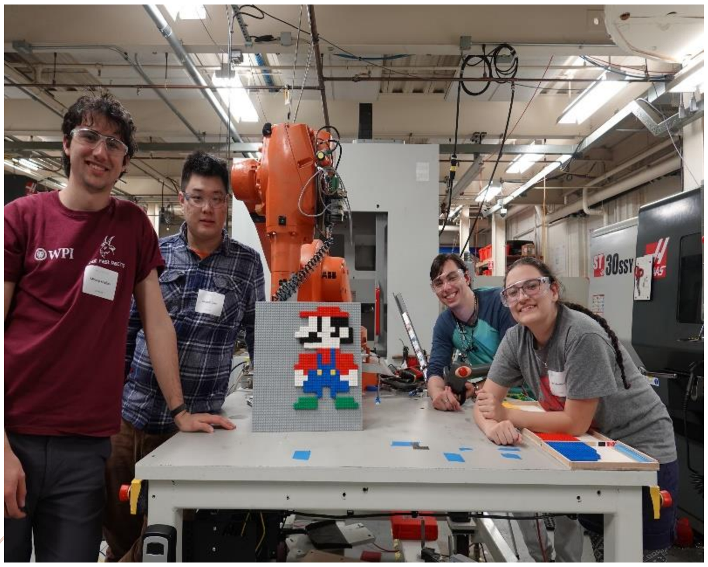
12/13/2019

Team 1 Final Project Report

RBE 4815 B19



Michael Abadjiev, Qingyuan Chen, Maricella Ramirez, Orion Strickland

TEAM 1

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

This project required us to apply everything we have been learning in class about industrial robots to accomplish a specific goal of our choosing. We used RobotStudio to paint a picture using LEGO bricks. This required a MATLAB image parser to communicate with RobotStudio and send the image information. The robot was then able to use a venturi-powered suction gripper to pick and place pre-sorted LEGOs based on their color into the image.

GOAL STATEMENT

The goal of this project will be to create functionality with the ABB robot to "paint" a 2-dimensional picture using LEGO blocks. 2x2 LEGO pieces will represent colored pixels and be placed in the appropriate locations by the robot arm. This will present significant challenges in both software and hardware, as the pieces will need to be oriented in order to properly connect to a base plate and picked based on color for placement.

TASK SPECIFICATIONS

In order to make this project achievable for us, some constraints were created to define a problem that could be solved in the space of 1 term. First, it was decided to restrict image size to 32x32 pixels. This was done primarily to reduce the number of LEGO blocks needed, as even 32x32 results in needing 1024 blocks. Additionally, it was decided to restrict the number of colors in the image to 6 - 5 block colors and 1 background color, from the base plate. MATLAB would be used to parse images, scaling them and reducing the number of colors, and then sending the image information to RobotStudio for the robot to execute. In order to simplify the gripper design, we decided to use 2x2 LEGO bricks, rather than 1x1 bricks, which would be very difficult to pick up. Finally, it was decided to use pre-sorted LEGO bricks, rather than adding a camera for picking specific colors.

WORKSPACE SETUP

A hopper that contains 5 separate areas for each color of LEGO is aligned at the edge of the table. The pickup point for the first color is the reference point for all other pickup points. Before the RAPID code is run, the reference point is obtained through jogging the arm. The origin point of the base plate must be obtained so the robot can properly place the LEGO pieces.

GRIPPER DESIGN

Two design philosophies were considered for the gripper design - a traditional “finger” gripper, or something using suction. Using a two-finger gripper was quickly identified as impossible, since the fingers on the side of each block would interfere with placing them next to each other. For this reason, we decided to use a suction-based system for grabbing blocks from above. Because LEGOs are designed to stack with each other, it was easy to build this out of a LEGO block. The inside of the piece was hollowed out, as seen in Figure 2, to prevent the pieces from sticking together in the absence of a vacuum. Then a hole was drilled in the top of the piece to accommodate the vacuum line from a venturi vacuum. This way, when the gripper block is placed on top of a block to pick up and the venturi is enabled, they stick together, and then fall apart again as soon as the vacuum is released. This block was simply attached in place of the suction cup on the venturi gripper from lab 4, resulting in the gripper shown below in Figure 1.

