Alexander Gruhl May 4, 2021

Robot Manipulator Conveyor Belt Project

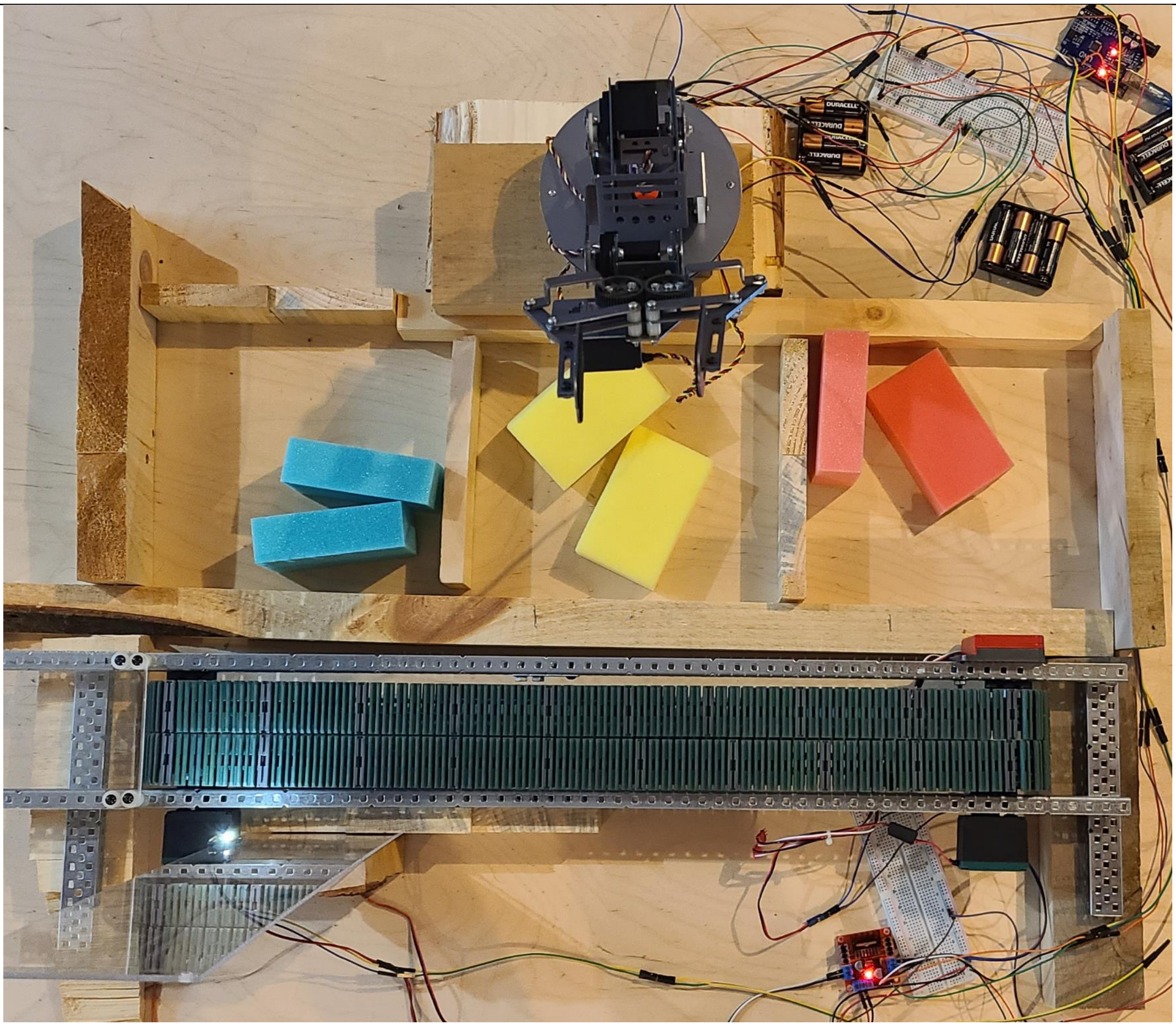
Description:

The goal of this project is to create a robot manipulator that sorts objects by colour sent to it using a conveyor belt. This setup is like one that may be found in an industrial setting where a robot manipulator is used to pickup objects and sort them. The major components used in this project are a four degrees of freedom robot manipulator with four revolute joints and a claw end effector, a conveyor belt made from the VEX tank robot kit, an Arduino, an Adafruit TCS34725 colour sensor, and a L298N Motor Driver.

The robot uses servo motors of varying size for its joints, an aluminum frame for its links, and is controlled by the Arduino. Each servo is (except the claw servo and the fourth joint servo) have a separate battery 6V battery pack powering them because the servo motors draw a lot of current which quickly drains the batteries of current. Therefore, by powering each servo separately, the batteries do not need to be replaced as quickly. The conveyor belt is made of VEX tank treads and two VEX DC motors from the VEX tank kit. The DC motors are controlled by the L298N Motor Driver which is controlled by the Arduino. When the Adafruit TCS34725 colour sensor detects the colour of an object in front of it, the DC motors are powered on to move the conveyor belt. The conveyor belt movement moves the object on it from in front of the colour sensor to the robot manipulator pickup location. In this project the objects that are picked up are sponges with colours red, blue, and yellow. Sponges are used as they are in a rectangular shape which fits nicely into the robot claw (end effector) and have a good texture for being picked up. Once the sponge is in the pickup location, the DC motors turn off, the robot manipulator picks up the sponge, and the robot manipulator drops the sponge in the bin associated with the sponge's colour.

This project uses MATLAB solve the Denavit-Hartenburg parameter calculations as the DH convention is a widely used and relatively easy method for calculating the end effector of robot manipulators given the theta angles of the joints (forward kinematics) or calculating the theta angles given the end effector x, y, z, and orientation (phi) relative to the base axis (inverse kinematics). Both forward and inverse kinematics have their own methods to solving the calculations, and with the materials available to me, the geometric solution method for inverse kinematics is the only viable option (I do not have velocity calculators or encoders for the servo motors). Also, since the robot manipulator used in this project has four revolute joints and four degrees of freedom, inverse kinematics is very difficult using the geometric method, so forward kinematics was used. Examples of inverse kinematics calculations can be seen in the project with both a six degree of freedom robot and a three degrees of freedom robot in the GitHub folder.

