MECH 472 Mechatronics and Automation Final Report for an Autonomous Robot

By
Alex Boehler 6258743
James Bracken 6251420
Matthew Brownridge 2204126
Maria Papadopoulos 9350845

A report
submitted in partial fulfillment
of the requirements of the MECH 472 Mechatronics
Concordia University
Proffessor: Dr. K. Khorasani
29 April 2015

ABSTRACT

Alex Boehler, James Bracken, Matthew Brownridge, and Maria Papadopoulos

A wheeled mobile robot is to be controlled in real-time based on the ever changing environment via computer vision. The robot is to arrange Lego pieces such that the blocks from the letters “UHI” in a certain color sequence. To achieve this, the robot identifies the location of the Lego pieces and finds the closest block of a specific color. The robot is then programmed to move towards the block, pick it up, turn around, and lay in down in a specific coordinate. This process would repeat this process until the Legos spelled out UHI. The overall process is conducted over approximately 20 minutes with significant accuracy and acceptable precision.

Table of Contents

1 BACKGROUND INFORMATION

Developed by Intel Russia research center, OpenCV is an open-source library of programming functions. The library is cross-platform and therefore possesses functions for various programming languages such as C++ and Python. OpenCV's library main focus is on real-time computer vision [1].

The goal of computer vision is to create a model of the world from images. The computer vision system analyzes the images in order to produce an appropriate behavior. This field of computer science is desirable for control systems which are required to respond in real-time to an ever changing environment [2].

Due to the inherit nature of a robot requiring mobile operation–it was found necessary to make the robot wireless, as this would allow for more maneuverability, including the ability to do full 360 degree turns, without tangling a cable. To accomplish this task, an Arduino microcontroller is used to control the servo motors attached to the robot, and as a way to facilitate wireless communication via Bluetooth.

2 STATEMENT OF THE PROBLEM

The objective of this project is to construct a mobile robot that is able to build a structure which consists of Lego pieces based on real-time feedback from a webcam. The mission is to be completed within the guaranteed time duration of 25 minutes. The workspace consists of a white 3x4 ft^2 board which serves as the background of the image. Lego pieces of various colors and sizes will be randomly distributed on half of this workspace, as illustrated in Figure 1. The robot is to be placed in a random location at the starting region. The robot is to identify and localize Lego pieces with use of the webcam which provides imaging feedback. The robot must construct the three alphabet-shaped structures with Lego pieces in the location specified in Figure 1.

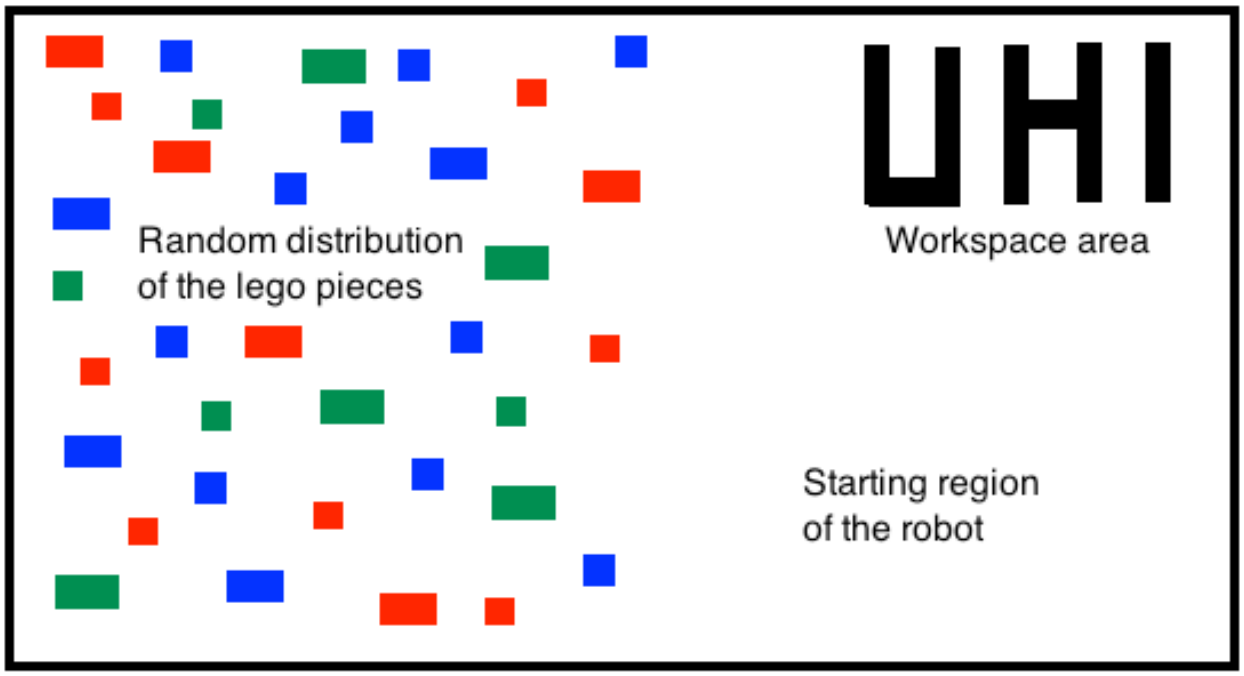

Figure 1 - Workspace layout

The two Lego dimensions available are small 1.6cm(length) x 1.6cm(width) x 0.96cm(height) and large 3.2cm(length) x 1.6cm(width) x 0.96cm(height). The three colors used for this project are blue, red and green. The structures the robot must construct are the letters U, H, and I in the Lego configuration illustrated in Figure 2. The letters must be aligned horizontally in a straight line.

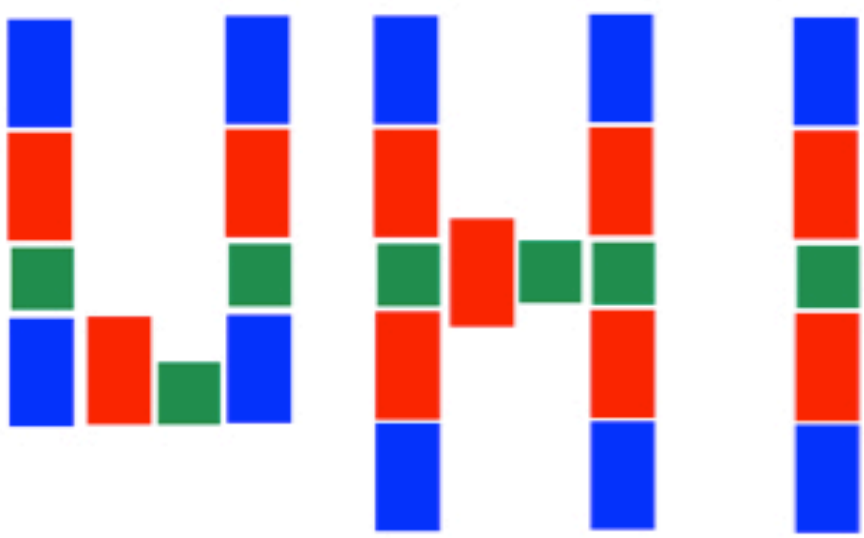

Figure 2 - Configuration of Lego pieces for letters

3 METHODOLOGY

After thorough research on approaches in computer vision, the use of OpenCV library of programming functions for real-time image processing was selected. This selection was based on a number of factors. Although sample code was provided to all the teams in room H-1066, the lack of available information on the system used as well as the overcrowding of students on the four available computers greatly influenced the decision of using OpenCV. Additional factors include the amount of online resources available for OpenCV and the short period time.

The objective of this project is to use computer vision to allow the wheeled mobile robot to accomplish a set the tasks which includes:

1) identifying the location, colors and size of all the Lego pieces;
2) find the closest block of a certain color;
3) move towards the block;
4) pick up the block;
5) move towards the workspace area; and
6) Place the block in a systematic order such that the structure the blocks create yields the letters “UHI”.

These systematic steps create a control system which is event-based scheduling as oppose to time-based scheduling. The reason for this is because each event can only occur once the preceding event has been completed. In other words, each tasks occurs not at a specific time but after the completion of a specific prior event.

4 DESIGN

4.1 Hardware

A list of the specific components with their respective prices is found in Appendix A.

4.1.1 Aluminum Shield

An aluminum shield covers this wheeled mobile robot. The purpose for this is to attach a white paper with two yellow dots of different areas on top of the robot. These yellow circles enable the robot orientation to be identified by the visual program, which will be described later. The small yellow circle is located on the center of rotation of robot to reduce the amount of adjustments needed when the computer vision program instructs the Arduino to rotate clockwise or counter clockwise.

4.1.2 Wheels

Illustrated in Figure 3 is an assembled depiction of the constructed wheeled mobile robot. There are two wheels at the front of the robot, each controlled by separate servo motors. These serve to control the direction and position of the robot. The back wheel is a ball caster to allow for a turning circle based around the two front wheels. Additionally there are two servo motors: one to control the height of the clamp and the other to open and close the clamp. The base of the robot has been 3D printed from a one off design made to incorporate all the features required to mount the components and keep its footprint to a minimum. This includes grooves for the servo motors, holes for mounting points, tabs to space the circuit board for mounting and a housing for the battery back at the back of the robot to counteract the weight of the servo motors.

