Mechatronics System Design

Design Project Review

Department of Mechanical Engineering

Arduino controlled robotic arm design

To perform pick-n-place material movement

-Submitted by

Raghavendra Rao

Executive Summary

The objective at hand is to use a SainSmart 6-Axis robot arm to pick-and-place colored blocks. Our group has created two levels of requirements to achieve with the robot. Level I is our main objective to achieve while level II is our goal to achieve. Level I involves using an RGB sensor to detect the color of the block. By knowing the color of the block, the robotic arm will be able to pick-n-place a block to and from a known location. Level II involves an Xbox 360 Kinect. The Xbox 360 Kinect allows the system to detect the location and color of a random array of blocks. This allow the system to pick up a block from an unknown location and place it to a known location.

I. Project Functionality Level Descriptions

A. Level 1 Concept

Level 1 represents the minimum target the group has set for itself. This target declares that the system should at a minimum be able to:

- Pick up a block from a known location (pick)
- Sense the color of that block (sense)
- Place the block based on the sensed color (sort)

In order to meet the level 1 requirements, the system shown in Fig. 1 below has been chosen.

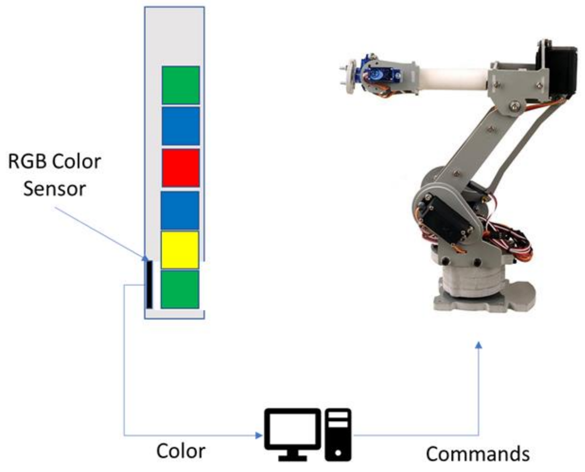


Fig. 1 Representation of component relationships for level 1 system.

This system consists of a chute that can be loaded with multiple blocks and is gravitationally fed. Mounted at the height of the lowest block is an RGB color sensor that sends information about the wavelength of light it sees to the microcontroller. The microcontroller then processes this information to decide the color and sends a command to the arm to grab the bottom block and place it in a location that has been designated for blocks of that color. This process would then repeat until there were no more blocks to sort.

B. Level 2 Concept

We plan to integrate the tasks from the level 1 to a more robust system setup in the level 2. The system uses a camera to detect the object with dynamic data fed by the RGB sensor into the computer. The position and orientation of the block in the form of a 3D point cloud are monitored which in turn actuates the servo. Arduino system is planned to be assimilated with the use of MATLAB. Future scope of working on machine learning and training the arm to detect the color cubes from the vicinity is being worked on by the team. The development of computer vision for the robust functioning of the robotic arm will make use of the Xbox 360 Kinect or similar line of motion-sensing technology to further refine the robotic arm operation by eliminating the noises in the surrounding environment. This vision system will be able to create the 3D point cloud and an overall system diagram for level 2 is shown in Fig. 2 below.

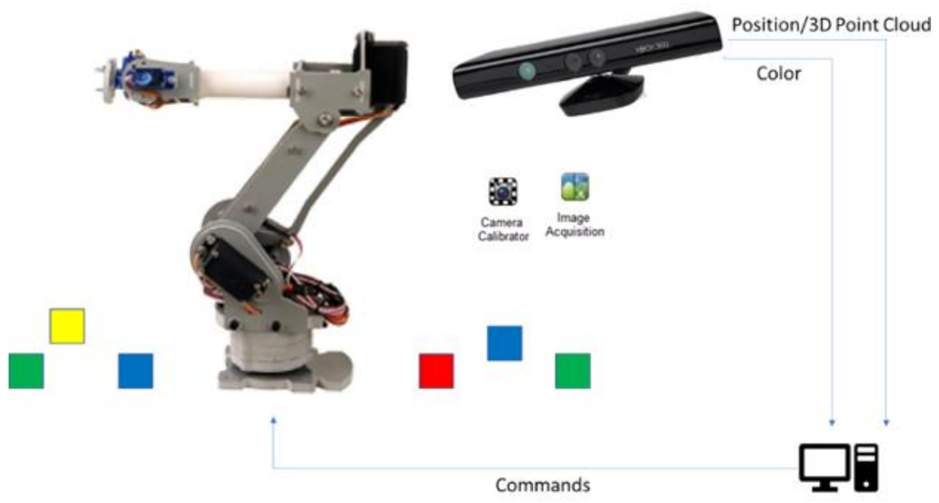


Fig. 2 Representation of component relationships for level 2 system.

II. Job Breakdown and Task Assignment

According to the levels shown above, 5 critical tasks were determined. Four were included as part of level 1 while the only current level 2 task is continued research into the methodology behind integrating the Kinect sensor into the system.

Level 1 Tasks and Designated Team Member(s) Responsible

- Design of block container with physical integration of RGB color sensor
  - Assigned to Cris
- Design of end effector capable of mounting on current robotic arm and operated by servo motor
  - Assigned to Jonathan
- Understanding of all physical connections and layout of servos, sensors, and Arduino
  - Assigned to Mac
- Creation of Arduino code capable of recognizing color blocks and moving them to a designated location based on color
  - Assigned to Noah and Graham

Level 2 Tasks and Designated Team Member(s) Responsible

- Continued research into Level 2 design solution
  - Assigned to Rao and John

III. Level 1 Concept Design

A. Gripper

Different style of grippers for a 9-gram micro servo have been designed, 3D printed, and experimented with to determine if they would be capable of gripping the blocks. The final design operates as a rack and pinion to that pinches shut to grip blocks when the servo rotates. The test confirmed the ability of the final design to do so, this gripper can be seen below in Fig. 3. Additionally, the gripper also features a base plate in which the 9g servo sits so that the center of its grip is aligned with the center of the end of the arm. The slots in the baseplate allow it to be fastened to the end of arm mounting surface. The servo can be friction fit into the base plate or glues for extra security. Part Drawings can be found in Appendix B.

