plan for itself the proper spatial coordinates at which to place the next block. Rather than input exact 3D destination coordinates, allow the robot to view the thickness of a block based on the depth values of the block itself relative to the immediate surrounding background. The depth gradient between the block and its surroundings would give the thickness of the block, and thus indicate the z offset necessary when placing the block onto the structure.

Some limitations of our system are in training the robot to distinguish between blocks and other objects of a similar size and color. The environment is often cluttered, and the more clutter there is, the larger the number of clusters initially detected. Processing a large number of detected clusters in a cluttered setting can be computationally expensive for our algorithm. More sophisticated forms of object recognition would be important when placing the robot into a cluttered setting.

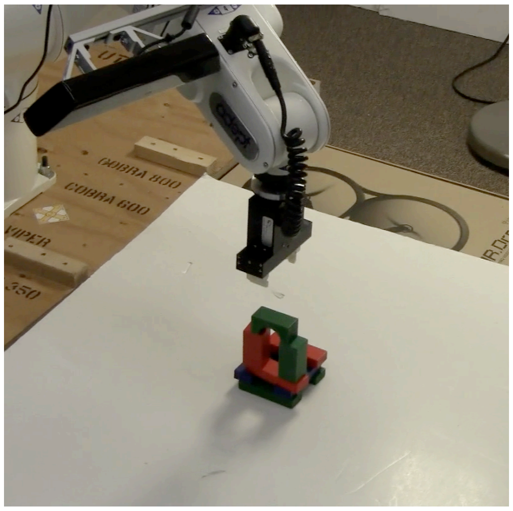
Figure 6: The completed structure

VI. Conclusion

In this paper, we have addressed the challenge of determining the location of simple-shaped objects randomly scattered in front of the robot. In a practical setting, these algorithms would allow a robot to take some randomly scattered objects and stack them into a structure specified by the user without the objects having to be placed at a predefined location. In addition, we have trained our robot to recognize which types objects to place at specific parts of a structure. Interesting to note is that we did not have any extremely complex learning algorithms in our project and were still able to build structures reliably, indicating that this field has even more potential if more powerful algorithms are used. Even if the number of each type of object is unknown, the robot can at least distinguish one type of object from another in case it must choose to place one type of object onto the stack before placing other types of objects on top. The ability of a robot to stack objects of different shapes and sizes at randomly scattered locations, and into a predefined structure, can be useful for many settings.

REFERENCES

Jiang Y, Zheng C, Lim M and Saxena A (2011) “Learning to Place New Objects” *Cornell University Technical Report*

Kemp CC, Edsinger A and Torres-Jara E “Challenges for Robot Manipulation in Human Environments” *IEEE Robotics and Automation Magazine* March 2007

Lozano-P´erez T, Jones J, Mazer E, and O’Donnell P (2002) “Task-level planning of pick-and-place robot motions” *Computer* 22(3): 21-29

Tibshirani R, Walther G and Hastie T “Estimating the number of clusters in a dataset via the Gap statistic” *JRSSB* March 2000