Setup of Project:



Setup of robot manipulator conveyor belt project. In the top center is the robot manipulator. In the top right is the battery holders, wiring, and the Arduino. The green object at the bottom is the conveyor belt. In between the conveyor belt and robot manipulator are the bins. The bins have sponges of their assigned colour in them (left bin is for blue objects and has blue sponges in it). On the bottom right is the L298N Motor Driver and on the bottom left (the light) is the Adafruit TCS34725 colour sensor.

Note: In the code the bins are inverted from this view (ex. left bin for blue object called right bin in code) because code is from the perspective of the side of the robot manipulator.

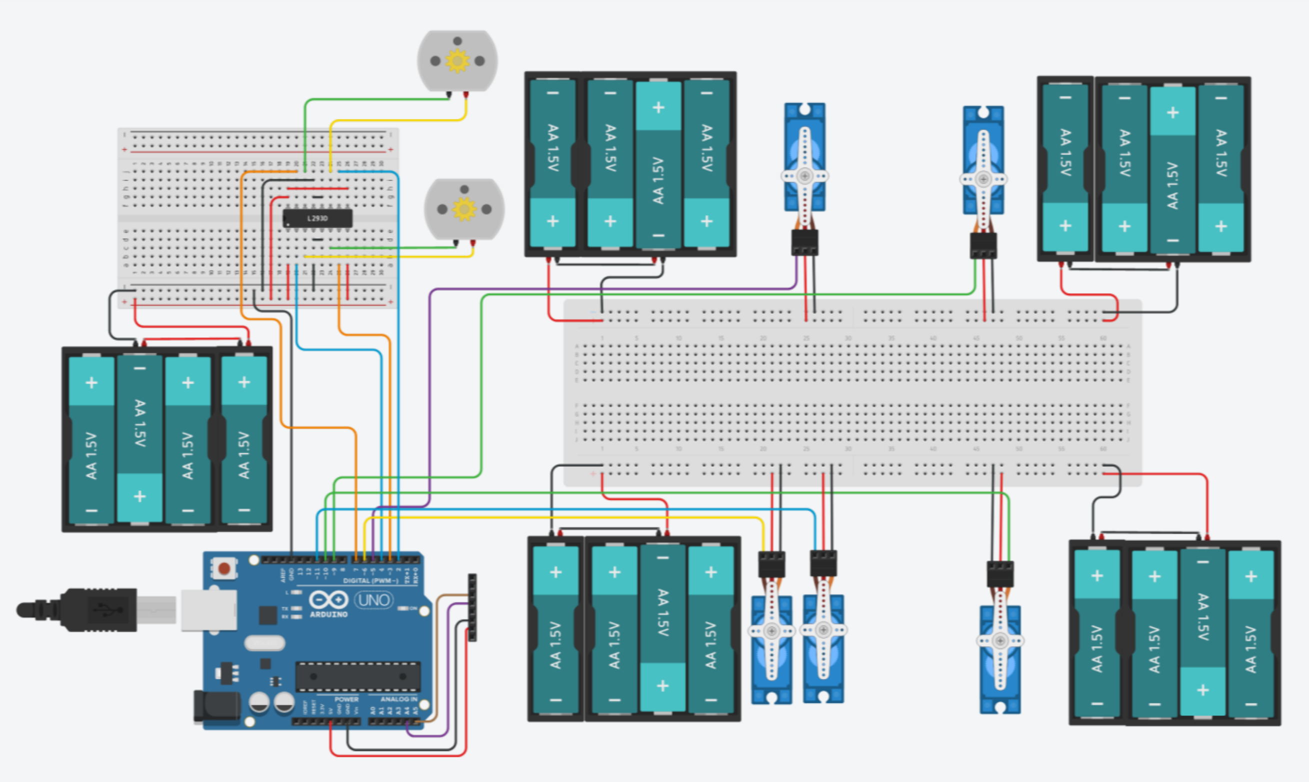
Robot Manipulator

Using map() to Adjust Real World Theta Angle to Servo Motor Angle:

The Arduino map() function is used to convert the real world theta angle (with respect to each frame's axis) to the angle the servo motor needs to move to in order to achieve going to the real world angle [1].

	Joint 1	Joint 2	Joint 3	Joint 4
Real World Angle	a) 0° -> 90° b) 0 -> -90	0° -> 180°	0° -> -180°	90° -> -90°
Servo Angle	a) 90° -> 0° b) 90° -> 180°	0° -> 180°	180° -> 0°	180° -> 0°

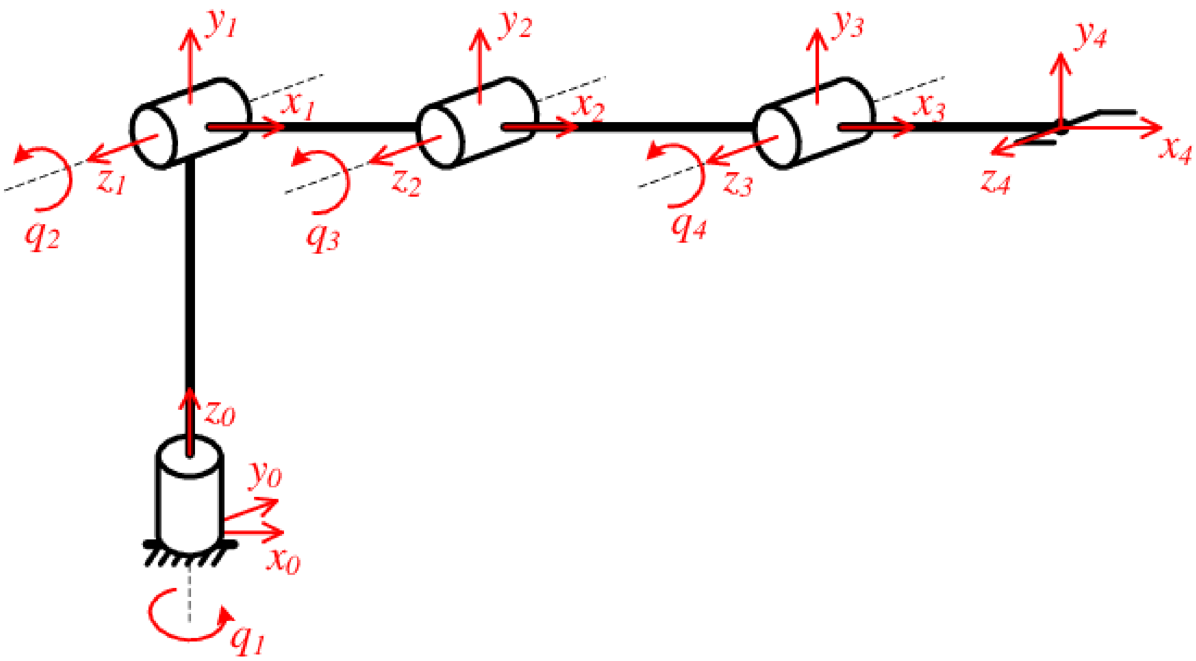
Wiring Diagram for Project:



Note 1: The eight pin header is supposed to represent the wiring of the Adafruit TCS34725 colour sensor. The colour sensor is only seven pins, so the bottommost pin is ignored.