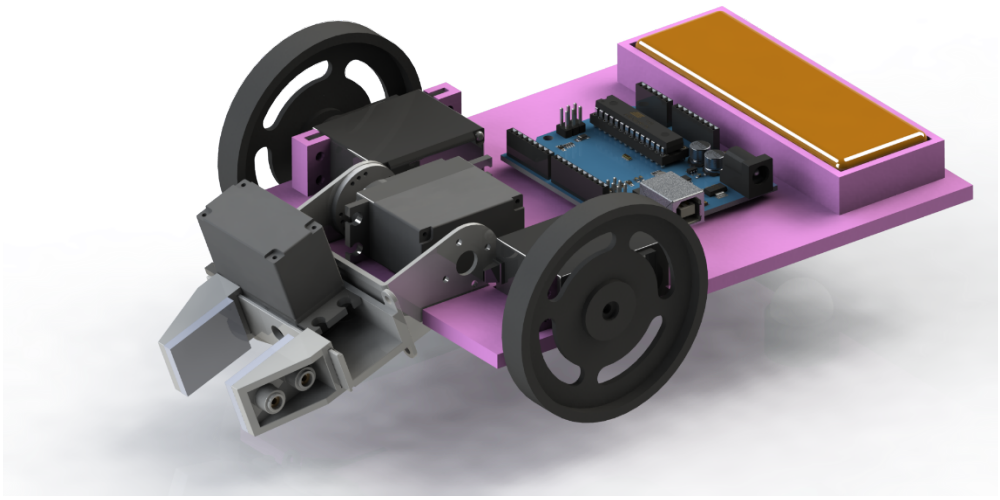

Figure 3 - Solidworks model of wheeled mobile robot

An Arduino has been incorporated into the design along with a Bluetooth shield. For the Arduino to receive commands from the main program: the computer outputs its signal through Bluetooth as it would through a serial port, this information is received at the Bluetooth shield and operates the servo's as desired. Bluetooth was incorporated to reduce additional forces and friction caused by an attached cable. Using a cable could restrict the movement of the vehicle, accidently hit blocks and confuse the visual system. The Bluetooth system is initially more work to setup and program however it proves beneficial for testing and reducing movement restrictions.

4.1.3 Power Source

Because the system is wireless the robot requires its own internal power source to operate, as previously mentioned a battery is incorporated in the design. The battery chosen to match the

Arduino is a 7.4V battery, however the servo motors are suggested to run between 4.8-6V, therefore a small 5V voltage regulator was implemented to supply power to the servo's.

4.1.4 Wheel Blocker

Due to the type of driving wheels and their high level of traction, when confronted with a Lego block the robot had a tendency to climb over or flip surrounding blocks when driving around. To solve this problem a wheel blocker was installed that pushes blocks that are in the way to prevent them from being run over.

4.1.5 Gripper Modification

Initially the scope chosen for of our project only included the use of small Lego blocks due to the inability of the unmodified claw to grab the larger blocks reliably. To account for this problem later in the project a gripper extension was made, this acts as a funnel for larger blocks and re-orients them to fit in the claw.

4.2 Software

The nature of the project concerns event-based scheduling. Thus the order of the tasks and their completion is of importance from a programming standpoint. The complete program can be found in Appendices B and C for the main.cpp and Arduino.ino codes, respectively. Due to the linearity of the system's behavior, no additional controls, such as PID, are required. Moreover, the light design of the wheeled mobile robot allows for the robot to climb over the Legos without moving them significantly. The ability of the claw to be raised and lowered aids in the avoidance

of moving the undesired blocks significantly. This allowed for the program to be simpler in terms of omitting functions responsible for obstacle avoidance and path finding.

4.2.1 Bluetooth

The program has been organized into two main classes. The `bluetooth` class mainly just organizes what is required for serial communication between the main program and the Bluetooth shield which is attached to the Arduino microcontroller. The member functions associated with this class include a constructor and destructor—where the constructor allocates a location in windows memory to use as the serial port transfer, and the destructor closes the file. During the operation of the system, it was found necessary to close and open the serial port file, as it was noticed there was occasional loss of communication with the Bluetooth hardware which required physical intervention. Therefore after ever 5 blocks that have been moved, the Bluetooth closes and re-establishes communication.

4.2.2 Image processing

The `lego_collection` class possesses all the variable and member functions required for image processing and event-based scheduling. Its constructor initializes the variables of the program as well as contains an array of coordinates for the letters UHI. The member function of class `lego_collection` represent the steps necessary to complete the objective of the project.

The `Image_Filtering()` member function filters the original image into three separate threshold variables in accordance to the colors red, blue, green and yellow. This is achieved by first blurring the image via OpenCV's function `blur()` in order to reduce possible noise. Next, the image color scale is changed from red-blue-green (RBG) to hue-saturation-value (HSV)

through the use of the `cvtColor()` function. The HSV image facilitates the process of isolating the different colors whereby each color's range of HSV is defined in the `inRange()` function. The approach of locating the blocks based on color instead of shape is to eliminate the noise caused by shadows when attempting to search for rectangles. The characteristic of this project allows this approach to hold true since the lego pieces are all rectangular thus eliminating the requirement of filtering shapes.

The `Block_locations(Mat MAT_threshold, char debug_name[])` member function searches for blocks of a given color and outputs their location. This is achieved by first inputting the desired color into the function. Next, with use of the threshold image of a given color, the OpenCV function `findContours()` creates counters around all objects in this image. This creates moments and center of mass vectors to all the objects contoured. For each object contoured, the area, defined by OpenCV as a pointer of the contour's moment `Moment.m00`, of the object must be above a certain value to eliminate possible noise. The coordinates of the blocks above this noise threshold are then calculated by dividing the x and y moments by the area of each object. The coordinates are then stored in an array for further analysis in other functions.

The `Robot_location(Mat MAT_threshold, char debug_name[])` member function receives data concerning the location of the two yellow circles which represent the front and back portions of the robot, respectively. These two circles are differentiated in the program by their

areas. Once established which circle represents the front and the back of the robot, their coordinates are calculated and saved in memory for further analysis.

4.2.3 Move robot

In order for the robot to spell out the desired UHI two arrays were created to store the coordinate and color information. Dropoff_array is a 3x27 array which has 1 row of information for each block to be picked up. For each row 3 columns contain the desired color, desired size and block number. The block number is then used in coord_array which is a 2x27 array which contains the x and y coordinates for each dropoff location. The coordinates are organized in such a fashion that the blocks will be dropped off to spell UHI with the desired spacing. As the code iterates through the move robot function it increases the block_number after each dropoff coordinate is reached. Once the block_number is increased the whole process is restarted.

Given the completion of these member functions, the distance between the robot and all the blocks of a specific color are calculated in the `Closest_block_distance(vector<Point2f> Block_Centroid)` member function with the use of the Pythagorean theorem. The function only occurs once before the robot begins moving. This approach reduces the amount of time required for the program to be executed since the terrain will only need to update the location of the robot throughout the steps until the next block needs to be identified and located.

The member function `update_image()` updates the image for the yellow threshold image in order to relocate the wheeled mobile robot. Moreover, the member function

`Update_block_distance(Point Block_Centroid)` recalculates the distance between the robot and the closest block. This is necessary to capture the occurrence of when to stop the advancement of the mobile robot and prepare it to pick up the block.

The `Closest_block_angle()` member function calculates the angle using arctangent between the closest block and the robot's two yellow circles. This is essential in rotating the robot to ensure proper alignment between the robot and the block. If this is not achieved, then the robot would miss the block. Similarly, the `Angle_to_DROPZONE(float xcord, float ycord)` member function concerns orienting the robot with the coordinate whereby the robot will release the block in order to create the letters UHI.

The order and conditions of all the previously mentioned member functions is set in the Move_robot(int block_location) member function. Firstly the Robot_location and the Update_block_distance are called in order to establish the current location of the robot in relation to the closest block.

The status of the claw is then considered. If the claw is open, the robot is programmed to attain a block within its grasp. To do this, the program locates the closest block to the robot and sends a command to the Arduino that orients and moves the robot towards this block. When the robot is within a certain distance of the block, the program sends a command to the Arduino to stop the robot from advancing and then to orient the robot's claw within one degree of the block. This increases the precision of the robot to pick up the block when programmed to do so. Once the claw is closed, the program assumes that the robot is holding a block. The robot orients and moves towards a particular coordinate in the workspace region until it reaches a certain distance

from the desired coordinate. Here, the robot orients itself to be within one degree of the coordinate the block is required to be located. Once the robot has released the block at the desired location, the claws are open which thus begins moving the robot to the next desired color which would then be placed at the next pre-established coordinate.

4.2.4 Main Loop Procedure

The `main()` function activates the webcam. The serial communication sends the command outputted by the `Move_robot` member function to the Arduino via bluetooth. The Arduino has been preprogrammed to behave in certain ways in accordance to the serial input. For example, when the `Move_robot` member function outputs the command `'w'`, the serial communication bring this `char` to the microcontroller which possesses a number of cases, one of which is case `'w'`. The programmed commands available to the Arduino are found in Table 1. Under this case, the microcontroller is programmed to turn both wheel's servo motors forward for a short period of time. Therefore, the communication of the image processing program main.cpp and the servo motor control program Arduino.ino allow for the desired overall behavior of the robot to complete the required tasks.

Table 1– Arduino commands

Command	Function
w	forward
a	left
s	reverse
d	right
t	forward fine
f	left fine
h	reverse fine
y	right fine
i	forward cont.

j	left cont.
k	reverse cont.
l	right cont.
g	grab
r	release
v	arm up
b	arm down
n	claw open
m	claw close
q	detach all

4.2.5 Arduino Code

During operation the Arduino is continuously listening for commands from the serial port. Once a command is received that matches a command stored within the Arduino program, that command is then executed using its own delay and pin out to the appropriate servo motor.