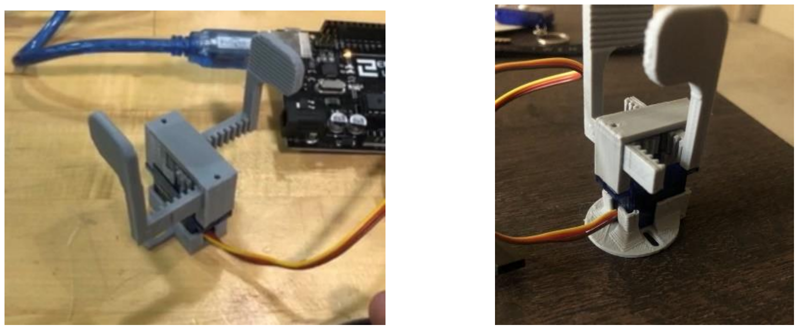


Fig. 3 Images of the end effect gripper that will be used on the robotic arm

B. Block Magazine and Sensor Integration

To reiterate, for our level 1 concept we decided to program a robot that could pick up blocks from a known location and place them in a different location based on their color. In order to move forward with this task, designing a vertical block container was necessary. The 3D printed result can be seen below in Fig. 4. The vertical block container can hold 6 blocks and features an opening for integration of a color sensor. The container component and RGB color sensor is only used for the level 1 concept.

The RGB sensor will be integrated into our robotic system by mounting the sensor to the back of the block magazine. There will be an open hole to allow the sensor to sense the block in the bottom of the magazine. The output from the system will be connected to SDA, SLA, 5V, and ground ports of the Arduino Uno board. The code that is written for the RGB sensor allows the sensor to detect the color blue, green, yellow and red. This was done by having the sensor read in the individual RGB color values for each of these colors. The values range from 0 to 250. But before the sensor starts to read in the RGB color values, the code initializes the sensor to start with values of red, blue and green at zero. This is to recalibrate the sensor every time the sensor is turned on. The code to distinguish the different color values is shown in Appendix A. Finally, the code outputs the color of the block in the command prompt window of the Arduino as red, blue, green or yellow.

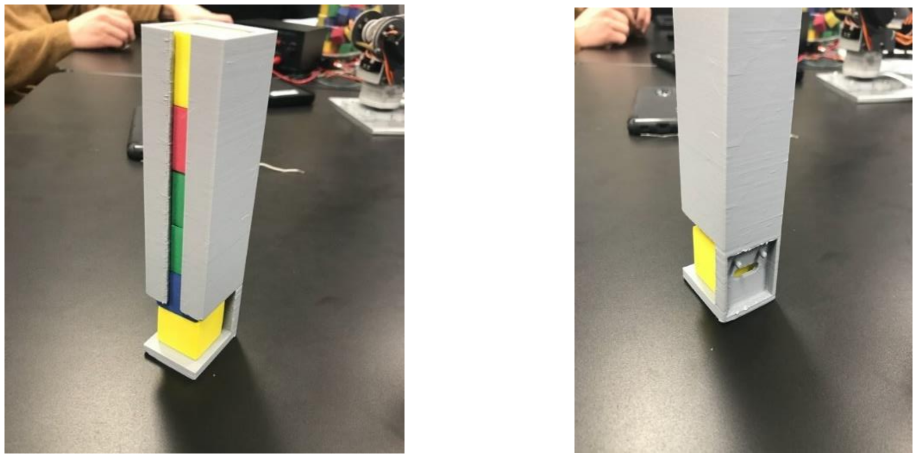


Fig. 4 Images of the block magazine for the robot to pick from for level 1 concept

C. Pick and Place Strategy

To accomplish the goal of the pick and place operation, our team first developed a strategy to use when developing the code. When supplied with an XYZ coordinate, the angular positions of each stepper motor on the robot would be calculated . The main idea is that we just had to define where we wanted the pincher to be and the code would calculate and write the values to the servo motors.

The initial pick and place strategy is outlined by the following steps:

1. Move arm with open pincher to block magazine at a known XYZ coordinate.
2. Close pincher to grab block.
3. Using RGB sensor determine color and input into system.
4. Move arm to known XYZ coordinates of determined color from step 3.
5. Index XYZ coordinate of color being placed to the next place location for that color block.
6. Open pincher to place block.
7. Repeat

There were a few additions that had to be made due to problems noticed in initial testing. First, when moving from the magazine to where it will place the block, each stepper would just power to the angle it needed, the path of the robotic arm was never dealt with. This would cause the possibility of knocking over stacks or the magazine itself. To solve this an intermittent step every time the arm moved from picking to placing and back, the arm needed to retract, then rotate, then extend to desired angles. This prevents an extended arm sweeping across the entire work surface. Next, we found the order in which the motors were powered mattered to ensure that the gripper didn't hit the ground or stacks. The angle of the "humerus" of the arm had to be set first, followed by the "ulna". By extending the arms in this order, the gripper being in contact with the ground was avoided, as well as stacks were always approached from above or the side, never from under risking knocking the stacks over. Lastly, it was found that a horizontal gripper wouldn't always work for certain placing arrangements like pyramids etc. To solve this, whenever the XYZ matrices which told the arm where to place each successive color block were made, an extra column holding the desired gripper angle to place that block was input.

These changes led to the final level I, also feasible for level II, concept pick and place strategy below:

1. Rotate base servo to angle needed to pick up block from magazine.
2. Power elbow and wrist servos to angle needed to pick up block from magazine.
3. Power shoulder servo to angle needed to pick up block from magazine
4. Close pincher to grab block.
5. Using RGB sensor determine color and input into system, find for XYZ-Wrist angle place coordinates.
6. Solve for arm angles for placing.
7. Retract shoulder servo to neutral position.
8. Retract elbow servo to neutral position.
9. Rotate base servo to angle needed to place block.
10. Power elbow and wrist servos to angle needed to place block.
11. Power shoulder servo to angle needed to place block.
12. Index XYZ-Wrist angle coordinate of color being placed to the next place location for that color block.
13. Open pincher to place block.

14. Retract shoulder servo to neutral position.
15. Retract elbow servo to neutral position.
16. Repeat.

D. Programing Code

The team was able to find code online written to control a 6-axis robotic arm like the one used for this project. The code has inputs for the lengths of each member of the arm so that it can be modified to the size of the arm as necessary. The inputs to the code are the XYZ coordinates the end effect of the robot will arrive at, and the speed at which it will make those movements. The Y and Z coordinates can only be positive, which results in 180˚ of movement on the tabletop in front of the robotic arm. The Arduino has its own programming language but can also interface through MATLAB which is preferable to our team as we already know how to code in MATLAB. MATLAB's toolbox when working with Arduinos is fairly extensive, even giving all possible pins and output options through methods() command. The arm is already outfitted with a Servo Shield which allows for additional servos to be mounted to the Arduino UNO, which is necessary as we plan to use an additional servo to actuate the end effect.