Note 2: Wiring of L298N Motor Driver and Adafruit TCS34725 colour sensor based off of wiring instructions websites [2][3].

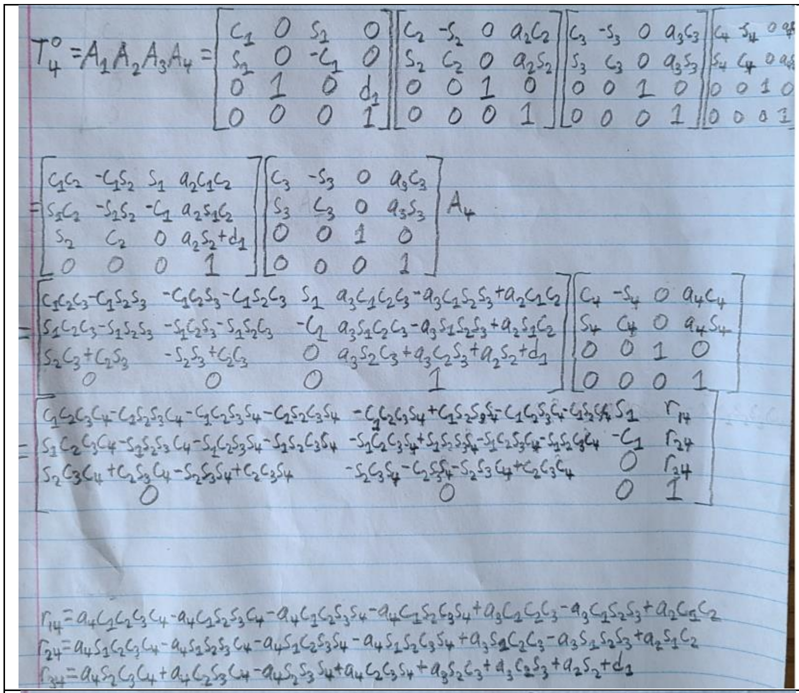
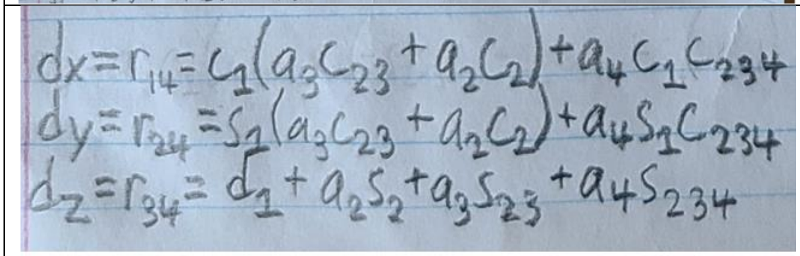
DH Calculations:



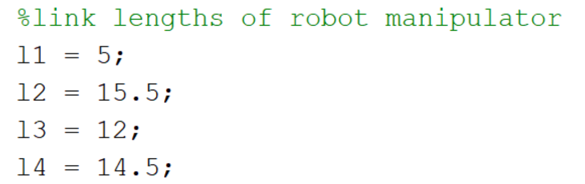
DH model of the 4 DOF robot manipulator in project [4]

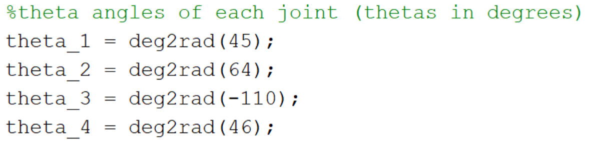
Link	a_i	α_i	d_i	θ_i
1	0	90°	d_1	θ_1
2	a_2	0°	0	θ_2
3	a_3	0°	0	θ_3
4	a_4	0°	0	θ_4

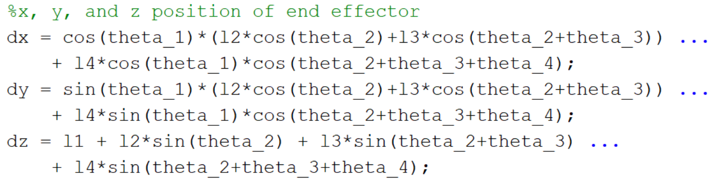
Table of the DH parameters of the 4 DOF robot manipulator. DH parameters assigned to each link of the manipulator. Parameter a_i is the displacement between the center of frame i-1 and frame i, measured in the x_i axis. Parameter α_i is the angle of rotating axis z_{i-1} to match z_i, rotating around x_i. To find the positive direction of α_i, the right-handed rule is used. Parameter d_i is the displacement between the center of frame i-1 and frame i, measured in the z_{i-1} direction. Parameter θ_i is the rotation angle needed to get axis x_{i-1} to match with axis x_i, by rotating around axis z_{i-1}. Also, if joint i-1 is revolute, then any rotation from the revolute joint is included in θ_i because the rotation is around axis z_{i-1}.