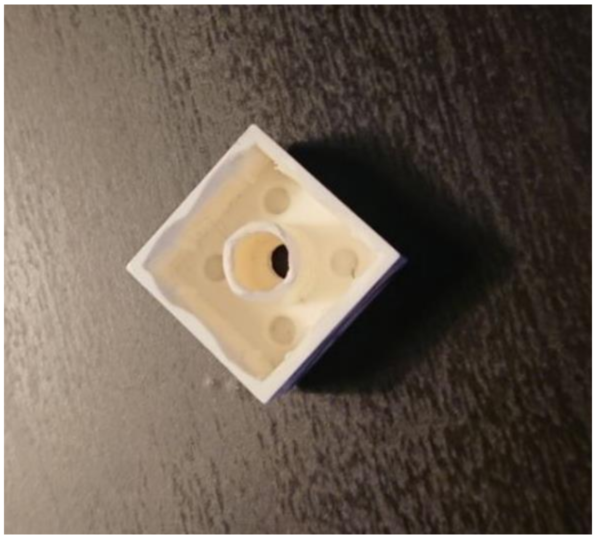


Figure 2: Hollowed out LEGO EOAT

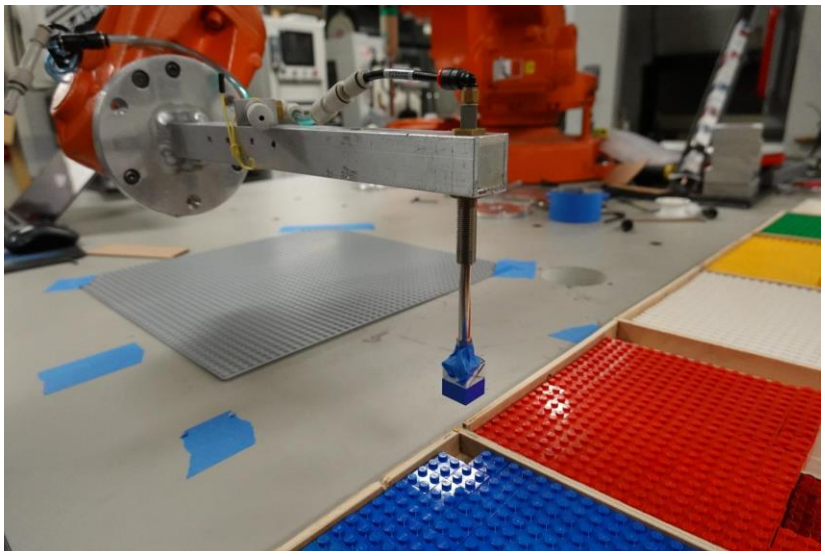


Figure 1: Finalized Gripper

PROGRAMMING

MATLAB IMAGE PARSING

There are 3 steps in parsing an image for robot to build.

First step is to scale the image. Robot’s work area has a size limit while most images are few hundred if not thousands. For a normal image, MATLAB’s built in image scaling function would be enough. This method scales a group of pixels into one by taking the average of the group. However, this method doesn’t work for some pixel art images. These images have one single color square represented by a large group of pixels; the averaging will create a lot of fuzzy regions that have transition colors. Since we have a limited number of colors, the transition colors don’t always get detected correctly. In this case, scaling is done by taking the center point color as the color for this group of pixels. This will keep the sharpness of the image.

Second step is to convert the image into HSV color space to prepare for the next step. RGB is how computers understand color while HSV is now human understand colors. Since we want to find the closest color for each pixel, HSV will be a much better option in this case. The MATLAB has built in HSV RGB conversion functions. However, these functions expect a different number scale than usual 8-bit RGB values. To make sure the available LEGO color in the preset is at the same scale of the image, the preset color will go through the same steps of conversion as the images.

The last step in image processing is to represent each pixel as the index in the hopper. This is done by finding the smallest numerical difference between the given pixel and all the preset colors. A weighted average is taken among the differences in three channels of HSV. The Hue channel is weighted heavier since it is the channel for colors.

MATLAB TO ROBOTSTUDIO SOCKET COMMUNICATION

The RobotStudio Code acts as the server and the MATLAB code is the client. The server and client connect over the same IP between the pc and the ABB controller, “192.168.100.100” on port 160. When the string from image processing is sent to RobotStudio it is converted from raw bytes to a string. The string is then parsed into an array of numbers starting with the column and row values followed by the number indicator for each LEGO piece after.