The full sample code can be found in Appendix A. A brief overview of the code is as follows. The program begins by calling in libraries and definitions for the servos and RGB color sensor. Then the program sets up necessary system addresses and data locations for the RGB sensor. The gain control and timing of the sensor is established next, with the if statements for displaying the color output coming directly after. With the preliminary setup complete for the sensor and servos, the void setup section of the code is used to start the serial output for the sensor and attach the servos to the Arduino pin inputs. An initial position for the arm is also initiated here. Within the void loop section, the sensor is actuated and the if statements for color sensing are repeated. These if statements are now used to determine the drop off location for the block based on its color. The code then begins to run the positioning program to pick up the block and drop it off at the necessary location based on color. Lastly, the program will read and print the location of the arm's servos and start a 5 second delay to ensure the arm's movement can finish before reading the color of the next block.

IV. Robotics Field Overview and Relevance to Project

A. Computer Vision

Computer vision uses Artificial Intelligence that trains computers to perceive and interpret the visual world. Vision-guided robots are an integral part of the robotic industry. The advancements in the automation have led to disruptive innovation in the robotics. The robots in this generation of Industrial Revolution 4.0 are more smart, robust and reliable. Using digital images of the surrounding from the camera is integrated into the deep learning model to accurately identify and classify the objects in real-time.

The Kinect used in the robotic arm project uses an image processing technique to convert image coordinates from camera to pixel frame. Camera calibration consists of the estimation of the model to find the intrinsic and extrinsic parameters of the camera. Tsai's model is used to calibrate the camera using the MATLAB code. This technique is used to convert image to coordinates and store the data for identifying the exact location of the colored cubes. For placing multiple cubes at different locations or to arrange in a pattern, we can make use of stereo vision using the least square based approach. Different techniques like correlation based and feature based methods can be used depending on the requirements of a sparse disparity mapping of the surrounding.

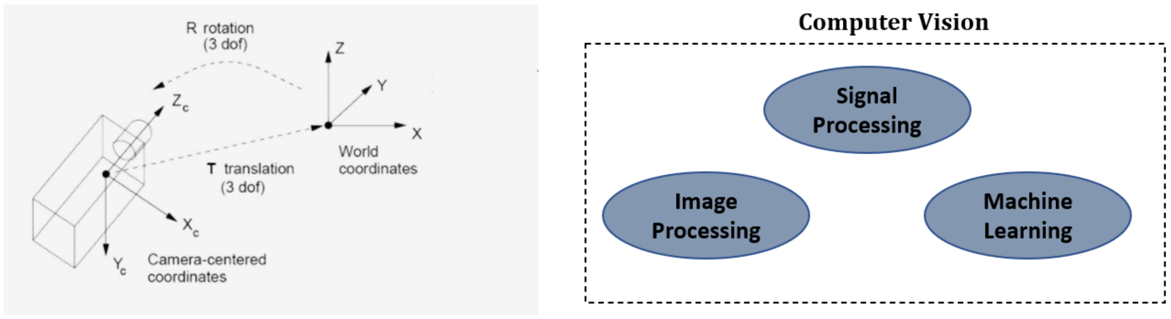


Fig. 5 Image processing and computer vision basic model.

The position and dimension of the block is stored in the form of a 3D point cloud. The data signal from the post-processing controls the actuation of the servo to perform the necessary operation.

B. Sensor Fusion

Sensor fusion is the technique of a combination of sensory data or data derived from disparate sources resulting in accurate, complete and reliable information. It is an important technique while using multipurpose and multi-tasking robots performing simple or complex tasks.

The Kinect 360 camera and the RGB sensor must be integrated using the sensor fusion technique. Future scope in the project might also use a proximity sensor to determine the optimal distance of the arm from the cube. We intend to use Kalman filter in the sensor fusion to refine the positioning data of the cubes. The integration of the sensor information from the Arduino can be read and complied within the MATLAB code.

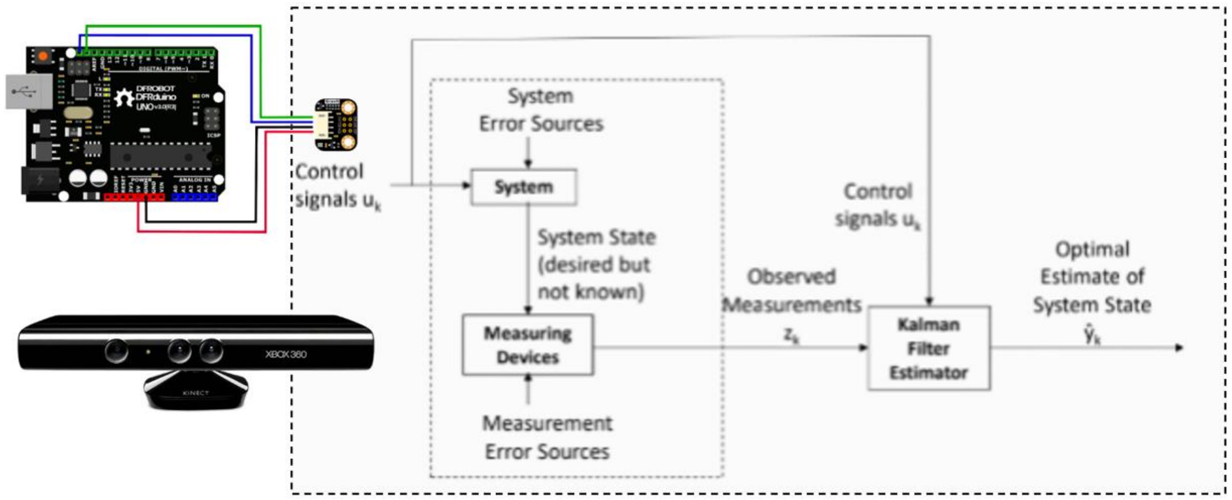

Control signals u_k
System Error Sources
System
System State (desired but not known)
Measuring Devices
Measurement Error Sources
Control signals u_k
Observed Measurements z_k
Kalman Filter Estimator
Optimal Estimate of System State $\hat{y}_k$
XBOX 360

Fig. 6 Sensor fusion model with Kalman filter

C. Machine Learning

Machine learning and AI is set to disrupt and enhance the capabilities of the robotic industry to its full potential. The scope of machine learning includes Vision, Motion Control, Data and Integration. The prime example of machine learning in the robotic application is in picking and placing different part types in a warehouse. The automation is in the transformative path and its impact is evident all around us.