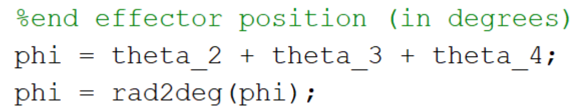
 $T_4^0 = A_1A_2A_3A_4 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_3 & -s_3 & 0 & a_3c_3 \\ s_3 & c_3 & 0 & a_3s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_4 & -s_4 & 0 & a_4 \\ s_4 & c_4 & 0 & a_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $= \begin{bmatrix} c_1c_2 & -c_1s_2 & s_1 & a_2c_1c_2 \\ s_1c_2 & -s_1s_2 & -c_1 & a_2s_1c_2 \\ s_2 & c_2 & 0 & a_2s_2+d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_3 & -s_3 & 0 & a_3c_3 \\ s_3 & c_3 & 0 & a_3s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} A_4$ $= \begin{bmatrix} c_1c_2c_3-c_1s_2s_3 & -c_1c_2s_3-c_1s_2c_3 & s_1 & a_3c_1c_2c_3-a_3c_1s_2s_3+a_2c_1c_2 \\ s_1c_2c_3-s_1s_2s_3 & -s_1c_2s_3-s_1s_2c_3 & -c_1 & a_3s_1c_2c_3-a_3s_1s_2s_3+a_2s_1c_2 \\ s_2c_3+c_2s_3 & -s_2s_3+c_2c_3 & 0 & a_3s_2c_3+a_3c_2s_3+a_2s_2+d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_4 & -s_4 & 0 & a_4c_4 \\ s_4 & c_4 & 0 & a_4s_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $= \begin{bmatrix} c_1c_2c_3c_4-c_1s_2s_3c_4-c_1c_2s_3s_4-c_1s_2c_3s_4 & -c_1c_2c_3s_4+c_1s_2s_3s_4-c_1c_2s_3c_4-c_1s_2c_3c_4 & s_1 & r_{14} \\ s_1c_2c_3c_4-s_1s_2s_3c_4-s_1c_2s_3s_4-s_1s_2c_3s_4 & -s_1c_2c_3s_4+s_1s_2s_3s_4-s_1c_2s_3c_4-s_1s_2c_3c_4 & -c_1 & r_{24} \\ s_2c_3c_4+c_2s_3c_4-s_2s_3s_4+c_2c_3s_4 & -s_2c_3s_4-c_2s_3s_4-s_2s_3c_4+c_2c_3c_4 & 0 & r_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $r_{14} = a_4c_1c_2c_3c_4 - a_4c_1s_2s_3c_4 - a_4c_1c_2s_3s_4 - a_4c_1s_2c_3s_4 + a_3c_1c_2c_3 - a_3c_1s_2s_3 + a_2c_1c_2$ $r_{24} = a_4s_1c_2c_3c_4 - a_4s_1s_2s_3c_4 - a_4s_1c_2s_3s_4 - a_4s_1s_2c_3s_4 + a_3s_1c_2c_3 - a_3s_1s_2s_3 + a_2s_1c_2$ $r_{34} = a_4s_2c_3c_4 + a_4c_2s_3c_4 - a_4s_2s_3s_4 + a_4c_2c_3s_4 + a_3s_2c_3 + a_3c_2s_3 + a_2s_2 + d_1$	DH calculations of the transformation matrix of the robot manipulator. A transformation matrix is used to calculate the position and orientation of one frame in comparison to another frame. This transformation matrix calculates the position and orientation of frame 4 relative to frame 0. Note: c_i and s_i stand for cosine(i) and sine(i), respectively
 $dx = r_{14} = c_1(a_3c_{23} + a_2c_2) + a_4c_1c_{234}$ $dy = r_{24} = s_1(a_3c_{23} + a_2c_2) + a_4s_1c_{234}$ $d_z = r_{34} = d_1 + a_2s_2 + a_3s_{23} + a_4s_{234}$	The x, y, and z position of frame 4 relative to frame 0. These values correspond to the first three values of the transformation matrix's fourth column (from top of the matrix to the bottom). Note 1: c_{ij} and s_{ij} stand for cosine(i +j) and sine(i + j), respectively Note 2: c_{ijk} and s_{ijk} stand for cosine(i +j + k) and sine(i + j + k), respectively

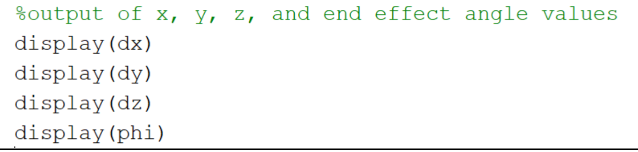
MATLAB Code:

 `%link lengths of robot manipulator`
`l1 = 5;`
`l2 = 15.5;`
`l3 = 12;`
`l4 = 14.5;`
	Lengths of the links for the robot manipulator in centimeters.

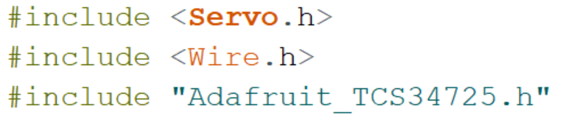
 `%theta angles of each joint (thetas in degrees)`
`theta_1 = deg2rad(45);`
`theta_2 = deg2rad(64);`
`theta_3 = deg2rad(-110);`
`theta_4 = deg2rad(46);`
	Input of the theta angles for the Denavit-Hartenbug forward kinematics. Theta angles inputted in degrees and are converted to radians in MATLAB.

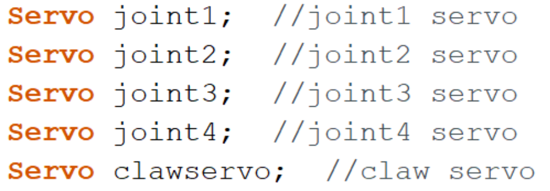

`%x, y, and z position of end effector`
`dx = cos(theta_1)*(l2*cos(theta_2)+l3*cos(theta_2+theta_3)) ...`
`    + l4*cos(theta_1)*cos(theta_2+theta_3+theta_4);`
`dy = sin(theta_1)*(l2*cos(theta_2)+l3*cos(theta_2+theta_3)) ...`
`    + l4*sin(theta_1)*cos(theta_2+theta_3+theta_4);`
`dz = l1 + l2*sin(theta_2) + l3*sin(theta_2+theta_3) ...`
`    + l4*sin(theta_2+theta_3+theta_4);`
	End effector x, y, and z coordinates of the robot manipulator. Equations are determined from the fourth column of the transformation matrix.

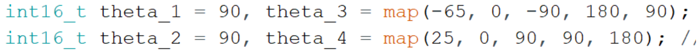

`%end effector position (in degrees)`
`phi = theta_2 + theta_3 + theta_4;`
`phi = rad2deg(phi);`
	End effector angle of the robot manipulator (called phi in this calculation). Phi determined from adding the theta angles of all the joints and is in degrees.

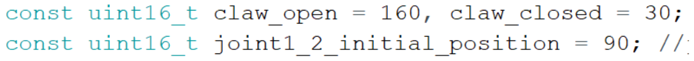

`%output of x, y, z, and end effect angle values`
`display(dx)`
`display(dy)`
`display(dz)`
`display(phi)`
	Displays the output of the robot manipulator's end effector x, y, z, and phi positions in the command window.