ROBOTSTUDIO PALLETIZING

The RobotStudio program is set so that the LEGO hopper has a work object set to the center of the first LEGO brick. The other hopper target locations are set in reference to this first work object. The hopper has its own work object and respective target set to the center of the upper right LEGO brick on the base plate as shown in Figure 3.

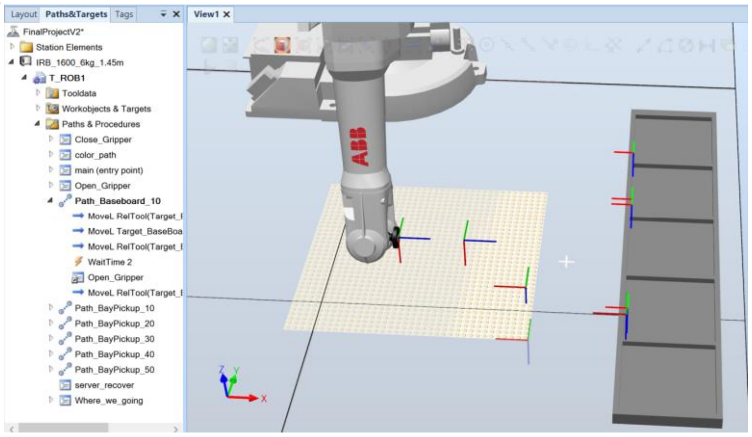


Figure 3: RobotStudio Setup

Since the Robot is in constant communication with the MatLAB socket we have MatLAB calculate which x and y coordinate to send the brick to. Then, we send those coordinates along with the color number of the brick. The robot then moves to the defined targets over the bays to pick up the corresponding colored brick. Then, based on the x and y coordinate given from MatLAB, RobotStudio calculates the relative distance from the base plate target using the RelTool() function to then place the brick at the desired location.

RESULTS AND DISCUSSION

Our robot was successfully able to construct a 2-dimensional picture of the fictional character Mario. The final workspace layout can be seen in Figure 4. The gripper was able to consistently pick up LEGO pieces, if it was positioned correctly. One issue we ran into had to do with inaccuracies in the final hopper. Because it was simply made from manually assembled plywood, the pickup locations for the LEGO pieces were not very precise, leading to the robot sometimes having difficulty aligning itself on top of the pickup location. Additionally, since the pickup LEGO piece was only loosely attached to the venturi, it tended to rotate slightly with respect to the arm, resulting in some failed pickups.

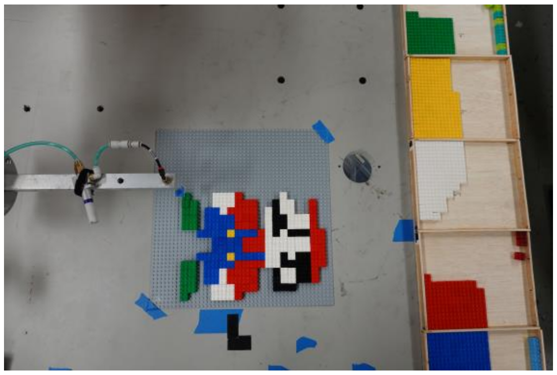


Figure 4: Final Resulting Mario

The scaling of the images will sometimes run into the edge cases of picking up an intermediate color which will result in a small line of discontinuity or wrong color in the image. The only problem with HSV is it represent black and write with full saturation or full value. This means the S and V channels can’t be ignored, and the weight between the 3 channels need to be tuned to get a good result.

RobotStudio has a max buffer size of 1024 which isn’t large enough to process the whole string sent from MATLAB. It was decided to have a continuous open communication and send one point at a time. Although this work around was successful this made the process exponentially longer because of the time needed to send data before placing the next piece. In order to run to program continuously without any pauses in between placing LEGOs the entire string from MATLAB was copied and added into the RobotStudio code. An improvement to the string processing instead of communicating over TCP would be to have MATLAB export the string to a text file and continuously parse the string from the file.

CONCLUSIONS

This project was a good way to apply what we learned in this class to a more real-world application, to accomplish a task of our own choosing. We our knowledge of RobotStudio from this class, as well as some previous knowledge of MATLAB to successfully implement the functionality we desired. While there was some functionality still lacking, like auto feeding the hopper, the basic functionality of painting an image worked well and produced a recognizable picture.

APPENDICES: IMAGES