The robotic arm project can be made robust by implementing motion planning using trapezoidal decomposition and probabilistic approaches. This helps in the movement of the arm around the robotic axis to find and pick the colored cubes in the vicinity. The arm motion control can be made reliable using various control systems applications. The statistics and machine learning toolbox in MATLAB provide functions and apps to describe, analyze and model the data. The robot is trained using deep learning technique where approximately 100 images of the robot surrounding and cubes are fed. The computer vision recognizes the cube of a certain color from the trained dataset and reaches out to pick the cube. The toolbox provides supervised and unsupervised machine learning algorithms for computations on data sets. Further stretch in the project can make use of neural networks to provide enough stimuli for identifying the colored cubes without the use of the RGB sensor. Faster R-CNN can be used as a region-based approach to object detection which can be used to formulate a set number of regression models.

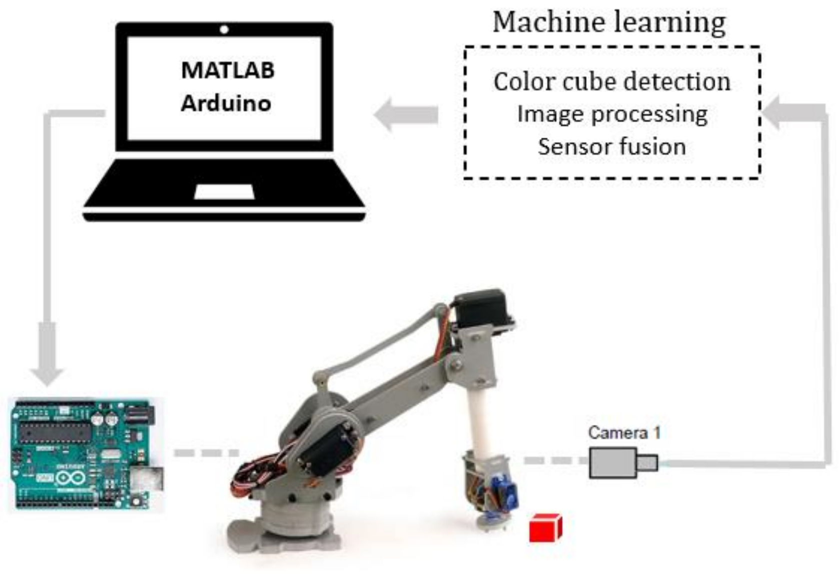


Fig. 7 Machining learning integration diagram

V. Purchase Component List

The components needed for the team to proceed with these concepts are listed below in Table 1. The team already has access to other various components needed for the design that are not listed in this table. The RGB sensor and Zoom for Kinect are related to system functionality and the wires are to have on hand to physical connect the components to the Arduino module.

Level	Item	Part Number	Seller	Cost
1	Adafruit RGB Color Sensor w/ IR Filter	TCS34725	Amazon	$9.81
1	Elegoo Jumper Wires Ribbon Cables	EL-CP-004	Amazon	$6.98
2	Zoom for Kinect – Xbox 360	B0050SYS5A	Amazon	$18.99

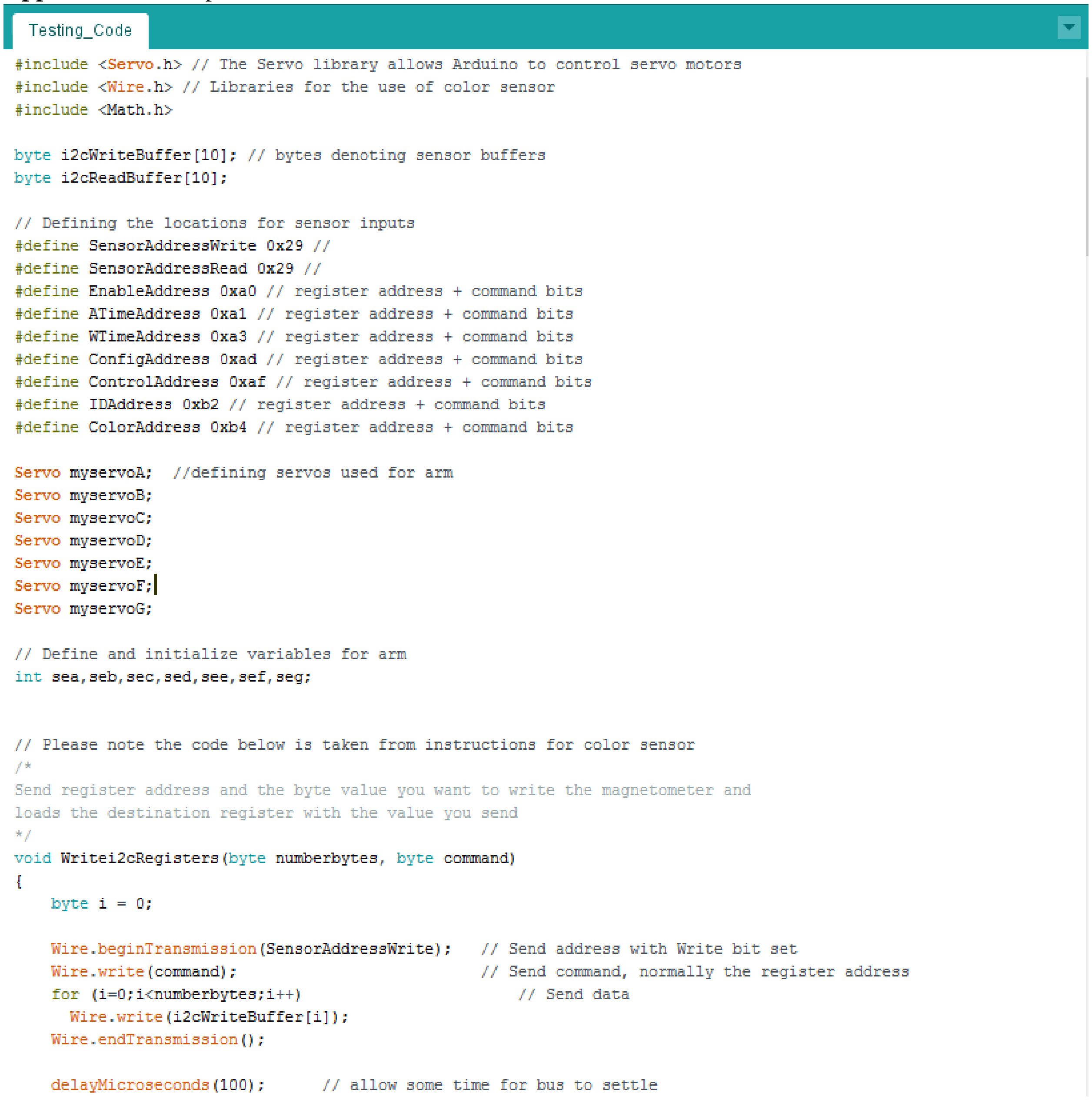
Table 1 Detailed information about the requested components. A link is included in an electronic version of this document.

