EML 6805 Project 1 Report

Terrance Williams

September 30, 2022

Abstract

A report detailing the specifications, assembly, and testing of the Lynxmotion 3 DoF Robot Arm.

Contents

1 Introduction

EML 6805 is a project-based, graduate level engineering course concerning the fundamentals of robotics. For the first project assignment, the Lynxmotion 3 Degrees of Freedom (DoF) Robot Arm was assembled. Upon completion, tests on the robot's range of motion were conducted in order to both characterize the driving actuators and ensure the expected functionality.

2 Platform Specifications

The Lynxmotion robot platform utilized in this project is a 3 DoF robot consisting of three revolute actuating joints. The actuators are Lynxmotion Smart Servo motors (LSS) as well as one mini RC servo to control the arm's gripper. The robot links connected to servo 2 form a four-bar linkage system, giving the the robot a bit more complex geometry than a serial bot. The default coordinate system used to represent the position of the end-effector is the Cartesian system, though cylindrical may be utilized instead [1].

Figure 1: Lynxmotion 3DoF Arm Model[1]

[1]Image Source: https://wiki.lynxmotion.com/info/wiki/lynxmotion/download/servo-erector-set-robots-kits/ses-v2-robots/ses-v2-arms/lss-3-dof-arm/WebHome/LSS-3DOF-ISO.PNG

The robot has a (theoretical) max horizontal reach of 15" and a max vertical reach of 8.25". The mini gripper is able to open to 34mm and its driving servo has a torque rating of 1.5kg cm, and each LSS motor has a torque rating of 14kg cm. The theoretical working envelope is shown below, but for purposes of this project and the following projects, the envelope's lower-bound will be constrained vertically to be parallel with the base. [2]

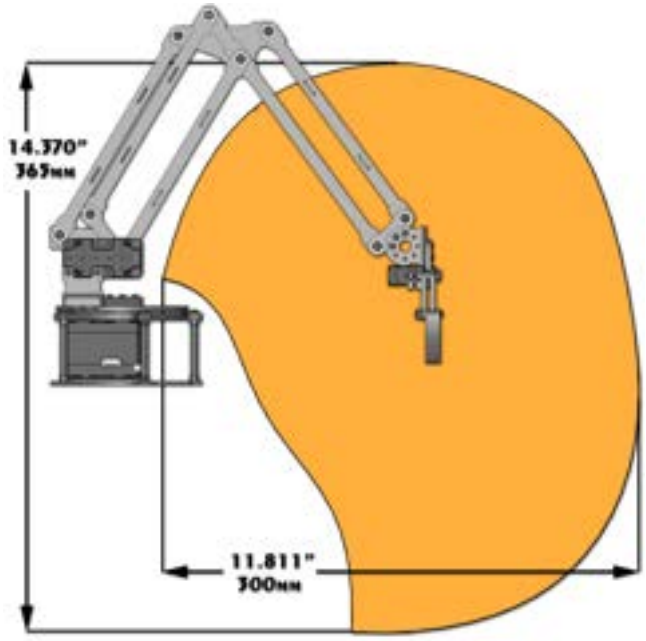


Figure 2: Theoretical Working Envelope of the Lynxmotion 3DoF Robot Arm[2]

3 Assembly

To assemble the Lynxmotion platform, the Quick Assembly Guide on the Lynxmotion Wiki was utilized [3]. The guide segments the assembly process into six steps: servo setup, the base, link preparation and assembly, the mini gripper, final assembly, and wiring. Following this visual guide rendered what would have been a complex assembly relatively simple.

[2]Image Source: https://wiki.lynxmotion.com/info/wiki/lynxmotion/download/servo-erector-set-robots-kits/ses-v2-robots/ses-v2-arms/lss-3-dof-arm/WebHome/LSS-3DOF-DIMENSIONS-Envelope.PNG

3.1 Servo Setup

For the servo setup, I first updated the firmware of each of the servos using LSS Config software and used this same software to assign each servo motor its own ID (1 through 3). The purpose of assigning each servo its unique ID is to facilitate communication with specific servos when desired by the user. After this, I removed the idler horns on each of the motors (as well as the driving horn from motor 1) and added mounting brackets on servos 2 and 3.