5 ANALYSIS

Considering that the width of the claw is approximately the same size at the width of the large Lego, this imposed a design constraint on the system. To resolve this issues, angled clamps were incorporated into the design of the robot. In addition, the robot was programmed to move left and right to move the Lego in order to grasp the Lego horizontally. This increased the efficiency of picking up the blocks to 90% calculated from experimental trials. The 10% error is caused when the scattered blocks are too close and the claw descends onto this close block. This misses to pick up the desired block thus leading to the placement of an inexistent block at the specified coordinate. Due to the three month time constraint of the project, the program does not incorporate feedback which ensures that the block is within the claw during the pickup phase of the program. This feedback would have ensured 100% efficiency of the program picking up the block.

The resulting structure yields the letters UHI as required. The horizontal blocks placed closed to one another and are nearly perfectly straight. However, the middle blocks of the letters have gaps between them. This was to prevent pushing the previously placed blocks for the letter. The resulting structure 85% of what is considered ideal, as seen in Figure 4.

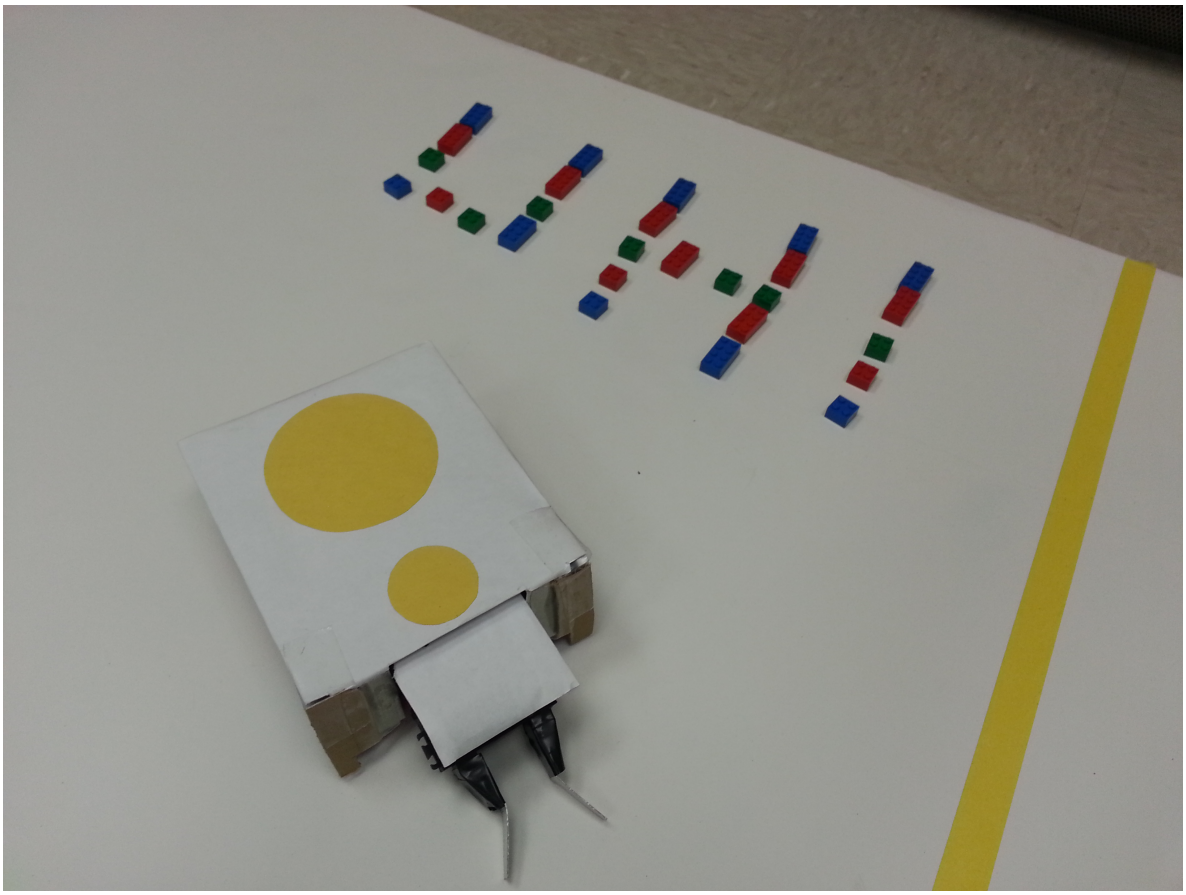

Figure 4 – Result

The overall procedure is deemed successful. The resulting structure is similar to that of the desired structure previously illustrated in Figure 2 with a high success rate over the many trials which have been conducted. The net amount of time is approximately 21 minutes. This is well within the 25 minute limit.

6 EXPERIMTATION AND OBSERVATION

During experimentation of the project, it was found necessary to make adjustments to the methodology and implementation of the hardware and software. One of the largest issues faced with this project was the wireless communication—which had frequent issues the Bluetooth disconnecting after a certain amount of time. To fix this issue, after five blocks have been placed, the Bluetooth will disconnect and then reconnect to flush and buffer that may have been filling up and causing the crash.

During the block placement to ensure alignment, it was found necessary to make the robot go to a “home” location, which would make the robot go to a specific location that would allow the robot to always go place the blocks an angle perpendicular to the letter being spelled; as opposed to picking up the block and going directly to the drop off location. Finally during the pickup and drop off, it was found necessary to add fine tune speed control when a precise angle is needed. Therefore, the servo motors are slowed down during robot angle fine tuning, to allow the robot to accurately position itself.

7 CONCLUSION

To complete this project, OpenCV library is used for its real-time computer vision functions. The OpenCV program communicates with an Arduino which is used for servo control via serial port communication. This communication is achieved with the use of Bluetooth. The objective of this project was met. The program was able to location of the blocks with their respective colors and output a desirable behavior of the wheeled mobile robot which picks up and places blocks in the proper configuration of colors and overall structure of UHI. The overall process was completed within the required 25 minutes.

REFERENCES

[1] "Introduction to Computer Vision with the OpenCV Library," Linux Config, 2007. [Online] Available: http://linuxconfig.org/introduction-to-computer-vision-with-opencv-on-linux

[2] “Machine Vision Background,” Dissertationen Online Der Freien Universitat Berlin. [Online] Available: http://www.diss.fu-berlin.de/diss/servlets/MCRFileNodeServlet/FUDISS_derivate_000000002505/02_Chapter2.pdf;jsessionid=4360B7D4C9EACC62141109EF14180C29?hosts=

[3] Aleksandar Krstikj, "Tracking a ball and rotating camera with OpenCV and Arduino," Aleksandar Krstikj, 2013. [Online] Available: http://aleksandarkrstikj.com/tracking-a-ball-and-rotating-camera-with-opencv-and-arduino/

[4] “The OpenCV Reference Manual,” Release 2.4.9.0 , 2014. [Online] Available: http://docs.opencv.org/opencv2refman.pdf

APPENDIX A: List of components

Component	P/N	Quantity
Lynxmotion Pan and Tilt kit/ Aluminum	RB-Lyn-77	1
Lynxmotion Little grip kit	RB-Rox-01	1
Lynxmotion Little Grip Attachment Kit LGA-KT	RB-Lyn-118	1
HS-422 Servo Motor	RB-HIT-27	5
Lynxmotion Servo Wheel 2.63" x 0.35" (pair)	RB-Lyn-27	1
Lynxmotion Sticky Servo Tire - 2.75" x 0.38" (pair, w/o Rim)	RB-Lyn-439	1
Pololu Ball Caster with 3/4" Plastic Ball	RB-pol-94	1
Arduino Uno USB Microcontroller	RB-ARD-34	1
DFRobot 7.4V Lipo 2200mAh Battery (Arduino Power Jack)	RB-Dfr-160	2
7.4V Lipo Battery Charger	RB-Dfr-607	1
LM78M05CT 5V 0.5A Voltage Regulator	RB-Sha-05	2
Step-up/Step-down voltage regulator S7V7F5	RB-Pol-207	1
Bluetooth Shield for Arduino	RB-lte-12	1
210mm M/M Premium Jumper Wires (30pk)	RB-Dfr-133	1

