**Michael Williams, 211087798, mw1992**

**Andrew Lau, 212905253, Pellost**

**November 21 2016**

**EECS4422**

MID-WAY PROGRESS REPORT

FOR JENGA STACKING SYSTEM

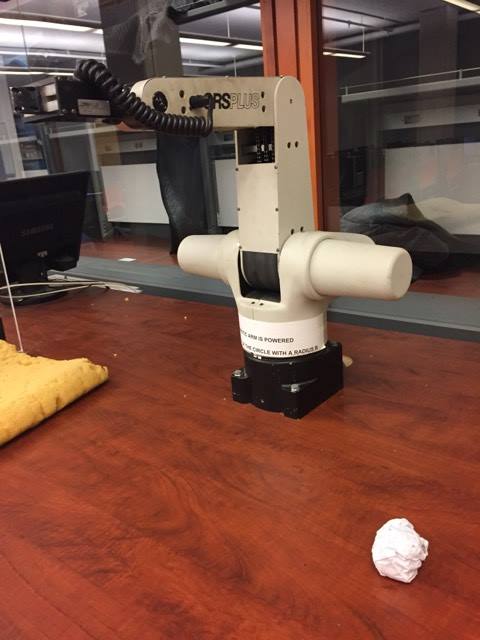
***INTRODUCTION***

This progress report focuses on the critical components of the Jenga stacking system described in the project proposal and discusses what has been completed as well as what is left to implement to complete the project. All code for the completed component is at the end of the lab, in the appendix.

***PROGRESS***

The first of the components completed is the component for moving and picking up objects in the plane by means of the robotic arm. The movement of the robotic arm was programmed using the inverse kinematic formulas derived in the first assignment, as a starting point or template. We altered this code, which originally moved the robot arm to any reachable point in a two dimensional plane, to form a program that moves to a point attainable in the three dimensional plane instead. The exact methodology is shown in the appendix but basically we assign a new parameter for the z-axis. This parameter, given by the user, is used in combination with the adjustments made to the kinematic formulas which allow us to accurately place the object at a point one the 3d plane. Ultimately, we are taking an object, given its location on a two dimensional plane, and moving it to a point on a three dimensional plane.

Below are diagrams to show the robotic arm at work. The first depicts the object we wish to pick up with the robotic arm alongside the actual arm being used. The object to be picked up Is a regular piece of paper crushed into a ball.



This next images shows that, just as in the previous assignment, the arm can move to a specified space and pick up the given object at that space.



The last image below shows that, given a new position on a three dimensional plane, the arm can accurately move to and drop the arm at that location. As explained below, this does not consistently work as of yet.

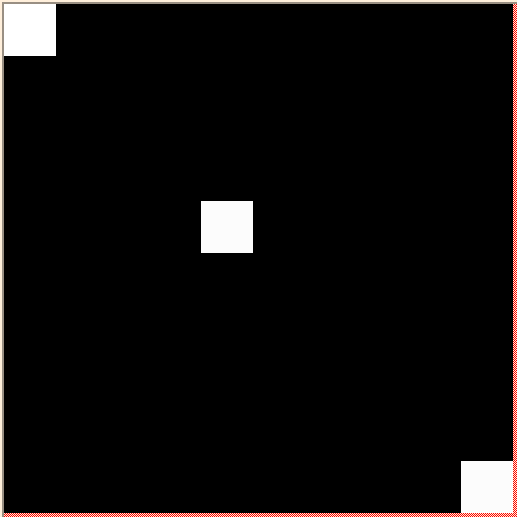
With this critical component complete, we can effectively move an object to some reachable points on a 3d plane, given the values. However it is worth noting that the algorithm doesn’t yet work all the time. Throughout testing, errors occurred when the arm was used multiple times. The next step is to review the kinematics used and find a proper method that ensures correct transformations each and every time. Further steps in utilizing this are discussed further below in the “Next Steps” section.

The second component completed is the component for detecting multiple objects on an image and distinguishing the individual objects, for individual use. The purpose of this is simple: The code needs some way of distinguishing the individual blocks so that they can be separately dealt with. By this I am of course referring to separately moving blocks to their destination. This specific component only deals with separating the individual objects for later use and showing that they can be separately dealt with. This is primarily because the methodology still needs to account for the angles at which the blocks may be laid and the scaling function of the actual hand-eye coordination between the camera and the robotic arm, components that have yet to be implemented until a later date. The methodology for programming this functionality is as such: we use a depth-first-search algorithm to find, given any point of the object, the additional points or pixels corresponding to the object. We take the points found by the depth-first-search algorithm and store them into a container. Thus, we effectively have all the pixels corresponding to the individual object. With this, we can then place the pixels onto a blank image and have the object alone on the image plane. The idea is to then take these and use them to find out the exact coordinates the robotic arm must move to, in order to pick the object up.