Figure 3: The three servos after the setup phase.

3.2 Base

The second step of this assembly was assembling the base of the robot arm. I attached the base platform to the spindle of motor 1, mounted the 5V regulator unit and the 2IO board, and connected the rotating base as well as the supports to the assembly.

Figure 4: Base of the Robot

3.3 Link Assembly

The link assembly was perhaps the most involved step of the assembly as one had to be mindful of the precise orientation of the links in order to produce the proper motion between rotary joints. This step also introduced the first sticking point in the assembly. The provided lock nuts used to connect and secure the links did not thread far enough along the screws used in this step. This meant that the screws had freedom to move within the assembly, allowing two planar links to become misaligned, thereby introducing complexity within the degrees of freedom that could have negatively affected the kinematics calculations for movement.

To remedy this, additional washers were introduced along length of the exposed screw. By doing this, the lock nuts were able to apply force on the washers which was then transferred to the links, thereby properly securing the connected parts. The desired result was achieved: the affected links were constrained to move in the same plane.

(a) Link Assembly: Side View

(b) Link Assembly: Back View

3.4 Mini Gripper

The next step was the mini gripper. The gripper was assembled using the laser cut acrylic pieces and attached the assembly to the provided mini servo.

(a) Assembled Mini Gripper: Front View

(b) Assembled Mini Gripper: Side View

3.5 Final Assembly

Finally, the previously assembled parts were combined in order to complete the total assembly. The link assembly was attached to the base, and the mini gripper was attached to the end of the link assembly. Orientation-wise, the gripper was positioned pointing downward, though the piece can be oriented horizontally if desired. Future testing on the vertical configuration will be conducted in order to determine whether the alternate positioning should be used. Finally, the LSS adapter board was attached to the supports of the base, completing the build.

(a) Full Assembly: Extended

(b) Full Assembly: Contracted

3.6 Wiring

Upon completion of the assembly, the servos were wired to the 5V regulator, the 2IO board, and the adapter board. The mini servo wire and subsequent extensions were connected to the 2IO board, allowing for control via the LSS FlowArm software. Finally, the power supply and micro-USB were both connected to the LSS Adapter Board, which then completed the electrical wiring and therefore completed the entire Lynxmotion build.

(a) Full Wiring: Side View

(b) Wiring: Front View

4 Platform Analysis

4.1 Initial Operation

It was deemed prudent to conduct an initial operations test for the robot to ensure the build was completed properly. This test was conducted via LSS FlowArm app, where the robot was taken through an assortment of motion tests. The rotation about the base of the robot as well as the gripper operated as expected with little to no issues. However, the elbow and shoulder rotation did not perform as expected. Even after the recommended calibration [1] of the robot through the LSS FlowArm app, the robot's physical movement did not match that as displayed in the GUI. A movement that was within the robot's expected range of motion in the LSS app could have resulted in a request outside of the capabilities of the smart servos, resulting in a Current Overflow warning on Servo 2 and, occasionally, Servo 3.

4.2 Remedy & Characterization

To fix the problem presented by the FlowArm app, the robot platform was controlled using an Arduino sketch. Fortunately, Lynxmotion provides an accessible Arduino development library for controlling their LSS servos, the primary servos controlling the robot. The robot arm assembly kit came with a 2IO Board which, in addition to the ability to serve as 2RC control via the FlowArm app, can also serve as an Arduino [4]. One can load an Arduino sketch onto the board in order to control the robot in a user-defined manner. After initial experimentation, the robot was able to be controlled more concretely through the Arduino sketch.