APPENDIX B: SOFWARE: MAIN.CPP

```
#include <iostream>
#include<sstream>
#include<string>
#include<cmath>
#include<ctime>

#include<opencv\cv.h>
#include<opencv\highgui.h>
#include "opencv2/opencv.hpp"

#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#define UNICODE 1
#include <conio.h>
#include<stdlib.h>
#include<stdio.h>
#include <Windows.h>
using namespace std;
using namespace cv;

//global variables

const bool debug = 0;
bool Manual_CTRL = false;

const int NMAX_x = 100;
const int NMAX_y = 2;

int initial_block_number;

//////////////////Class Definitions/////////////////////////////////

class bluetooth {

public:

        char outputchar[1];

        HANDLE hSerial;

        DWORD btsIO;

        bool connected;
        bool drop_fix;

        bluetooth(int n);  //constructor

        ~bluetooth(); //destructor
};

class lego_collection{

public:

        Mat MAT_Original_image;
        Mat MAT_Gray;
        Mat MAT_threshold;
        Mat MAT_HSV;
        Mat MAT_HSV_THRESHOLD_RED;
        Mat MAT_HSV_THRESHOLD_BLUE;
        Mat MAT_HSV_THRESHOLD_GREEN;
        Mat MAT_HSV_THRESHOLD_ROBOT;
        int x_y_cordinate_storage[NMAX_x][NMAX_y];
        int number_of_blocks_color;
        int X_COORD;
        int X_COORD_CIRCLE_front;
        int X_COORD_CIRCLE_back;
        int Y_COORD;
```

```
        int     Y_COORD_CIRCLE_front;
        int     Y_COORD_CIRCLE_back;
        int min_X_COORD;
        int min_Y_COORD;
        int DISTANCE_TO_BLOCK;
        int block_found;
        bool Block_in_claw;
        Point Robot_Centroid;
        Point Closest_Block_PT;
        double angle_circle_front_to_block;
        double angle_circle_back_to_block;
        float Angle_DIFF;
        void Image_Filtering();
        vector<Point2f>  Block_locations(Mat MAT_threshold, char debug_name[]);
        void Robot_location(Mat MAT_threshold, char debug_name
[]);                                    void Closest_block_distance(vector<Point2f>
Block_Centroid);
        char Closest_block_angle();
        char Move_robot(int block_location);
        void Find_closest_block(int block_color);
        void Update_block_distance(Point Block_Centroid);
        Mat line_image;
        void update_image();
        char Angle_to_DROPZONE(float xcord, float ycord);

        int angle_counter;
        bool Go_FWD;
        int flag;

        int block_number;
        int dropoff_array [27][3];
        int coord_array[27][2];

        bool backwards;
        bool home;

        lego_collection();

        ~lego_collection();

};


////////////////////////////////Constructors//////////////////////////////


bluetooth::bluetooth(int n){

        outputchar[0] = '0';

        connected = false;
        drop_fix = false;

        // Setup serial port connection and needed variables.
        hSerial = CreateFile(L"\\\\.\\COM14", GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0);

        while (!connected){
                if (hSerial != INVALID_HANDLE_VALUE)
                {
                        printf("Port opened! \n");

                        DCB dcbSerialParams;
                        GetCommState(hSerial, &dcbSerialParams);

                        dcbSerialParams.BaudRate = CBR_9600;
                        dcbSerialParams.ByteSize = 8;
                        dcbSerialParams.Parity = NOPARITY;
                        dcbSerialParams.StopBits = ONESTOPBIT;

                        SetCommState(hSerial, &dcbSerialParams);
                        connected = true;
```

```
                }
                else
                {
                                if (GetLastError() == ERROR_FILE_NOT_FOUND)
                                {
                                        printf("Serial port doesn't exist! \n");
                                }

                                printf("Error while setting up serial port! \n");

                                //delete file
                                DeleteFile(L"\\\\.\\COM14");

                                ClearCommError(hSerial, &btsIO, NULL);

                                //try to reconnect
                                hSerial = CreateFile(L"\\\\.\\COM14", GENERIC_READ | GENERIC_WRITE, 0, 0,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);

                }
        }

}

lego_collection::lego_collection(){

        block_found = 0;
        Block_in_claw = false;
        angle_counter = 0;
        Go_FWD = false;
        flag = 0;
        block_number = initial_block_number;

        backwards = false;
        home = false;

        //dropoff_array[block number][color]
        //dropoff_array[block number][size]
        //dropoff_array[block number][location]

        //color: 0=red, 1=blue, 2=green
        //size: 0=small, 1=large

        //Letter I with small blocks only, color sequence as per spec
        dropoff_array[0][0] = 1;
        dropoff_array[0][1] = 1;
        dropoff_array[0][2] = 0;

        dropoff_array[5][0] = 0;
        dropoff_array[5][1] = 1;
        dropoff_array[5][2] = 1;

        dropoff_array[10][0] = 2;
        dropoff_array[10][1] = 0;
        dropoff_array[10][2] = 2;

        dropoff_array[17][0] = 0;
        dropoff_array[17][1] = 1;
        dropoff_array[17][2] = 3;

        dropoff_array[24][0] = 1;
        dropoff_array[24][1] = 1;
        dropoff_array[24][2] = 4;

        //Letter H
        dropoff_array[1][0] = 1;
        dropoff_array[1][1] = 1;
        dropoff_array[1][2] = 5;

        dropoff_array[6][0] = 0;
        dropoff_array[6][1] = 1;
        dropoff_array[6][2] = 6;
```

```
dropoff_array[11][0] = 2;
dropoff_array[11][1] = 0;
dropoff_array[11][2] = 7;


dropoff_array[2][0] = 1;
dropoff_array[2][1] = 1;
dropoff_array[2][2] = 8;

dropoff_array[7][0] = 0;
dropoff_array[7][1] = 1;
dropoff_array[7][2] = 9;

dropoff_array[14][0] = 2;
dropoff_array[14][1] = 0;
dropoff_array[14][2] = 10;


dropoff_array[13][0] = 0;
dropoff_array[13][1] = 1;
dropoff_array[13][2] = 11;

dropoff_array[12][0] = 2;
dropoff_array[12][1] = 0;
dropoff_array[12][2] = 12;


dropoff_array[18][0] = 0;
dropoff_array[18][1] = 1;
dropoff_array[18][2] = 13;

dropoff_array[25][0] = 1;
dropoff_array[25][1] = 1;
dropoff_array[25][2] = 14;


dropoff_array[19][0] = 0;
dropoff_array[19][1] = 1;
dropoff_array[19][2] = 15;

dropoff_array[26][0] = 1;
dropoff_array[26][1] = 1;
dropoff_array[26][2] = 16;

//Letter U
dropoff_array[3][0] = 1;
dropoff_array[3][1] = 1;
dropoff_array[3][2] = 17;

dropoff_array[8][0] = 0;
dropoff_array[8][1] = 1;
dropoff_array[8][2] = 18;

dropoff_array[15][0] = 2;
dropoff_array[15][1] = 0;
dropoff_array[15][2] = 19;

dropoff_array[20][0] = 1;
dropoff_array[20][1] = 1;
dropoff_array[20][2] = 20;


dropoff_array[4][0] = 1;
dropoff_array[4][1] = 1;
dropoff_array[4][2] = 21;

dropoff_array[9][0] = 0;
dropoff_array[9][1] = 1;
dropoff_array[9][2] = 22;

dropoff_array[16][0] = 2;
```

```
dropoff_array[16][1] = 0;
dropoff_array[16][2] = 23;

dropoff_array[23][0] = 1;
dropoff_array[23][1] = 1;
dropoff_array[23][2] = 24;


dropoff_array[22][0] = 0;
dropoff_array[22][1] = 1;
dropoff_array[22][2] = 25;

dropoff_array[21][0] = 2;
dropoff_array[21][1] = 0;
dropoff_array[21][2] = 26;

//coordinates for dropoff
//coord_array[location][x]
//coord_array[location][y]

//Letter I
coord_array[0][0] = 580;
coord_array[0][1] = 40;

coord_array[1][0] = 580;
coord_array[1][1] = 60;

coord_array[2][0] = 580;
coord_array[2][1] = 80;

coord_array[3][0] = 580;
coord_array[3][1] = 100;

coord_array[4][0] = 580;
coord_array[4][1] = 120;

//Letter H
coord_array[5][0] = 530;
coord_array[5][1] = 40;

coord_array[6][0] = 530;
coord_array[6][1] = 60;

coord_array[7][0] = 530;
coord_array[7][1] = 80;

coord_array[8][0] = 470;
coord_array[8][1] = 40;

coord_array[9][0] = 470;
coord_array[9][1] = 60;

coord_array[10][0] = 470;
coord_array[10][1] = 80;

coord_array[11][0] = 490;
coord_array[11][1] = 80;

coord_array[12][0] = 510;
coord_array[12][1] = 80;

coord_array[13][0] = 530;
coord_array[13][1] = 100;

coord_array[14][0] = 530;
coord_array[14][1] = 120;

coord_array[15][0] = 470;
coord_array[15][1] = 100;

coord_array[16][0] = 470;
coord_array[16][1] = 120;
```

```
        //Letter U

        coord_array[17][0] = 420;
        coord_array[17][1] = 40;

        coord_array[18][0] = 420;
        coord_array[18][1] = 60;

        coord_array[19][0] = 420;
        coord_array[19][1] = 80;

        coord_array[20][0] = 420;
        coord_array[20][1] = 100;

        coord_array[21][0] = 360;
        coord_array[21][1] = 40;

        coord_array[22][0] = 360;
        coord_array[22][1] = 60;

        coord_array[23][0] = 360;
        coord_array[23][1] = 80;

        coord_array[24][0] = 360;
        coord_array[24][1] = 100;

        coord_array[25][0] = 380;
        coord_array[25][1] = 100;

        coord_array[26][0] = 400;
        coord_array[26][1] = 100;

}


////////////////////////////////Destructors/////////////////////////////////


bluetooth::~bluetooth(){

        CloseHandle(hSerial);

        //clear stuff for reconnect
        DeleteFile(L"\\\\.\\COM14");
        ClearCommError(hSerial, &btsIO, NULL);

        printf("Port closed! \n");

}

lego_collection::~lego_collection(){

        printf("lego destructed \n");

}


////////////////////////////////Member Functions///////////////////////////////////


void lego_collection::update_image(){

        MAT_Gray.create(MAT_Original_image.size(), MAT_Original_image.type());
        cvtColor(MAT_Original_image, MAT_Gray, CV_BGR2GRAY);
        Mat MAT_myblur;
        blur(MAT_Original_image, MAT_myblur, Size(10, 10), Point(-1, -1), BORDER_DEFAULT);


        cvtColor(MAT_myblur, MAT_HSV, COLOR_BGR2HSV);
```

```
        inRange(MAT_HSV, Scalar(23, 114, 0), Scalar(47, 255, 255), MAT_HSV_THRESHOLD_ROBOT);     erode
(MAT_HSV_THRESHOLD_ROBOT, MAT_HSV_THRESHOLD_ROBOT, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
        dilate(MAT_HSV_THRESHOLD_ROBOT, MAT_HSV_THRESHOLD_ROBOT, getStructuringElement(MORPH_ELLIPSE, Size
(5, 5)));
        erode(MAT_HSV_THRESHOLD_ROBOT, MAT_HSV_THRESHOLD_ROBOT, getStructuringElement(MORPH_ELLIPSE, Size
(5, 5)));

        return;
}


void lego_collection::Image_Filtering()
{

        MAT_Gray.create(MAT_Original_image.size(), MAT_Original_image.type());