References

Aggarwal, J.K., Wang, Y.F., Sensor data fusion in robotics system-https://doi.org/10.1016/B978-0-12-012739-9.50015-X
https://www.cs.toronto.edu/~kriz/cifar.html
https://www.sas.com/content/dam/SAS/en_us/doc/whitepaper1/deep-learning-with-sas-109610.pdf
http://mcube.mit.edu/
https://vision.princeton.edu/projects/2017/arc/
https://www.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html

Appendix

Appendix A – Sample Arduiono Code



```
#include <Servo.h> // The Servo library allows Arduino to control servo motors
#include <Wire.h> // Libraries for the use of color sensor
#include <Math.h>

byte i2cWriteBuffer[10]; // bytes denoting sensor buffers
byte i2cReadBuffer[10];

// Defining the locations for sensor inputs
#define SensorAddressWrite 0x29 //
#define SensorAddressRead 0x29 //
#define EnableAddress 0xa0 // register address + command bits
#define ATimeAddress 0xa1 // register address + command bits
#define WTimeAddress 0xa3 // register address + command bits
#define ConfigAddress 0xad // register address + command bits
#define ControlAddress 0xaf // register address + command bits
#define IDAddress 0xb2 // register address + command bits
#define ColorAddress 0xb4 // register address + command bits

Servo myservoA;  //defining servos used for arm
Servo myservoB;
Servo myservoC;
Servo myservoD;
Servo myservoE;
Servo myservoF;
Servo myservoG;

// Define and initialize variables for arm
int sea,seb,sec,sed,see,sef,seg;


// Please note the code below is taken from instructions for color sensor
/*
Send register address and the byte value you want to write the magnetometer and
loads the destination register with the value you send
*/
void Writei2cRegisters(byte numberbytes, byte command)
{
    byte i = 0;

    Wire.beginTransmission(SensorAddressWrite);   // Send address with Write bit set
    Wire.write(command);                          // Send command, normally the register address
    for (i=0;i<numberbytes;i++)                       // Send data
      Wire.write(i2cWriteBuffer[i]);
    Wire.endTransmission();

    delayMicroseconds(100);      // allow some time for bus to settle
```

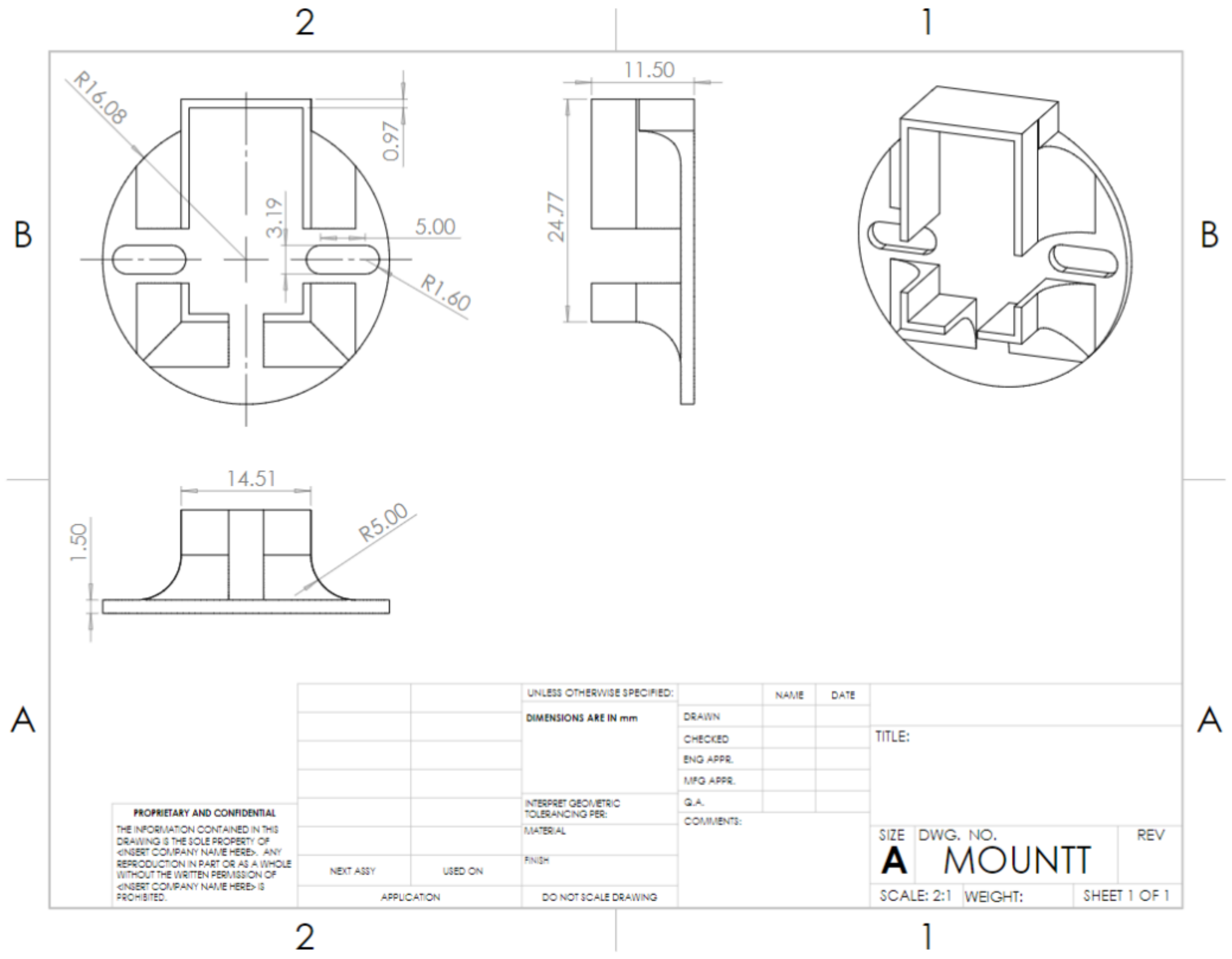
```
/*
Send register address to this function and it returns byte value
for the magnetometer register's contents
*/
byte Readi2cRegisters(int numberbytes, byte command)
{
   byte i = 0;

    Wire.beginTransmission(SensorAddressWrite);   // Write address of read to sensor
    Wire.write(command);
    Wire.endTransmission();

    delayMicroseconds(100);      // allow some time for bus to settle

    Wire.requestFrom(SensorAddressRead,numberbytes);   // read data
    for(i=0;i<numberbytes;i++)
      i2cReadBuffer[i] = Wire.read();
    Wire.endTransmission();

    delayMicroseconds(100);      // allow some time for bus to settle
}

void init_TCS34725(void)
{
  i2cWriteBuffer[0] = 0x10;
  Writei2cRegisters(1,ATimeAddress);    // RGBC timing is 256 - contents x 2.4mS =
  i2cWriteBuffer[0] = 0x00;
  Writei2cRegisters(1,ConfigAddress);   // Can be used to change the wait time
  i2cWriteBuffer[0] = 0x00;
  Writei2cRegisters(1,ControlAddress);  // RGBC gain control
  i2cWriteBuffer[0] = 0x03;
  Writei2cRegisters(1,EnableAddress);    // enable ADs and oscillator for sensor
}

void get_TCS34725ID(void)
{
  Readi2cRegisters(1,IDAddress);
  if (i2cReadBuffer[0] = 0x44)
    Serial.println("TCS34725 is present");
  else
    Serial.println("TCS34725 not responding");
}
```