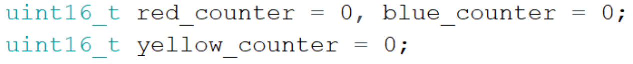
Arduino Code:


`#include <Servo.h>`
`#include <Wire.h>`
`#include "Adafruit_TCS34725.h"`
	<Servo.h> is used to allow the Arduino to control the servo motors of the robot manipulator. <Wire.h> is used to allow the Arduino to communicate with the TCS34725 colour sensor. "Adafruit_TCS34725.h" implements the functions needed to calculate the input readings of the TCS34725 colour sensor.

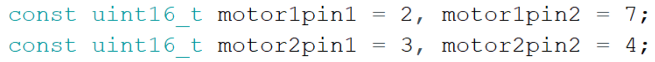

`Servo joint1;  //joint1 servo`
`Servo joint2;  //joint2 servo`
`Servo joint3;  //joint3 servo`
`Servo joint4;  //joint4 servo`
`Servo clawservo;  //claw servo`
	Creates servo objects for each of the robot manipulator's joints and the end effector claw. Each joint of the robot manipulator is a servo motor, and the claw end effector is opened and closed using a servo motor.


`int16_t theta_1 = 90, theta_3 = map(-65, 0, -90, 180, 90);`
`int16_t theta_2 = 90, theta_4 = map(25, 0, 90, 90, 180); /`
	Variables for each joint's theta value. Since the theta values are variable, the variables are not constant. Theta 3 and 4 use the map() function to adjust the real theta values to the angle of the servo for the Arduino (ex. If the real world angle is -65° and the joint can range between 0° and -90° while the Arduino has this angle between 180° and 90°, map() changes -65° to 115°).


`const uint16_t claw_open = 160, claw_closed = 30;`
`const uint16_t joint1_2_initial_position = 90; //`
	Variables for the open and closed servo degrees. Variables for the initial (home) position (in degrees) for the servo motors of joints 1 and 2. All these variables are constant because they do not change.

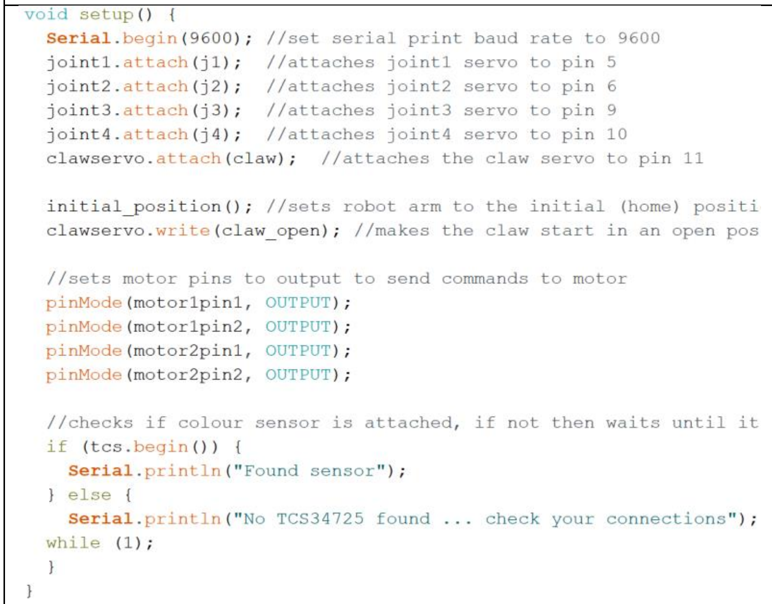

`uint16_t red_counter = 0, blue_counter = 0;`
`uint16_t yellow_counter = 0;`
	Variables for counting how many times the colour sensor has detected a colour in succession. These colour counter variables are used to determine a more accurate reading of an object's colour since the colour sensor can detect a wrong colour under

<table>
<tr>
<td></td>
<td>conditions like when positioning the object in front of it.</td>
</tr>
<tr>
<td>



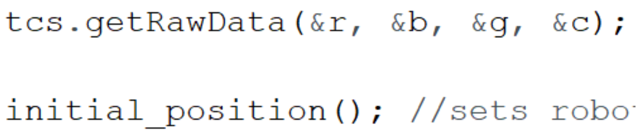
```
const uint16_t motor1pin1 = 2, motor1pin2 = 7;
const uint16_t motor2pin1 = 3, motor2pin2 = 4;
```

</td>
<td>Pins on the Arduino that are connected to the DC motors of the conveyor belt. Each DC motor uses two pins on the Arduino, hence there are two variables for each DC motor. The pins on the Arduino that are connected to the conveyor belt DC motors do not changed so the variables are constant.</td>
</tr>
<tr>
<td>



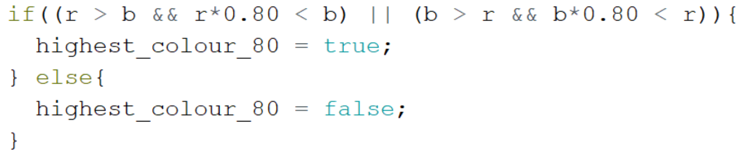
```
void setup() {
  Serial.begin(9600); //set serial print baud rate to 9600
  joint1.attach(j1);  //attaches joint1 servo to pin 5
  joint2.attach(j2);  //attaches joint2 servo to pin 6
  joint3.attach(j3);  //attaches joint3 servo to pin 9
  joint4.attach(j4);  //attaches joint4 servo to pin 10
  clawservo.attach(claw);  //attaches the claw servo to pin 11

  initial_position(); //sets robot arm to the initial (home) positi
  clawservo.write(claw_open); //makes the claw start in an open pos

  //sets motor pins to output to send commands to motor
  pinMode(motor1pin1, OUTPUT);
  pinMode(motor1pin2, OUTPUT);
  pinMode(motor2pin1, OUTPUT);
  pinMode(motor2pin2, OUTPUT);

  //checks if colour sensor is attached, if not then waits until it
  if (tcs.begin()) {
    Serial.println("Found sensor");
  } else {
    Serial.println("No TCS34725 found ... check your connections");
  while (1);
  }
}
```

</td>
<td>Setup of code. Allows serial output, attaches the pins of the Arduino to the servos, sets the robot arm to the initial (home) position, opens the claw end effector, attaches the pins of the Arduino to the conveyor belt DC motors, and checks if the colour sensor is attached.</td>
</tr>
<tr>
<td>