        cvtColor(MAT_Original_image, MAT_Gray, CV_BGR2GRAY);

        Mat MAT_myblur;         // blur image to take out noise
        blur(MAT_Original_image, MAT_myblur, Size(10, 10), Point(-1, -1), BORDER_DEFAULT);

        cvtColor(MAT_myblur, MAT_HSV, COLOR_BGR2HSV);    // change to HSV

        // red

        inRange(MAT_HSV, Scalar(0, 26, 0), Scalar(20, 255, 255), MAT_HSV_THRESHOLD_RED);
        erode(MAT_HSV_THRESHOLD_RED, MAT_HSV_THRESHOLD_RED, getStructuringElement(MORPH_ELLIPSE, Size(5,
5)));
        dilate(MAT_HSV_THRESHOLD_RED, MAT_HSV_THRESHOLD_RED, getStructuringElement(MORPH_ELLIPSE, Size(5,
5)));

        if (debug == 1) imshow("Red threshold", MAT_HSV_THRESHOLD_RED);


        // blue

        inRange(MAT_HSV, Scalar(103, 35, 0), Scalar(178, 204, 255), MAT_HSV_THRESHOLD_BLUE);     erode
(MAT_HSV_THRESHOLD_BLUE, MAT_HSV_THRESHOLD_BLUE, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)));
        dilate(MAT_HSV_THRESHOLD_BLUE, MAT_HSV_THRESHOLD_BLUE, getStructuringElement(MORPH_ELLIPSE, Size
(5, 5)));

        if (debug == 1) imshow("Blue threshold", MAT_HSV_THRESHOLD_BLUE);


        // green

        inRange(MAT_HSV, Scalar(61, 23, 47), Scalar(82, 100, 206), MAT_HSV_THRESHOLD_GREEN);
        erode(MAT_HSV_THRESHOLD_GREEN, MAT_HSV_THRESHOLD_GREEN, getStructuringElement(MORPH_ELLIPSE, Size
(5, 5)));
        dilate(MAT_HSV_THRESHOLD_GREEN, MAT_HSV_THRESHOLD_GREEN, getStructuringElement(MORPH_ELLIPSE, Size
(5, 5)));

        if (debug == 1) imshow("Green threshold", MAT_HSV_THRESHOLD_GREEN);


        //yellow

        inRange(MAT_HSV, Scalar(23, 114, 0), Scalar(47, 255, 255), MAT_HSV_THRESHOLD_ROBOT);
        erode(MAT_HSV_THRESHOLD_ROBOT, MAT_HSV_THRESHOLD_ROBOT, getStructuringElement(MORPH_ELLIPSE, Size
(5, 5)));
        dilate(MAT_HSV_THRESHOLD_ROBOT, MAT_HSV_THRESHOLD_ROBOT, getStructuringElement(MORPH_ELLIPSE, Size
(5, 5)));
        erode(MAT_HSV_THRESHOLD_ROBOT, MAT_HSV_THRESHOLD_ROBOT, getStructuringElement(MORPH_ELLIPSE, Size
(5, 5)));


}

vector<Point2f>  lego_collection::Block_locations(Mat MAT_threshold, char debug_name[])
{
        int     X_COORD = 0; // temporary coordinate
        int     Y_COORD = 0; // temporary coordinate
```

```
	//possibly more filtering
	Mat Overaly_gray = MAT_Gray.clone();
	vector<vector<Point> > contours;
	findContours(MAT_threshold.clone(), contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
	vector<Point> APPROX_POLYGON;

	number_of_blocks_color = 1;

	vector<Moments> Moment(contours.size());// moment
	vector<Point2f> Block_Centroids_Container(contours.size());	// center of mass


	for (int i = 0; i < contours.size(); i++)
	{

		drawContours(Overaly_gray, contours, i, (0, 0, 255), 4, 8, vector<Vec4i>(), 0, Point());

		approxPolyDP(Mat(contours[i]), APPROX_POLYGON, arcLength(Mat(contours[i]), true)*0.02,
true);
		Moment[i] = moments(contours[i], false);
		if (Moment[i].m00 > 3)				// reduce noise
		{
					vector<Rect> boundRect(contours.size());
					vector<vector<Point> > contours_poly(contours.size());
					approxPolyDP(Mat(contours[i]), contours_poly[i], 3, true);
					boundRect[i] = boundingRect(Mat(contours_poly[i]));
					Scalar color(0, 0, 255);
					drawContours(Overaly_gray, contours_poly, i, color, 4, 8,
vector<Vec4i>(), 0, Point());
					// Find center point of rectangles
					X_COORD = Moment[i].m10 / Moment[i].m00;
					Y_COORD = Moment[i].m01 / Moment[i].m00;

					// delete objects in work zone
					if (X_COORD  > 320 && Y_COORD < 240){

						break;
					}
					Block_Centroids_Container[i] = Point2f(X_COORD,
Y_COORD);
					circle(Overaly_gray, Block_Centroids_Container[i], 4, Scalar(0,
255, 0), -1, 8, 0);			// draws center point

					std::string my_rectangle;
					std::stringstream my_number_of_blocks_color;
					my_number_of_blocks_color << number_of_blocks_color;
					number_of_blocks_color++;
					my_rectangle = "RECT " + my_number_of_blocks_color.str();
					putText(Overaly_gray, my_rectangle, Block_Centroids_Container[i],
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0, 255, 0), 2.0);

					// stores coordinates of blocks
					x_y_cordinate_storage[number_of_blocks_color][0] = X_COORD;
					x_y_cordinate_storage[number_of_blocks_color][1] = Y_COORD;
		}
	}
	if (debug == 1) imshow(debug_name, Overaly_gray);
	return Block_Centroids_Container;

}



void lego_collection::Robot_location(Mat MAT_threshold, char debug_name[])
{
	Mat Overaly_gray = MAT_Gray.clone();
	vector<vector<Point> > contours;
	findContours(MAT_threshold.clone(), contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
	vector<Point> APPROX_POLYGON;

	vector<Moments> robot_Moment(contours.size());  // moment
	vector<Point2f> robot_Centroid_temp(contours.size());	// center of mass
```

```
	for (int i = 0; i < contours.size(); i++)
	{

		drawContours(Overaly_gray, contours, i, (0, 0, 255), 4, 8, vector<Vec4i>(), 0, Point
());
		robot_Moment[i] = moments(contours[i], false);
		if (robot_Moment[i].m00 > 20)
		{
			double area = cv::contourArea(contours[i]);
			cv::Rect r = cv::boundingRect(contours[i]);
			int radius = r.width / 2;

			if (std::abs(1 - ((double)r.width / r.height)) <= 0.2 && std::abs(1 - (area /
(CV_PI * std::pow(radius, 2)))) <= 0.2)
			{
					//// Draw circle
					vector<Rect> boundRect(contours.size());
					vector<vector<Point> > contours_poly(contours.size());
					approxPolyDP(Mat(contours[i]), contours_poly[i], 3, true);
					boundRect[i] = boundingRect(Mat(contours_poly[i]));
					Scalar color(0, 0,
255);

					drawContours(Overaly_gray, contours_poly, i, color, 2, 8,
vector<Vec4i>(), 0, Point());

					// Find center point of circle
					robot_Moment[i] = moments(contours[i], false);

					if (robot_Moment[i].m00 > 20 && robot_Moment[i].m00
<1000)                                          // for small circle
					{
					X_COORD_CIRCLE_front = robot_Moment[i].m10 / robot_Moment[i].m00;
					Y_COORD_CIRCLE_front = robot_Moment[i].m01 / robot_Moment[i].m00;
					if (X_COORD_CIRCLE_front == 0 || Y_COORD_CIRCLE_front == 0) break;
					robot_Centroid_temp[i] = Point2f(X_COORD_CIRCLE_front,
Y_COORD_CIRCLE_front);                          // coordinates
					circle(Overaly_gray, robot_Centroid_temp[i], 4, Scalar(0, 255, 0),
-1, 8, 0);
					std::string my_rbg;
					std::stringstream my_array_rbg;
					my_array_rbg << "Robot front";
					my_rbg = my_array_rbg.str();
					putText(Overaly_gray, my_rbg, robot_Centroid_temp[i],
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0, 255, 0), 2.0);
					}

					if (robot_Moment[i].m00 > 1000) // for large circle
					{
					X_COORD_CIRCLE_back = robot_Moment[i].m10 / robot_Moment[i].m00;
					Y_COORD_CIRCLE_back = robot_Moment[i].m01 / robot_Moment[i].m00;
					robot_Centroid_temp[i] = Point2f(X_COORD_CIRCLE_back,
Y_COORD_CIRCLE_back);                           // coordinates
					circle(Overaly_gray, robot_Centroid_temp[i], 4, Scalar(0, 255, 0),
-1, 8, 0);                                      // draws center point

					}
			}
		}

	}
	if (debug == 1) imshow(debug_name, Overaly_gray);
}

void lego_collection::Closest_block_distance(vector<Point2f> Block_Centroid)
{
	int tempDISTANCE_TO_BLOCK;

	for (int n = 0; n < Block_Centroid.size(); n++){
```

```
                if (Block_Centroid[n].x != 0 || Block_Centroid[n].y != 0){
                        tempDISTANCE_TO_BLOCK = sqrt((pow((Block_Centroid[n].x - X_COORD_CIRCLE_front),
2)) + (pow((Block_Centroid[n].y - Y_COORD_CIRCLE_front), 2)));

                        if (DISTANCE_TO_BLOCK > tempDISTANCE_TO_BLOCK)
                        {
                                DISTANCE_TO_BLOCK = tempDISTANCE_TO_BLOCK;
                                Closest_Block_PT = Block_Centroid[n];
                        }
                }
        }

}

void lego_collection::Update_block_distance(Point Block_Centroid)
{

        DISTANCE_TO_BLOCK = sqrt((pow((Block_Centroid.x - X_COORD_CIRCLE_front), 2)) + (pow
((Block_Centroid.y - Y_COORD_CIRCLE_front), 2)));

}

void lego_collection::Find_closest_block(int block_color){

        // IGNORE DROP OFF ZONE LOCATIONS
        Robot_location(MAT_HSV_THRESHOLD_ROBOT, "robot");

        // reset block distance
        DISTANCE_TO_BLOCK = 100000;

        if (block_color == 0)
        {
                Closest_block_distance(Block_locations(MAT_HSV_THRESHOLD_RED, "red"));
                // check distance with temp var
                cout << "locating closest red block" << endl;
        }
        else if (block_color == 2)
        {
                Closest_block_distance(Block_locations(MAT_HSV_THRESHOLD_GREEN, "Green"));
                // check distance with temp var
                cout << "locating closest green block" << endl;
        }
        else if (block_color == 1)
        {
                Closest_block_distance(Block_locations(MAT_HSV_THRESHOLD_BLUE, "Blue"));
                // check distance with temp var
                cout << "locating closest blue block" << endl;
        }
        else {
                cout << "invalid color choice" << endl;
        }

        line_image = MAT_Gray.clone();

        //draw line to closest block

        cout << "closest block at:\t" << Closest_Block_PT << endl;
        line(line_image, Closest_Block_PT, Point(X_COORD_CIRCLE_front, Y_COORD_CIRCLE_front), Scalar(0,
255, 0), 4, 8, 0);      // closest block to robot

        imshow("Line image", line_image);
}

char lego_collection::Closest_block_angle()
{

        Point v1 = Closest_Block_PT;
        Point v2 = Closest_Block_PT;

        v1.x = v1.x - X_COORD_CIRCLE_front;
        v1.y = v1.y - Y_COORD_CIRCLE_front;
        v2.x = v2.x - X_COORD_CIRCLE_back;
```

```
        v2.y = v2.y - Y_COORD_CIRCLE_back;

        float len1 = sqrt(v1.x * v1.x + v1.y * v1.y);
        float len2 = sqrt(v2.x * v2.x + v2.y * v2.y);

        float dot = v1.x * v2.x + v1.y * v2.y;

        Angle_DIFF = dot / (len1 * len2);

        Angle_DIFF = atan2((float)v1.y, (float)v1.x) - atan2((float)v2.y, (float)v2.x);

        if (Angle_DIFF < 0) Angle_DIFF += 2 * CV_PI;

        Angle_DIFF = Angle_DIFF * (180.0 / CV_PI);

        if (len2 < len1){
                backwards = true;
         }

        return '0';
}

char lego_collection::Angle_to_DROPZONE(float xcord, float ycord)
{

        Point v1;
        Point v2;

        v1.x = xcord - X_COORD_CIRCLE_front;