After successfully interfacing with the three Smart Servos, a characterization test was performed in order to determine the rotation range for the individual servos. This rotation range determines the workspace of the end-effector. To perform the characterization, each servo was looped through rotation steps via Arduino, moving through $+5^\circ$ from zero until the servo responded with the Current Overflow signal: the servo would blink its Red LED. This step-through was repeated for the -5° direction. After repeating this test for each LSS Servo, the following comfortable operating range (the range in which the robot can feasibly operate without failing) was determined:

Servo ID	Angle Range (in degrees)
1	[-85, 125]
2	[-75, 20)
3	[-45, 60]

Table 1: Table of the operating angle ranges for the Lynxmotion Smart Servos (LSS).

Also tested was the difference in movement options when controlling the LSS. It was determined that, depending on how far the arm needs to move, timed-move commands are preferable to immediate moves. The Lynxmotion Arduino library provides *move*(angle_position) and *moveT*(angle_position, time) commands; the former results in immediate movement at a pre-determined speed (i.e. quickly) while the latter results in the desired movement in the given amount of time. So, for large movements of, for example, >20°, it is better to use the time move because, as seen through experimentation, it results in smoother movements, forestalling system jerk. On the other hand, immediate movements are better for shorter movement distances because the servo does not have to restrain its movement to reach a defined time threshold.

Finally, in order to completely replace the LSS FlowArm app, the mini-gripper motion control was implemented using Arduino's "Servo" library. With the implementation of this package, the user is able to control the mini-gripper servo that is attached to the D9 pin on the robot and interact with other objects.

5 Flowchart(s)

Both the servo characterization and movement tests can be modeled using flowcharts.

5.1 Servo Characterization

As stated previously, the servo characterization test consisted of stepping through the servos' workable range by ±5° until the current warning activated. A simple flow diagram is seen below:

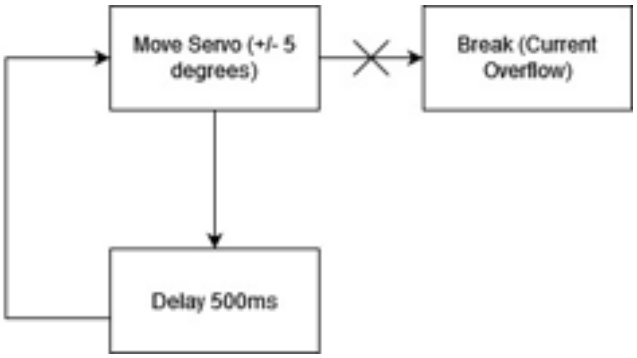


Figure 9: Servo Characterization Flowchart

5.2 Movement Test

The movement test was slightly more complex than the servo characterization. It entailed rotating the arm to the right relative to the front of the robot, contracting back, and moving the end-effector downward. Next, the bot moves to the HOME position, flipping the gripper state (open or close) and querying the servos for relevant information such as current and temperature readings. The bot then moves to the left, extending the arm and moving the end effector upward. After this, the arm moves HOME again, performing the same operations of gripper toggling and servo querying. This sequence is allowed to run ad infinitum, allowing the user to observe potential servo strain or movement hiccups until the device is powered down.

The flowchart for the movement test is displayed below:

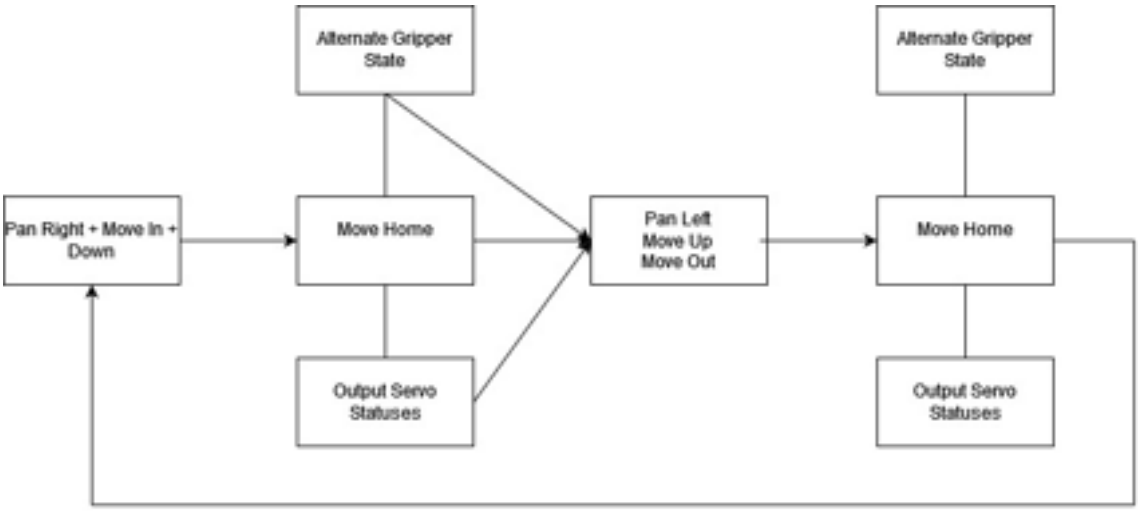


Figure 10: Movement Test Flowchart

6 Conclusion & Future Work

Project 1 consisted of the assembly and testing of the Lynxmotion 3 DoF Robot Arm platform system. The assembly was successful, and the LSS servos were characterized in order to determine the working movement range for the robot. After some trouble with the Lynxmotion FlowArm control application, the robot was instead controlled with an Arduino sketch.

For future projects the following will need to be implemented:

- Implement the Forward Kinematics and Inverse Kinematics code for the robot. For Project 1, I was satisfied with inducing movement in the robot. Moving forward, I must implement the ability to determine the coordinates of the end-effector and programmatically calculate the necessary servo angles in order to move to the desired location.
- Integrate camera sensor and, potentially, a position detection method to the robot circuit. Find or write a program that detects basic solid colors (red, blue, green, yellow, etc.) via RGB as well as can calculate the universal coordinates of a desired object.
- Implement an algorithm for sorting colored objects. Synthesize the other components of projects to complete the target objective.

References

[1] E. Nantel, *Lss flowarm*, 2022. [Online]. Available: https://wiki.lynxmotion.com/info/wiki/lynxmotion/view/ses-software/lss-flowarm/.

[2] E. Nantel, *3 dof robotic arm*, 2022. [Online]. Available: https://wiki.lynxmotion.com/info/wiki/lynxmotion/view/servo-erector-set-robots-kits/ses-v2-robots/ses-v2-arms/lss-3-dof-arm/.

[3] E. Nantel, *3 dof arm quickstart*, 2022. [Online]. Available: https://wiki.lynxmotion.com/info/wiki/lynxmotion/view/servo-erector-set-robots-kits/ses-v2-robots/ses-v2-arms/lss-3-dof-arm/3dof-arm-quickstart/.