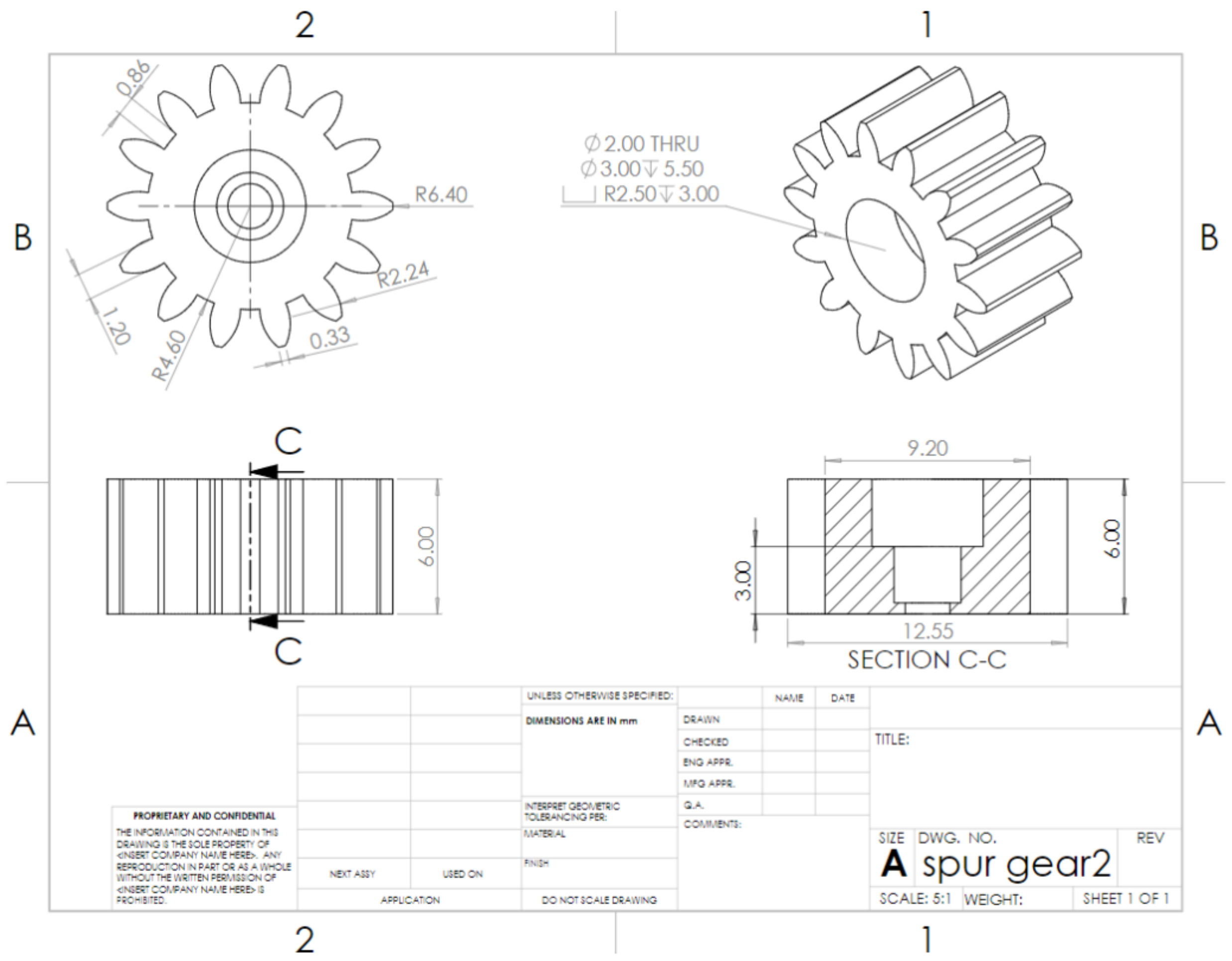
```
/*
Reads the register values for clear, red, green, and blue.
*/
void get_Colors(void)
{
  unsigned int clear_color = 0;
  unsigned int red_color = 0;
  unsigned int green_color = 0;
  unsigned int blue_color = 0;

  Readi2cRegisters(8,ColorAddress);
  clear_color = (unsigned int)(i2cReadBuffer[1]<<8) + (unsigned int)i2cReadBuffer[0];
  red_color = (unsigned int)(i2cReadBuffer[3]<<8) + (unsigned int)i2cReadBuffer[2];
  green_color = (unsigned int)(i2cReadBuffer[5]<<8) + (unsigned int)i2cReadBuffer[4];
  blue_color = (unsigned int)(i2cReadBuffer[7]<<8) + (unsigned int)i2cReadBuffer[6];

 // Basic RGB color differentiation can be accomplished by comparing the values and the largest reading will be
 // the prominent color

  if((red_color>blue_color) && (red_color>green_color) && (clear_color<50000))
    Serial.println("detecting red");
  else if((green_color>blue_color) && (green_color>red_color)&& (clear_color>14000))
    Serial.println("detecting green");
  else if((blue_color>red_color) && (blue_color>green_color))
    Serial.println("detecting blue");
  else if(clear_color>50000)
    Serial.println("detecting yellow");
  else if(clear_color<14000)
    Serial.println("Hopper Is Empty Please Reload");
  else
    Serial.println("color not detectable");
}

void setup() {

   Wire.begin();
  Serial.begin(9600);  // start serial for output
  init_TCS34725();
  get_TCS34725ID();     // get the device ID for color sensor, and test connection
```

```
// attach all servos to the appropriate pins
  myservoA.attach(3); // Upper Arm (A): Port 3
  myservoB.attach(5); // Waist (B): Port 5
  myservoC.attach(6); // Forearm (C): Port 6
  myservoD.attach(9); // Rotational Forearm (D): Port 9
  myservoE.attach(10); // Wrist (E): Port 10
  myservoF.attach(11); // Wrist Rotation (F): Port 11
  myservoG.attach(13); // Gripper (G): Port 11

// write initial start possition to arm
  myservoA.write(30);
  myservoB.write(45);
  myservoC.write(65);
  myservoD.write(90);
  myservoE.write(70);
  myservoF.write(90);
  myservoG.write(5);

}

void loop() {

   get_Colors(); //determine color of box or if empty and output info to determine position
    delay(1000);


  unsigned int clear_color = 0;
  unsigned int red_color = 0;
  unsigned int green_color = 0;
  unsigned int blue_color = 0;

  Readi2cRegisters(8,ColorAddress);
  clear_color = (unsigned int)(i2cReadBuffer[1]<<8) + (unsigned int)i2cReadBuffer[0];
  red_color = (unsigned int)(i2cReadBuffer[3]<<8) + (unsigned int)i2cReadBuffer[2];
  green_color = (unsigned int)(i2cReadBuffer[5]<<8) + (unsigned int)i2cReadBuffer[4];
  blue_color = (unsigned int)(i2cReadBuffer[7]<<8) + (unsigned int)i2cReadBuffer[6];

  // Create variable to control drop locations based on robotic arm waist
  int DropLocation = 0;
```