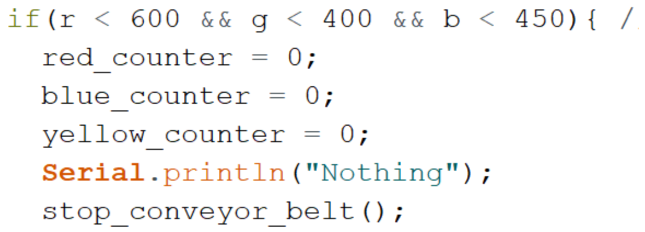
```
tcs.getRawData(&r, &b, &g, &c);

initial_position(); //sets robo
```

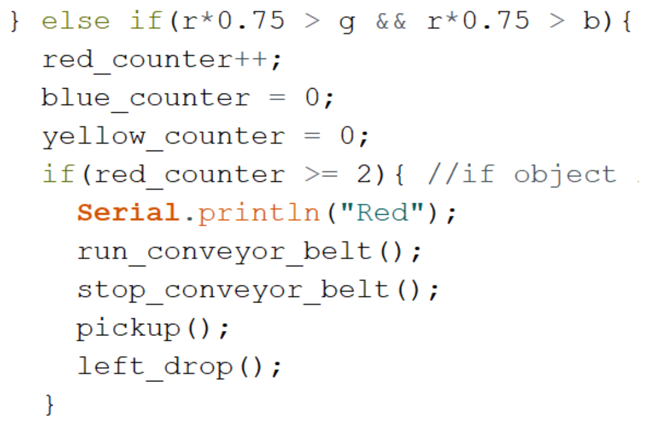
</td>
<td>The initial functions of the looped code are for the colour sensor to get the red, blue, green, and clear colour readings and then to set the robot manipulator to its initial position. The colour reading is put first because the rest of the looped code using conditional statements that rely on the colour reading. The robot manipulator is brought to its initial position to ensure it is in a known position at the beginning of every loop. This prevents unforeseen paths of the robot manipulator from occurring. Code snippet used for the project is from how to use an Adafruit TCS34725 colour sensor and an Arduino [3].</td>
</tr>
<tr>
<td>

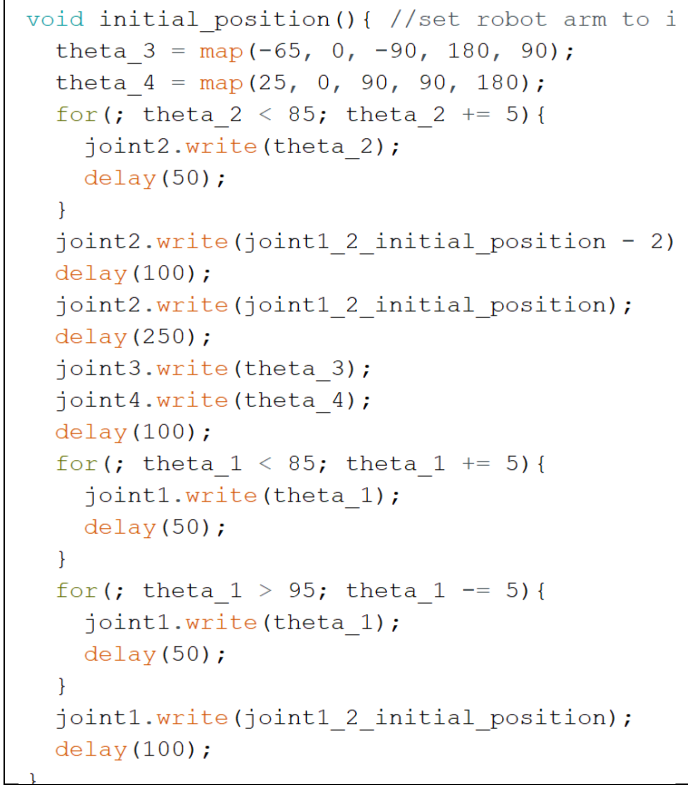


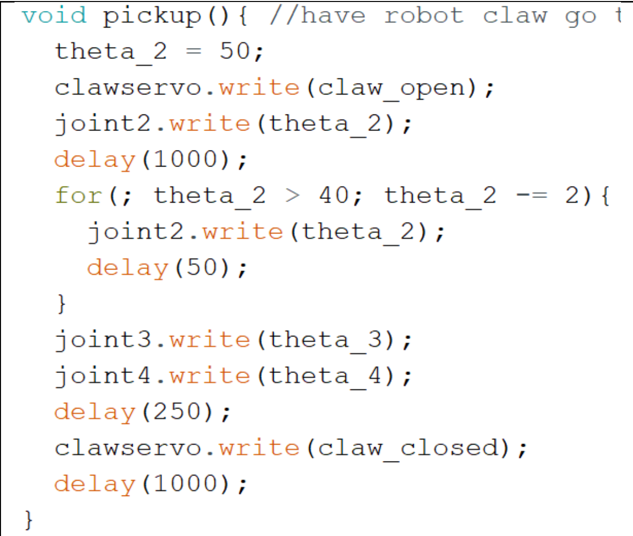
```
if((r > b && r*0.80 < b) || (b > r && b*0.80 < r)){
  highest_colour_80 = true;
} else{
  highest_colour_80 = false;
}
```

</td>
<td>If-else statement to check if 80% of the highest colour value is greater than the full value of the second highest colour value. This is used to determine what colour the object is. Only red and blue are compared because these are always the two highest colour values with the sponge objects used.</td>
</tr>
</table>


if(r < 600 && g < 400 && b < 450){ /
  red_counter = 0;
  blue_counter = 0;
  yellow_counter = 0;
  Serial.println("Nothing");
  stop_conveyor_belt();
	If the red, green, and blue values are less than 600, 400, and 450, respectively, than an object is not in front of the colour sensor. These values were found through testing using the Colour_Sensor_and_Conveyor_Belt_Setup Arduino file by putting objects in front of the colour sensor and seeing the range of values of the red, blue, and green colours. If no object is detected than each colour's counter is reset, the serial output is "Nothing", and the conveyor belt is stopped.