[4] E. Nantel, *Lss-2io board*, 2021. [Online]. Available: https://wiki.lynxmotion.com/info/wiki/lynxmotion/view/servo-erector-set-system/ses-electronics/ses-modules/lss-2io-board/.

```
/*
    Description:  Basic Control Function Testing and Movement Test
    References: LSS control examples taken from the following:
       - LSS Sweep: LSS_Sweep.ino
https://github.com/Lynxmotion/LSS_Library_Arduino/blob/master/exampl
es/LSS_Sweep/LSS_Sweep.ino
       - LSS Query: LSS_Query.ino
https://github.com/Lynxmotion/LSS_Library_Arduino/blob/master/exampl
es/LSS_Query/LSS_Query.ino
       - Servo Operation: Sweep.ino
https://github.com/arduino-libraries/Servo/blob/master/examples/Swee
p/Sweep.ino
*/

#include <LSS.h>
#include <Servo.h>
/* SERVO Angle Position ranges (in 1/10 a degree)
 *
 * Servo 1: (-850, 1250] rotates end effector about universal Z axis
 * Servo 2: [-750, 200) (moves end effector vertically)
 * Servo 3: [-450, 600] (moves end effector; reach and contract)
 */

// ID set to default LSS ID = 0
#define LSS_ID1    (1)
#define LSS_ID2    (2)
#define LSS_ID3    (3)
#define LSS_BAUD  (LSS_DefaultBaud)
// Choose the proper serial port for your platform
#define LSS_SERIAL  (Serial)
// Define Pulse Widths for mini-gripper servo
#define MIN_PULSE_WIDTH (900)
#define MAX_PULSE_WIDTH (2100)


// Create one LSS object
LSS myLSS1 = LSS(LSS_ID1);
```

```
LSS myLSS2 = LSS(LSS_ID2); // -angle values move end effector down
(CCW servo rotation); wider range of motion (-750 to 200)
LSS myLSS3 = LSS(LSS_ID3); // -angle values move end effector back
(contraction)

// Instantiate gripper servo
Servo myservo;
int servo_angle = 180; // Servo Position Default closed

void setup()
{
  // Initialize the LSS bus
  LSS::initBus(LSS_SERIAL, LSS_BAUD);
  myservo.attach(9, MIN_PULSE_WIDTH, MAX_PULSE_WIDTH); // attaches
the servo on pin 9 to the servo object
  myservo.write(servo_angle);
  // start up time
  delay(3000);

  // Initialize LSS to position 0.0 °
  myLSS1.move(0);
  myLSS2.move(0);
  myLSS3.move(0);

  // Wait for it to get there
  delay(2000);
}

// Loops between -180.0° and 180°, taking 1 second pause between
each half-circle move.
void loop()
{
  // movement test
  pan_right();
  mv_home();
  pan_left();
  mv_home();
```

```
}


// Useful functions

void pan_right(){
  myLSS1.moveT(500, 500);
  delay(1000);
  myLSS3.moveT(-250, 1000);
  delay(1500);
  myLSS2.moveT(-400, 1000);
  query_all();
  delay(2500);
}

void pan_left(){
  myLSS1.moveT(-500, 500);
  delay(1000);
  myLSS3.moveT(250, 1000);
  delay(1500);
  myLSS2.moveT(-400, 1000);
  query_all();
  delay(2500);

}

void mv_home() {
  if (abs(myLSS1.getPosition()) > 300){
    myLSS1.moveT(0, 900);
  }
  if (abs(myLSS2.getPosition()) > 200){
    myLSS2.moveT(0, 900);
  }
  if (abs(myLSS3.getPosition()) > 200){
    myLSS3.moveT(0, 900);
```

```
  }
  else{
    myLSS1.move(0);
    myLSS2.move(0);
    myLSS3.move(0);
  }
  query_all();
  servoAlternate();
  delay(2000);
}
void query(LSS servo){
  Serial.println("\r\nQuerying servo...");

  // Get LSS ID, position, speed, current, voltage, temperature
  uint8_t id = servo.getServoID();
  int32_t pos = servo.getPosition();
  uint8_t rpm = servo.getSpeedRPM();
  uint16_t current = servo.getCurrent();
  uint16_t voltage = servo.getVoltage();
  uint16_t temp = servo.getTemperature();

  // Header 2
  Serial.println("\r\n\r\n---- LSS telemetry ----");

  // Display LSS position, speed, current, voltage, temperature
  Serial.print("Servo ID: ");
  Serial.println(id);
  Serial.print("Position  (1/10 deg) = ");
  Serial.println(pos);
  Serial.print("Speed          (rpm) = ");
  Serial.println(rpm);
  Serial.print("Current          (mA) = ");
  Serial.println(current);
  Serial.print("Voltage          (mV) = ");
  Serial.println(voltage);
  Serial.print("Temperature (1/10 C) = ");
  Serial.println(temp);
```

```
}

void query_all(){
  query(myLSS1);
  query(myLSS2);
  query(myLSS3);
}

void servoAlternate(){
  // pos 0 = open
  // pos 180 = closed
  /* FUTURE To-Do: Figure out when to detach the servo
   * (via servo.detach()) in order to make the servo go limp.
  */
  if (myservo.read() == 0 and servo_angle == 0){
    servo_angle = 180;
    myservo.write(servo_angle);
    delay(150);
  }
  else{
    servo_angle = 0;
    myservo.write(servo_angle);
    delay(150);
  }
}
```

EML 6805 Project 2 Report

Terrance Williams

November 3, 2022

Contents

1 Introduction

EML 6805 is a graduate-level course in which students learn the fundamentals of robotics. For Project 1 of this course, I assembled and tested the Lynxmotion robot arm with three degrees of freedom (3DoF) for bare-bones operation such as basic movement of the three servos and interaction of the arm's mini-gripper. With its completion, more complex features can be designed and integrated into the robot's operation.