        v1.y = ycord - Y_COORD_CIRCLE_front;
        v2.x = xcord - X_COORD_CIRCLE_back;
        v2.y = ycord - Y_COORD_CIRCLE_back;

        float len1 = sqrt(v1.x * v1.x + v1.y * v1.y);
        float len2 = sqrt(v2.x * v2.x + v2.y * v2.y);

        float dot = v1.x * v2.x + v1.y * v2.y;

        Angle_DIFF = dot / (len1 * len2);

        Angle_DIFF = atan2((float)v1.y, (float)v1.x) - atan2((float)v2.y, (float)v2.x);

        if (Angle_DIFF < 0) Angle_DIFF += 2 * CV_PI;

        Angle_DIFF = Angle_DIFF * (180.0 / CV_PI);

        if (len2 < len1){
                backwards = true;
        }

        return '0';
}


char lego_collection::Move_robot(int block_location){

        Robot_location(MAT_HSV_THRESHOLD_ROBOT, "robot");

        Update_block_distance(Closest_Block_PT);

        line_image = MAT_Gray.clone();

        backwards = false; //reset this every loop

        //draw line to closest block
        if (Block_in_claw == false){

                if (home == true){
                        Update_block_distance(Point(500, 350));
                        Angle_to_DROPZONE(500, 350);
                        line(line_image, Point(500, 350), Point(X_COORD_CIRCLE_front,
Y_COORD_CIRCLE_front), Scalar(0, 255, 0), 4, 8, 0);
```

```
                        cout << "going to home location" << endl;
                }
                else{
                        Closest_block_angle();
                        line(line_image, Closest_Block_PT, Point(X_COORD_CIRCLE_front,
Y_COORD_CIRCLE_front), Scalar(0, 255, 0), 4, 8, 0);     // closest block to robot
                }

                // grid lines
                line(line_image, Point(320, 240), Point(640, 240), Scalar(0, 255, 0), 4, 8, 0);
                line(line_image, Point(320, 0), Point(320, 480), Scalar(0, 255, 0), 4, 8, 0);

                // text on robot (distance and angle)
                std::string my_rbg;
                std::stringstream my_array_rbg;
                my_array_rbg <<
DISTANCE_TO_BLOCK;                                                                      my_rbg =
my_array_rbg.str();

                std::string my_rbg2;
                std::stringstream my_array_rbg2;
                my_array_rbg2 << Angle_DIFF;
                my_rbg2 = my_array_rbg2.str();

                putText(line_image, my_rbg, Point(X_COORD_CIRCLE_back, Y_COORD_CIRCLE_back),
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0, 255, 0), 2.0);
                putText(line_image, my_rbg2, Point(X_COORD_CIRCLE_front, Y_COORD_CIRCLE_front),
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0, 255, 0), 2.0);

                imshow("Line image", line_image);

                if (DISTANCE_TO_BLOCK >= 120){

                        block_found = 1;

                        //angle robot to closest block
                        if ((Angle_DIFF > 5) && (Angle_DIFF < 50) && Go_FWD == false)
                        {
                                cout << "positioning - right" << endl;
                                return 'j'; //fast turn
                        }
                        else if ((Angle_DIFF > 310) && (Angle_DIFF < 355) && Go_FWD == false)
                        {
                                cout << "positioning - left" << endl;
                                return 'l'; //fast turn
                        }

                        if ((Angle_DIFF > 0.5) && (Angle_DIFF < 5) && Go_FWD == false)
                                {
                                        cout << "positioning - right" << endl;
                                        return 'd'; //slow turn
                                }
                        else if ((Angle_DIFF > 355) && (Angle_DIFF < 359.5) && Go_FWD == false)
                                {
                                        cout << "positioning - left" << endl;
                                        return 'a'; //slow turn
                                }

                        if (((Angle_DIFF <= 0.5) || (Angle_DIFF >= 359.5)) && backwards == true)
                        {
                                return 'd'; //spin robot if directly backwards
                        }

                        if (((Angle_DIFF <= 0.5) || (Angle_DIFF >= 359.5)) && backwards==false)
                        {
                                Go_FWD = true;
                        }

                        //move robot forward
                        if (Go_FWD == true)
                        {
                                cout << "booking it!" << endl;
```

```
                        return 'i';
                }

        }


        if (DISTANCE_TO_BLOCK >= 80 && DISTANCE_TO_BLOCK < 120){

                block_found = 1;

                if (!flag){
                        Go_FWD = false; //reset Go_FWD to allow re-positioning
                        flag = 1;
                }

                if (home == true){
                        Block_in_claw = false;
                        home = false;
                        flag = 0;
                        Go_FWD = false;
                        return '0'; //dont try to pickup when going home
                }

                //angle robot to closest block
                if ((Angle_DIFF > 0.5) && (Angle_DIFF < 50) && Go_FWD == false)
                {
                        //cout << "positioning - right" << endl;
                        return 'd';
                }
                else if ((Angle_DIFF > 310) && (Angle_DIFF < 359.5) && Go_FWD == false)
                {
                        //cout << "positioning - left" << endl;
                        return 'a';
                }

                if (((Angle_DIFF <= 0.5) || (Angle_DIFF >= 359.5)) && backwards==false)
                {
                        Go_FWD = true;
                }

                //move robot forward
                if (Go_FWD == true)
                {
                        //cout << "on the move!" << endl;
                        return 'w';
                }

        }


        if (DISTANCE_TO_BLOCK > 70 && DISTANCE_TO_BLOCK < 80)  {

                block_found = 1;

                // FINE TUNE ANGLE
                if ((Angle_DIFF > 0.5) && (Angle_DIFF < 50) && angle_counter<5 && home==false)
                {
                        return 'h';
                }
                else if ((Angle_DIFF > 310) && (Angle_DIFF < 359.5) && angle_counter<5 && home ==
false)
                {
                        return 'f';
                }

                if (angle_counter<5 && ((Angle_DIFF <= 0.5) || (Angle_DIFF >= 359.5)) && home ==
false)
                {
                        cout << "locked on! #" << angle_counter + 1 << endl;
                        angle_counter++;
                        return '0'; //send a void command to arduino
                }
```

```
                        if (angle_counter >= 5 && home == false){ //ensures angle is small
                                cout << "final lock!" << endl;
                                cout << "Picking up block!" << endl;
                                angle_counter = 0;
                                flag = 0;
                                Go_FWD = false;
                                Block_in_claw = true;
                                home = true;
                                return 'g'; //pick up block

                        }

                }

                if (DISTANCE_TO_BLOCK <= 70)  {
                        return 'y'; //fine backup
                }

        }

        if (Block_in_claw == true){

                cout << "i have a block" << endl;

                if (home == true){
                        Update_block_distance(Point(coord_array[block_location][0]+50, 350));
                        Angle_to_DROPZONE(coord_array[block_location][0]+50, 350);
                        line(line_image, Point(coord_array[block_location][0]+50, 350), Point
(X_COORD_CIRCLE_front, Y_COORD_CIRCLE_front), Scalar(0, 255, 0), 4, 8, 0);
                        cout << "going to home location" << endl;
                }
                else {
                        Update_block_distance(Point(coord_array[block_location][0], coord_array
[block_location][1]));
                        Angle_to_DROPZONE(coord_array[block_location][0], coord_array[block_location][1]);
                        line(line_image, Point(coord_array[block_location][0], coord_array[block_location]
[1]), Point(X_COORD_CIRCLE_front, Y_COORD_CIRCLE_front), Scalar(0, 255, 0), 4, 8, 0);
                        cout << "going to dropoff location #" << block_location << endl;
                }


                //draw lines
                line(line_image, Point(320, 240), Point(640, 240), Scalar(0, 255, 0), 4, 8, 0);
                line(line_image, Point(320, 0), Point(320, 480), Scalar(0, 255, 0), 4, 8, 0);

                //text on robot (distance and angle)
                std::string my_rbg;
                std::stringstream my_array_rbg;
                my_array_rbg << DISTANCE_TO_BLOCK;
                my_rbg = my_array_rbg.str();

                std::string my_rbg2;
                std::stringstream my_array_rbg2;
                my_array_rbg2 << Angle_DIFF;
                my_rbg2 = my_array_rbg2.str();

                putText(line_image, my_rbg, Point(X_COORD_CIRCLE_back, Y_COORD_CIRCLE_back),
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0, 255, 0), 2.0);
                putText(line_image, my_rbg2, Point(X_COORD_CIRCLE_front, Y_COORD_CIRCLE_front),
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0, 255, 0), 2.0);

                imshow("Line image", line_image);

                if (DISTANCE_TO_BLOCK >= 100){

                        block_found = 1;

                        //angle robot to dropzone
                        if (home == true){
                                if ((Angle_DIFF > 5) && (Angle_DIFF < 50) && Go_FWD == false)
                                {
```

```
                        cout << "positioning - right" << endl;
                        return 'j'; //fast turn
                }
                else if ((Angle_DIFF > 310) && (Angle_DIFF < 355) && Go_FWD == false)
                {
                        cout << "positioning - left" << endl;