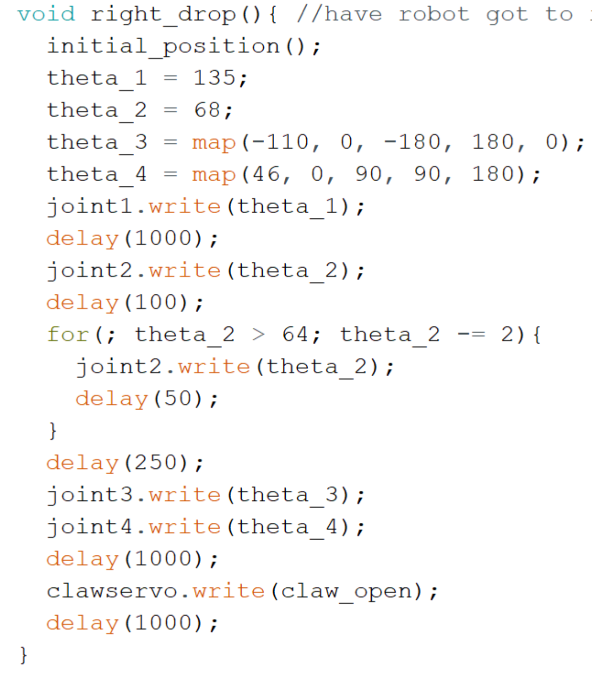

} else if(r*0.75 > g && r*0.75 > b){
  red_counter++;
  blue_counter = 0;
  yellow_counter = 0;
  if(red_counter >= 2){ //if object
    Serial.println("Red");
    run_conveyor_belt();
    stop_conveyor_belt();
    pickup();
    left_drop();
  }
	A red object has been detected if 75% of the red value is greater than the green and blue values. The red counter is then incremented while the blue and green counters are reset. If a red object has been detected twice (or more) in a row then "Red" is outputted serially, the object is moved from in front of the colour sensor to the pickup area, the robot manipulator picks up the object, and the robot manipulator drops the object in the left (red) bin. An identical pattern is followed when a blue or yellow object is detected.


void initial_position(){ //set robot arm to i
  theta_3 = map(-65, 0, -90, 180, 90);
  theta_4 = map(25, 0, 90, 90, 180);
  for(; theta_2 < 85; theta_2 += 5){
    joint2.write(theta_2);
    delay(50);
  }
  joint2.write(joint1_2_initial_position - 2)
  delay(100);
  joint2.write(joint1_2_initial_position);
  delay(250);
  joint3.write(theta_3);
  joint4.write(theta_4);
  delay(100);
  for(; theta_1 < 85; theta_1 += 5){
    joint1.write(theta_1);
    delay(50);
  }
  for(; theta_1 > 95; theta_1 -= 5){
    joint1.write(theta_1);
    delay(50);
  }
  joint1.write(joint1_2_initial_position);
  delay(100);
}
	Function used to put the robot manipulator into the initial (home) position. The joints have theta angles 0°, 90°, -65°, and 25°, respectively. The end effector x, y, z, and rotation (phi) are 20.1961, 0, 36.6791, and 50°, respectively. This displacement and rotation are used as the initial position because it a distinct location that from visual inspection will not be confused with another position and in this position the robot manipulator is far enough away from the conveyor belt and bins, so it will not hit them when moving to this position. The for loops are used to stop the movement of their respective joint earlier than the desired final position. This ensures the robot manipulator avoids hitting another object and has a smooth approach to the desired final position. Without the for loops the robot manipulator creates a large force when moving and hits the conveyor belt during pickup.



```
void pickup(){ //have robot claw go t
  theta_2 = 50;
  clawservo.write(claw_open);
  joint2.write(theta_2);
  delay(1000);
  for(; theta_2 > 40; theta_2 -= 2){
    joint2.write(theta_2);
    delay(50);
  }
  joint3.write(theta_3);
  joint4.write(theta_4);
  delay(250);
  clawservo.write(claw_closed);
  delay(1000);
}
```

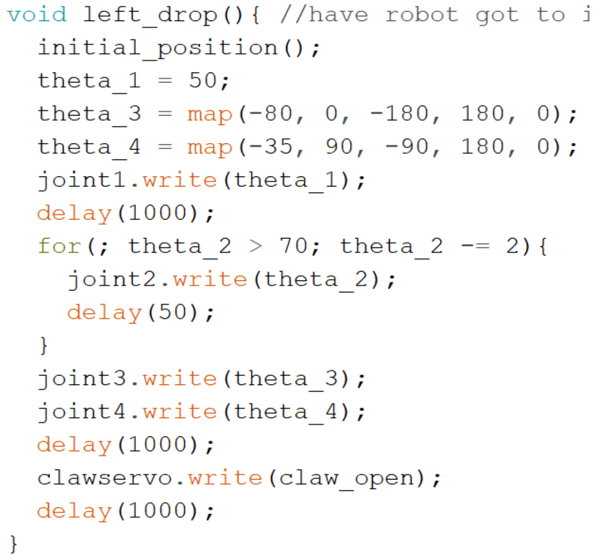
The pickup function sends commands to the joints to have the claw end effector go to the location of the pickup location of the objects on the conveyor belt. Each joint's real world theta angle then changes to $\theta_1 = 0°$, $\theta_2 = 50°$, $\theta_3 = -65°$, and $\theta_4 = 25°$. The end effector's position is x = 37.2494, y = 0, z = 9.8918 while its orientation is phi = 0°. As explained in the initial_positon() function, the for loop stops joint 2 early and then slowly moves joint 2 to the desired angle to ensure the robot manipulator will not hit the conveyor belt when it is picking up an object. The claw then closes to grasp the object. The delays allow an adequate amount of time to stop the servo motors that move the joints and then pick up the object in two distinct actions (rather than the two actions overlapping).



```
void right_drop(){ //have robot got to
  initial_position();
  theta_1 = 135;
  theta_2 = 68;
  theta_3 = map(-110, 0, -180, 180, 0);
  theta_4 = map(46, 0, 90, 90, 180);
  joint1.write(theta_1);
  delay(1000);
  joint2.write(theta_2);
  delay(100);
  for(; theta_2 > 64; theta_2 -= 2){
    joint2.write(theta_2);
    delay(50);
  }
  delay(250);
  joint3.write(theta_3);
  joint4.write(theta_4);
  delay(1000);
  clawservo.write(claw_open);
  delay(1000);
}
```

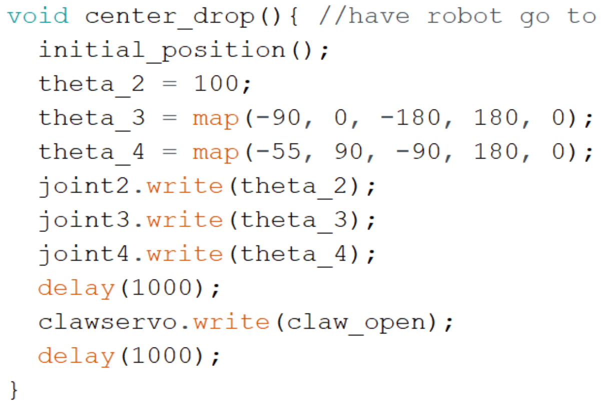
Function that drops the object in the right (blue) bin. Robot manipulator goes to the initial position at start of function. Each joint's real world theta angle then changes to $\theta_1 = -45°$, $\theta_2 = 68°$, $\theta_3 = -110°$, and $\theta_4 = 46°$. With these joint angles, the x, y, z, and phi angles are x = 20.9520, y = 20.9520, z = 10.2992, and phi = 4°. Phi is 4° so that z can be 10cm. The claw then opens to drop the object into the bin. The delays are one second each as they give the robot manipulator enough time to move the joints and then drop the object in a distinct manner from each other.