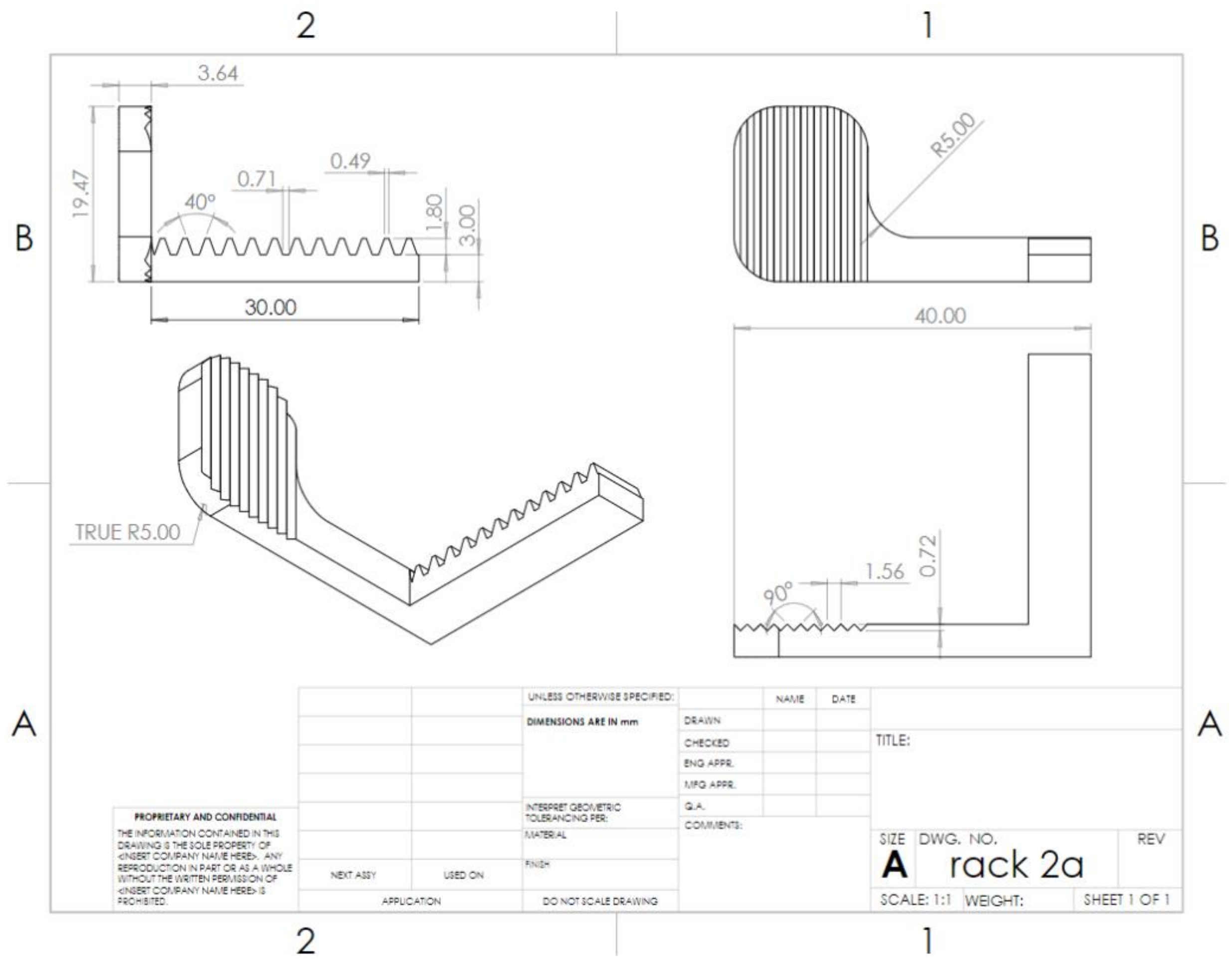
```
//Movement for different colored boxes to designated locations
if((red_color>blue_color) && (red_color>green_color) && (clear_color<50000))
    DropLocation = 50; // if red
  else if((green_color>blue_color) && (green_color>red_color)&& (clear_color>14000))
     DropLocation = 90; // if green
  else if((blue_color>red_color) && (blue_color>green_color))
     DropLocation = 130; // if blue
  else if(clear_color>50000)
     DropLocation = 170; // if yellow
  else (clear_color<14000)
     ;DropLocation = 10; // hopper empty



  myservoA.write(30); // pickup position
  myservoB.write(10);
  myservoC.write(65);
  myservoD.write(90);
  myservoE.write(70);
  myservoF.write(90);
  myservoG.write(120);

   delay(1000); // delay 1s

  myservoA.write(60); // middle location
  myservoB.write(90);
  myservoC.write(50);
  myservoD.write(90);
  myservoE.write(120);
  myservoF.write(90);
  myservoG.write(120);

   delay(1000); // delay 1s

  myservoA.write(30); // drop point
  myservoB.write(DropLocation);
  myservoC.write(65);
  myservoD.write(90);
  myservoE.write(70);
  myservoF.write(90);
  myservoG.write(5);

   delay(1000); // delay 1s
  myservoA.write(60); // middle location
  myservoB.write(90);
  myservoC.write(50);
  myservoD.write(90);
  myservoE.write(120);
  myservoF.write(90);
  myservoG.write(5);

   delay(1000); // delay 1s


// write output from servo location to command window
   sea=myservoA.read();
   seb=myservoB.read();
   sec=myservoC.read();
   sed=myservoD.read();
   see=myservoE.read();
   sef=myservoF.read();
   seg=myservoG.read();


   Serial.print(sea);
   Serial.print("\t");
   Serial.print(seb);
   Serial.print("\t");
   Serial.print(sec);
   Serial.print("\t");
   Serial.print(sed);
   Serial.print("\t");
   Serial.print(see);
   Serial.print("\t");
   Serial.print(sef);
   Serial.print("\t");
   Serial.print(seg);
   Serial.print("\t");
   Serial.print("\t");

   delay(5000); //5 second delay in order to allow arm to place block before reading new color
}
```