                        return 'l'; //fast turn
                }

                if ((Angle_DIFF > 0.5) && (Angle_DIFF < 5) && Go_FWD == false)
                {
                        cout << "positioning - right" << endl;
                        return 'd'; //slow turn
                }
                else if ((Angle_DIFF > 355) && (Angle_DIFF < 359.5) && Go_FWD == false)
                {
                        cout << "positioning - left" << endl;
                        return 'a'; //slow turn
                }
        }
        else{

                if ((Angle_DIFF > 0.5) && (Angle_DIFF < 50) && Go_FWD == false)
                {
                        cout << "positioning - right" << endl;
                        return 'd'; //slow turn
                }
                else if ((Angle_DIFF > 310) && (Angle_DIFF < 359.5) && Go_FWD == false)
                {
                        cout << "positioning - left" << endl;
                        return 'a'; //slow turn
                }

        }

        if (((Angle_DIFF <= 0.5) || (Angle_DIFF >= 359.5)) && backwards == true)
        {
                return 'd'; //spin robot if directly backwards
        }

        if (((Angle_DIFF <= 0.5) || (Angle_DIFF >= 359.5)) && backwards==false)
        {
                Go_FWD = true;
        }

        //move robot forward
        if (Go_FWD == true)
        {
                cout << "booking it!" << endl;
                return 'i';
        }

}

if (DISTANCE_TO_BLOCK > 42 && DISTANCE_TO_BLOCK < 100){

        block_found = 1;

        if (!flag){
                Go_FWD = false; //reset Go_FWD to allow re-positioning
                flag = 1;
        }

        if (home == true){
                Block_in_claw = true;
                home = false;
                Go_FWD = false;
                flag = 0;
                return '0'; //send a void command to arduino
        }

        //angle robot to dropzone
```

```
                        if ((Angle_DIFF > 0.5) && (Angle_DIFF < 50) && Go_FWD == false)
                        {
                                return 'd';
                        }
                        else if ((Angle_DIFF > 310) && (Angle_DIFF < 359.5) && Go_FWD == false)
                        {
                                return 'a';
                        }

                        if (((Angle_DIFF <= 0.5) || (Angle_DIFF >= 359.5)) && backwards==false)
                        {
                                Go_FWD = true;
                        }

                        //move robot forward
                        if (Go_FWD == true)
                        {
                                cout << "on the move!" << endl;
                                return 'w';
                        }

                }


                if (DISTANCE_TO_BLOCK <= 42 && DISTANCE_TO_BLOCK >= 38)
                {

                        // FINE TUNE ANGLE
                        if ((Angle_DIFF > 0.5) && (Angle_DIFF < 50) && angle_counter<5 && home==false)
                        {
                                return 'h';
                        }
                        else if ((Angle_DIFF > 310) && (Angle_DIFF < 359.5) && angle_counter<5 && home ==
false)
                        {
                                return 'f';
                        }

                        if (angle_counter<5 && ((Angle_DIFF <= 0.5) || (Angle_DIFF >= 359.5)) && home ==
false)
                        {
                                cout << "locked on! #" << angle_counter + 1 << endl;
                                angle_counter++;
                                return '0'; //send a void command to arduino
                        }

                        if (angle_counter >= 5 && home == false){ //ensures angle is small
                                cout << "dropping off lego, great success!" << endl;
                                angle_counter = 0;
                                flag = 0;
                                block_found = 0;
                                Go_FWD = false;
                                Block_in_claw = false;
                                home = true;
                                block_number++; //increment block number
                                return 'r'; //drop off block
                        }
                }

                if (DISTANCE_TO_BLOCK < 38){
                        return 'y'; //fine backup
                }

        }

        cout << "NO COMMAND, IM LOST!!!" << endl;

        return 'q';

}

////////////////////            MAIN LOOP              ////////////////////////////////////
```

```
int edgeThresh = 1;
int lowThreshold = 10;
int const max_lowThreshold = 100;
int ratio = 3;
int kernel_size = 3;
char* window_name = "Edge Map";

int main()
{
	//time variables
	double t_clock;
	double t0;
	static int init = 0;

	//set the block number
	cout << "Which block would you like to start with ? ";
	cin >> initial_block_number;
	cout << "Starting at block " << initial_block_number << "\n";

	//array starts at 0 so correct human input.
	initial_block_number = initial_block_number - 1;

	lego_collection Blocks; //Object initialization

	bluetooth *panther; //pointer for dynamic object

	panther = new bluetooth(8); //dynamic object

	if (panther == NULL) {
		printf("\nerror creating dynamic panther");
		return 0;
	}

	//debug variables
	bool debugMode = false;
	bool trackingEnabled = false;
	bool pause = false; //pause and resume code
	int counter = 0;

	// open webcam
	VideoCapture capWebcam(0);
	cout << "starting code  " << endl;
	if (!capWebcam.isOpened())  { cout << "Cannot open webcam/n"; return -1; }


	//////////////////////////Continuous Loop/////////////////
	while (1)
	{

		if (!init){
			t0 = clock();
			init = 1;
		}

		t_clock = clock() - t0; // time since start of program

		bool bSuccess = capWebcam.read(Blocks.MAT_Original_image);
		if (!bSuccess) { cout << "Cannot read a frame from video/n"; break; }

		Mat dst = Blocks.MAT_Gray.clone();
		Mat test1 = Blocks.MAT_Gray.clone();

		if (!Blocks.block_found || debug){
			Blocks.Image_Filtering();
                                        Blocks.Find_closest_block(Blocks.dropoff_array
[Blocks.block_number][0]);
			Blocks.block_found = 1;
		}
		else{
			Blocks.update_image();
```

```
		}


		if (Blocks.block_number == 5 || Blocks.block_number == 10 || Blocks.block_number==15 ||
Blocks.block_number==20){

			if (panther->drop_fix==false){
				cout << "Reconnecting the robot for safety!" << endl;
				if (panther != NULL) {
					delete panther;
					panther = NULL;
				}
				else {
				cout << "<<<error deleting dynamic panther >>>" << endl;
				}

				panther = new bluetooth(14); //dynamic object

				if (panther == NULL) {
				cout << "<<<error creating dynamic panther >>>" << endl;
					return 0;
				}
			}
			panther->drop_fix = true;
		}

		if (Blocks.block_number == 6 || Blocks.block_number == 11 || Blocks.block_number==16 ||
Blocks.block_number==21){
			panther->drop_fix = false; // reset this so it will reconnect again
		}

	//////////////////Char sent to Robot////////////////////////////////////
		if (t_clock > 80) { //time delay line 1

			panther->outputchar[0] = Blocks.Move_robot(Blocks.dropoff_array
[Blocks.block_number][2]);

			if (Manual_CTRL == false){

				if (panther != NULL){ //only try to write when panther exists
					WriteFile(panther->hSerial, panther->outputchar, strlen(panther-
>outputchar), &panther->btsIO, NULL);
				}
			}

			t0 = clock(); //time delay line 2
		}


	//////////////////////Switch for exiting program/////////////////////////////
		int keypress;
		keypress = waitKey(10);

		switch (keypress){

		case 27: //'esc' key has been pressed, exit program.

			if (panther != NULL) {
				delete panther;
				return 0;
			}
			else {
				printf("\nerror deleteing dynamic panther");
				return 0;
			}

		case 'q': //'esc' key has been pressed, exit program.

			if (panther != NULL) {
				delete panther;
				return 0;
```

```
                        }
                        else {
                                printf("\nerror deleteing dynamic panther");
                                return 0;
                        }
                case 'z': //'esc' key has been pressed, exit program.

                        cout << "Reconnecting the robot for safety!" << endl;
                        if (panther != NULL) {
                                delete panther;
                                panther = NULL;
                        }
                        else {
                                cout << "<<<error deleting dynamic panther >>>" << endl;
                        }

                        panther = new bluetooth(14); //dynamic object

                        if (panther == NULL) {
                                cout << "<<<error creating dynamic panther >>>" << endl;
                                return 0;
                        }

                case '1': // toggle Manual mode
                                Manual_CTRL = ! Manual_CTRL;
                                cout << "Manual_CTRL =  " << Manual_CTRL << endl;

                                break;


                case '2':
                                cout << "Block_in_claw =  " << Blocks.Block_in_claw << endl;
                                Blocks.Block_in_claw = !Blocks.Block_in_claw;

                                break;

                        default:
                                if (keypress > 10){
                                        panther->outputchar[0] = keypress;
                                        keypress = 0;

                                        if (panther != NULL)
{                                                                                                       WriteFile
(panther->hSerial, panther->outputchar, strlen(panther->outputchar), &panther->btsIO, NULL);
                                        }
                                }
                                break;


                }


        } ///////end of while loop

        if (panther != NULL) {
                delete panther;
                return 0;
        }
        else {
                printf("\nerror deleting dynamic panther");
                return 0;
        }

}
```