<table>
<tr>
<td>



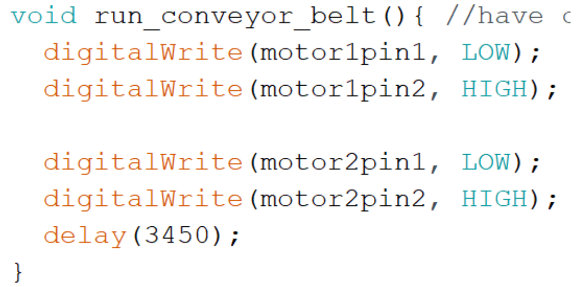
```
void left_drop(){ //have robot got to i
  initial_position();
  theta_1 = 50;
  theta_3 = map(-80, 0, -180, 180, 0);
  theta_4 = map(-35, 90, -90, 180, 0);
  joint1.write(theta_1);
  delay(1000);
  for(; theta_2 > 70; theta_2 -= 2){
    joint2.write(theta_2);
    delay(50);
  }
  joint3.write(theta_3);
  joint4.write(theta_4);
  delay(1000);
  clawservo.write(claw_open);
  delay(1000);
}
```

</td>
<td>Function that drops the object in the left (red) bin. Robot manipulator goes to the initial position at start of function. Each joint's real world theta angle then changes to θ_1 = 40°, θ_2 = 70°, θ_3 = -80°, and θ_4 = -35°. With these joint angles, the x, y, z, and phi angles are x = 20.9682, y = 17.5944, z = 7.2284, and phi = -45°. The claw then opens to drop the object into the bin. The delays are one second each as they give the robot manipulator enough time to move the joints and then drop the object in a distinct manner from each other. The delay in the for loop allows enough time to pause each joint movement so it does not create a large force.</td>
</tr>
<tr>
<td>



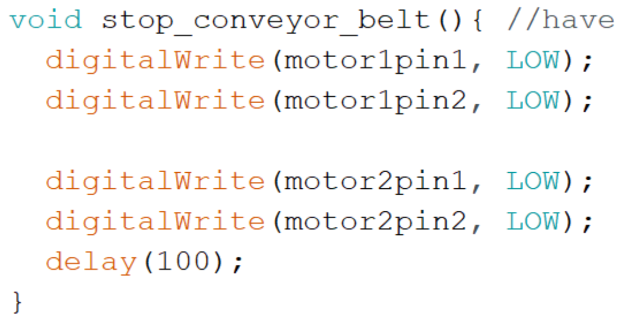
```
void center_drop(){ //have robot go to
  initial_position();
  theta_2 = 100;
  theta_3 = map(-90, 0, -180, 180, 0);
  theta_4 = map(-55, 90, -90, 180, 0);
  joint2.write(theta_2);
  joint3.write(theta_3);
  joint4.write(theta_4);
  delay(1000);
  clawservo.write(claw_open);
  delay(1000);
}
```

</td>
<td>Function that drops the object in the center (yellow) bin. Robot manipulator goes to the initial position at start of function. Each joint's real world theta angle then changes to θ_1 = 0°, θ_2 = 100°, θ_3 = -90°, and θ_4 = -55°. With these joint angles, the x, y, z, and phi angles are x = 19.3792, y = 0, z = 12.0952, and phi = 45°. The claw then opens to drop the object into the bin. The delays are one second each as they give the robot manipulator enough time to move the joints and then drop the object in a distinct manner from each other.</td>
</tr>
<tr>
<td>



```
void run_conveyor_belt(){ //have c
  digitalWrite(motor1pin1, LOW);
  digitalWrite(motor1pin2, HIGH);

  digitalWrite(motor2pin1, LOW);
  digitalWrite(motor2pin2, HIGH);
  delay(3450);
}
```

</td>
<td>Function that runs the conveyor belt's DC motors. The delay value of 3.45 seconds allows the DC motors to stay on long enough for the object to travel from in front of the colour sensor to the pickup location of the robot manipulator. Code snippet used for the project is from how to power up a DC motor with the L298N Motor Driver and an Arduino [2].</td>
</tr>
<tr>
<td>



```
void stop_conveyor_belt(){ //have
  digitalWrite(motor1pin1, LOW);
  digitalWrite(motor1pin2, LOW);

  digitalWrite(motor2pin1, LOW);
  digitalWrite(motor2pin2, LOW);
  delay(100);
}
```

</td>
<td>Function that stops the conveyor belt's DC motors. Delay for 100 milliseconds keeps each loop of the code as short as possible while still turning off the DC motors. Code snippet used for the project is from how to power up a DC motor with the L298N Motor Driver and an Arduino [2].</td>
</tr>
</table>

References

[1] "map()," *map() - Arduino Reference*. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/math/map/. [Accessed: 05-May-2021].

[2] R. Chan, "How to use the L298N Motor Driver," *Arduino Project Hub*. [Online]. Available: https://create.arduino.cc/projecthub/ryanchan/how-to-use-the-l298n-motor-driver-b124c5. [Accessed: 05-May-2021].

[3] shedboy71, "Arduino and TCS34725 Color Sensor," *Arduino Learning*, 26-Nov-2017. [Online]. Available: http://arduinolearning.com/code/arduino-tcs34725-color-sensor.php. [Accessed: 05-May-2021].

[4] T. Lens, J. Kunz, and O. von Stryk, "Dynamic Modeling of the 4 DoF BioRob Series Elastic Robot Arm for Simulation and Control," *Simulation, Modeling, and Programming for Autonomous Robots*, pp. 411–422, 2010.