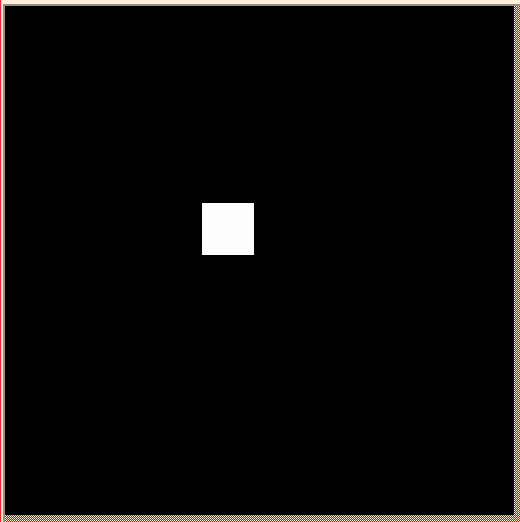
Implementing this method was not without its problems and required building subcomponents as well, due to software limitations in our implementation. Initially, the depth-first-search algorithm was recursion based, but because of the limitation to the amount of recursive calls that can be made in mediamath, alternatives had to be taken. This involved implementing a stack, along with the functionality of the stack (ie the push and pop methods). This was done using 8 bit arrays. However, this still limits the amount of pixels we can store. Therefore, multiple conjoined stacks were necessary to accurately hold object info of appropriate size. The problem with this is that the runtime takes considerably longer as a result.

Once these subcomponents were built, our software iterated through a black and white image until a white pixel, denoting a pixel of the object, was found and from there the depth first search algorithm was applied. The returned object was displayed separately to show that we have a means to individually look at and consider objects in the plane. This process was done for all objects in the image created.

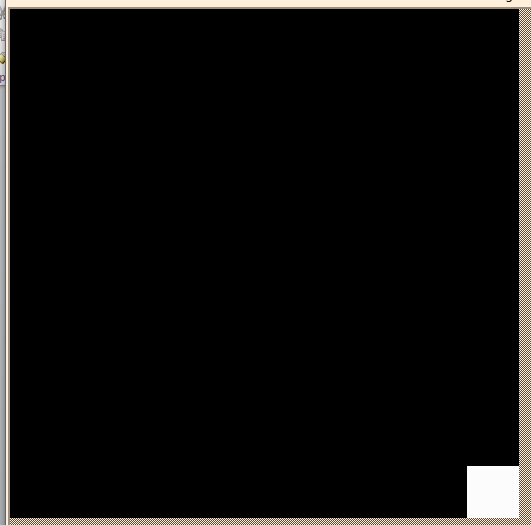
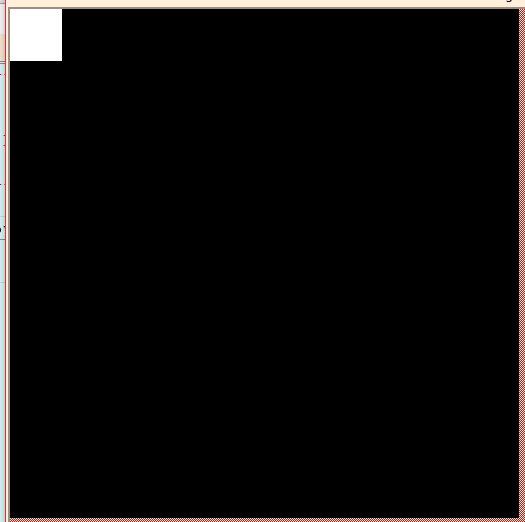
Below are images to show how the process works. First, we create a black and white image containing white blocks on a black background.



Given this image, we then iterate through the image and perform depth first search on the object once we have located one of its pixels. The image below depicts the first individual object found, after it has been extracted and isolated into its own separate image.



Below are the remaining blocks, separated into different images. This shows that we can separately analyze and use the object information and that we can effectively deal with multiple objects.



**NEXT STEPS**

With these components created, there are still some steps that need to be implemented, that build off of these critical components. In terms of the 2 components, the first thing that needs to be done is comparing images of the actual plane with the jenga blocks, in order to acquire a good idea of how many stacks will be necessary to hold individual blocks for processing. After this, the next most important step is fixing the kinematic equations to consistently work with the robotic arm transformations, each and every time. Then, we need to figure out how to properly implement the hand eye coordination, so that we can transform the image coordinates of the block into something the robot arm can use. By this I am referring to computing the orientation and dimensions of the objects and scaling them so that the robotic arm can pick them up. After we have this, we will next need to combine the two components. Doing so should be relatively simple and will leave us with a method for finding objects with a camera and picking them up with the robotic arm. Then all that is needed is a final component for moving the Jenga blocks to a specified area and stacking them.