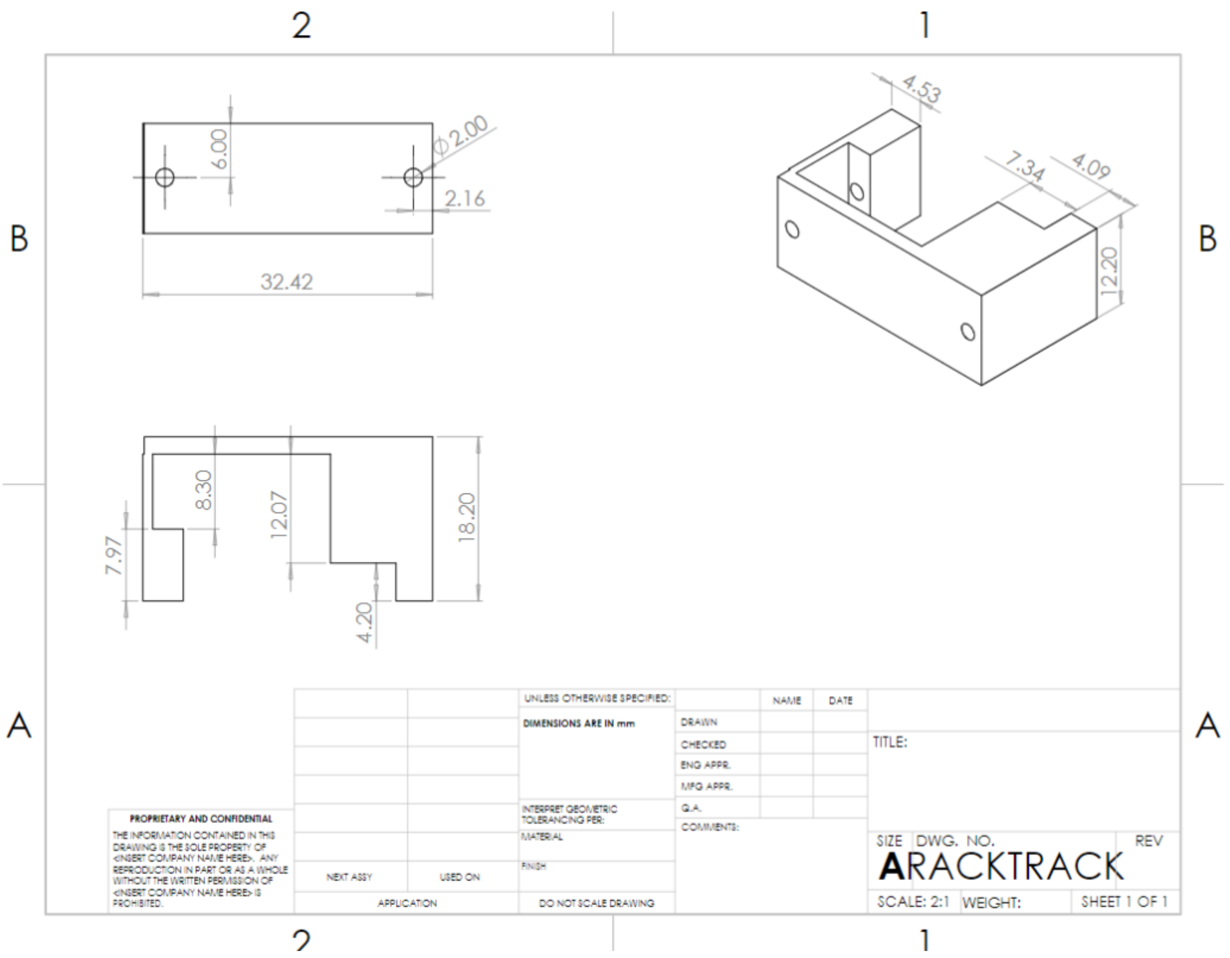
Appendix B – CAD Drawings


R16.08
0.97
3.19
5.00
R1.60
11.50
24.77
14.51
1.50
R5.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN mm
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
SIZE
A
DWG. NO.
MOUNTT
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
NEXT ASSY
USED ON
APPLICATION
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.



2
1
B
A
0.86
R6.40
R2.24
1.20
R4.60
0.33
⌀2.00 THRU
⌀3.00 ↧ 5.50
⌴ R2.50 ↧ 3.00
C
C
6.00
9.20
3.00
6.00
12.55
SECTION C-C
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN mm
NAME
DATE
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
TITLE:
SIZE
DWG. NO.
REV
A
spur gear2
SCALE: 5:1
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.


2
1
B
A
3.64
19.47
0.71
0.49
40°
1.80
3.00
30.00
R5.00
40.00
TRUE R5.00
1.56
0.72
90°
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN mm
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
NAME
DATE
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
SIZE
DWG. NO.
REV
A
rack 2a
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.


2
1
B
A
6.00
⌀2.00
2.16
32.42
4.53
7.34
4.09
12.20
8.30
12.07
18.20
7.97
4.20
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN mm
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
NAME
DATE
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
SIZE
DWG. NO.
REV
A
RACKTRACK
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.