APPENDIX C: ARDUINO SOFWARE: Arduino.ino

```
#include <Servo.h>
//#include <NewSoftSerial/NewSoftSerial.h>

//NewSoftSerial mySerial(0, 1);

////////////////////////////////////////////////////////////////////////////
Servo servo_right, servo_left, servo_up, servo_claw, servo_arm;
int servoPosition = 90;
char incomingByte;   // for incoming serial data

void wheel_attach();
void wheel_detach();
void arm_attach();
void arm_detach();
void claw_attach();
void claw_detach();


void setup()
{
        Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
        pinMode(13, OUTPUT);

        arm_attach(servo_arm);
        servo_arm.write(90);    //arm up
        delay(400);
        servo_arm.write(65);    //arm down
        delay(400);
        servo_arm.write(90);    //arm up
        delay(400);

        claw_attach(servo_claw);//claw open
        servo_claw.write(40);
        delay(400);
        servo_claw.write(100);  //claw close
        delay(400);
        servo_claw.write(40);   //claw open
        delay(400);
        claw_detach(servo_claw);
}

void loop()
{

        //96/96
        //Serial.print("1");

        if (Serial.available() > 0)
        {
                // read the incoming byte:
                // wait for a second
                incomingByte = Serial.read();
                Serial.println(incomingByte);

                switch (incomingByte)
                {
                        //Serial.print(incom)



                case 'w':                                               //straight
                        wheel_attach(servo_left, servo_right);

                        servo_right.write(255);
```

```
        servo_left.write(0);
        delay(25);

        wheel_detach(servo_left, servo_right);
        break;

case 'd':                                                   //right turn
        wheel_attach(servo_left, servo_right);

        servo_right.write(0);
        servo_left.write(0);
        delay(25);

        wheel_detach(servo_left, servo_right);
        break;


case 'a':                                                   //left turn
        wheel_attach(servo_left, servo_right);

        servo_right.write(255);
        servo_left.write(255);
        delay(25);

        wheel_detach(servo_left, servo_right);
        break;

case 's':                                                   //reverse
        wheel_attach(servo_left, servo_right);

        servo_right.write(0);
        servo_left.write(250);
        delay(25);

        wheel_detach(servo_left, servo_right);
        break;

case 't':                                                   //straight fine tune
        wheel_attach(servo_left, servo_right);

        servo_right.write(100);
        servo_left.write(90);
        delay(25);

        wheel_detach(servo_left, servo_right);
        break;

case 'h':                                                   //right turn fine tune
        wheel_attach(servo_left, servo_right);

        servo_right.write(90);
        servo_left.write(90);
        delay(25);

        wheel_detach(servo_left, servo_right);
        break;


case 'f':                                                   //left turn fine tune
        wheel_attach(servo_left, servo_right);

        servo_right.write(100);
        servo_left.write(100);
        delay(25);

        wheel_detach(servo_left, servo_right);
        break;

case 'y':                                                   //reverse fine tune
        wheel_attach(servo_left, servo_right);

        servo_right.write(90);
```

```
        servo_left.write(100);
        delay(25);

        wheel_detach(servo_left, servo_right);
        break;

case 'q':                                                   //disconnect
        wheel_detach(servo_left, servo_right);
        arm_detach(servo_arm);
        claw_detach(servo_claw);

        break;

case 'i':                                                   //straight cont.
        wheel_attach(servo_left, servo_right);

        servo_right.write(255);
        servo_left.write(0);

        break;

case 'j':                                                   //right turn cont.
        wheel_attach(servo_left, servo_right);

        servo_right.write(0);
        servo_left.write(0);

        break;

case 'l':                                                   //left turn cont.
        wheel_attach(servo_left, servo_right);

        servo_right.write(255);
        servo_left.write(255);

        break;

case 'k':                                                   //reverse cont.
        wheel_attach(servo_left, servo_right);

        servo_right.write(0);
        servo_left.write(250);

        break;


case 'g':                                                   //grab
        claw_attach(servo_claw);
        servo_claw.write(40);   //claw open
        delay(400);
        //claw_detach(servo_claw);

        arm_attach(servo_arm);
        servo_arm.write(90);    //arm up
        delay(150);

        servo_arm.write(65);    //arm down
        delay(400);
        arm_detach(servo_arm);

        wheel_attach(servo_left, servo_right);//move towards block

        servo_right.write(255);
        servo_left.write(0);
        delay(300);
        wheel_detach(servo_left, servo_right);


        wheel_attach(servo_left, servo_right);//shimmy right

        servo_right.write(0);
        servo_left.write(0);
```

```
        delay(100);

        servo_right.write(255);                         //shimmy left
        servo_left.write(255);
        delay(100);

        servo_right.write(255);                         //move in
        servo_left.write(0);
        delay(400);
        wheel_detach(servo_left, servo_right);

        claw_attach(servo_claw);
        servo_claw.write(100);  //claw close
        delay(400);
        //claw_detach(servo_claw);

        arm_attach(servo_arm);
        servo_arm.write(90);    //arm up
        delay(400);

        break;

case 'r':                                               //release
        claw_attach(servo_claw);
        servo_claw.write(100);  //claw close
        delay(400);

        arm_attach(servo_arm);
        servo_arm.write(90);    //arm up
        delay(400);

        servo_arm.write(65);    //arm down
        delay(400);

        servo_claw.write(40);   //claw opem
        delay(800);
        //claw_detach(servo_claw);

        servo_arm.write(90);    //arm up
        delay(400);

        wheel_attach(servo_left, servo_right);//reverse away from blocks
        servo_right.write(0);
        servo_left.write(255);
        delay(800);
        wheel_detach(servo_left, servo_right);

        break;

case 'v':                                       //arm up
        arm_attach(servo_arm);

        servo_arm.write(90);
        delay(150);

        break;

case 'b':                                       //arm down
        arm_attach(servo_arm);

        servo_arm.write(65);
        delay(150);

        break;

case 'n':                                       //claw open
        claw_attach(servo_claw);

        servo_claw.write(30);
        delay(400);

        //claw_detach(servo_claw);
```

```
                        break;

                case 'm':                                       //claw close
                        claw_attach(servo_claw);

                        servo_claw.write(100);
                        delay(400);

                        //claw_detach(servo_claw);

                        break;


                }
        }
}

void wheel_attach(Servo servo_left, Servo servo_right)
{
        servo_left.attach(11);
        servo_right.attach(10);
};
void wheel_detach(Servo servo_left, Servo servo_right)
{
        servo_left.detach();
        servo_right.detach();
};
void arm_attach(Servo servo_arm)
{
        servo_arm.attach(12);
};
void arm_detach(Servo servo_arm)
{
        servo_arm.detach();
};
void claw_attach(Servo servo_claw)
{
        servo_claw.attach(8);
};
void claw_detach(Servo servo_claw)
{
        servo_claw.detach();
};
```