Figure 1: Assembled Lynxmotion Platform

2 Project 2 Goals

The course progressed to Project 2, and keeping the above in mind, features for the robot's use were specified. Due to the complexity of the initial purpose of the arm (a variable colored-block sorter), the scope of the project was reduced to be more manageable for both the time constraint and my experience level. The updated plan for the arm is for it to be used as a "pick and place" (P&P) bot that can receive input colors to determine the object's target position.

In order to achieve this goal, there were three main tasks for Project 2:

- Implement a color detection system

- Derive and program forward and inverse kinematics for the Lynxmotion arm
- Implement object detection in order to detect the position of the P&P object.

The remainder of this report will discuss how these tasks were completed and their implications for the third project.

3 Color Detection

3.1 Motivation

Ultimately, the robot system should be able to detect a user-presented input color and use this color to place the given block on the associated colored platform. For example, given the input color red, the robot must place the block on the red platform. This color detection method was implemented using Python.

3.2 Color Filtering Overview

Using Python's OpenCV library as well as Numpy, an HP laptop webcam was controlled to capture images in a video stream and apply a filter that isolates a user-determined color. Based on available OpenCV tutorials [1],[2], this operation was relatively simple to program. To filter colors in OpenCV, the Hue-Saturation-Value (HSV) color representation is used. OpenCV uses Blue-Green-Red (BGR) representation by default, but this format is not conducive for color filtering along a specific hue channel. Changes along a given BGR channel (for example, slight variations along the Green channel) can result in a completely different base hue than desired [1].

HSV solves this problem by allowing the user to represent an entire range of colors for a given pigment, for example the spectrum of blues, by defining the *Hue* value (the base color) and allowing variations along this hue using *Saturation* (the amount of "gray" in the color) and *Value* (the intensity of the color) [3]. By defining static upper and lower bounds for the *Saturation* and *Value* parameters for all hues, one can then focus on finding the optimal *Hue* limits for the desired color filtering.

3.3 Defining the HSV Bounds

For this project, the colors used are red, blue, green, and yellow. The goal is to be able to detect these colors within a given image, so it was necessary to find suitable HSV ranges to represent them. The *Hue* ranges were found using the *GNU Image Manipulation Program*, also known as GIMP [1]. GIMP represents HSV values in the following way:

Hue values range from 0 to 360, where the colors of the color spectrum are spread evenly across this range (Red: 0-59 & 360, Yellow: 60-119, Green: 120-179, Cyan:b180-239, Blue:b240-299, and Magenta: 30-359). This is the conventional representation for the *Hue* parameter [3]. The *Saturation* and *Value* parameters range from 0-100, representing the percentage of grayness and intensity, respectively. To find the

proper *Hue* bounds, the Saturation and Value parameters were set to max (100) while the *Hue* parameter was varied along its range of values to determine an adequate estimate for a given color. After finding this baseline, the *Hue* values were reduced by half to fit within OpenCV's *Hue* range, which spans from 0 to 179 [4].

Figure 2: This figure displays GIMP's color tool. Shown in the image is the HSV representation for pure green (120, 100, 100). In BGR, the color would be represented as (0, 255, 0).

Naturally, the quantitative nature of computers vs. the qualitative color classification process used by humans will produce differing results, so trial-and-error was necessary to determine a satisfying range of values for red, blue, green, and yellow respectively that allowed some convergence between the two entities. After much testing, the following ranges were determined for each pigment:

Color	OpenCV Hue Range
Red	[0, 7] ∪ [165, 179]
Yellow	[20, 35]
Green	[40, 90]
Blue	[95, 140]

In OpenCV, *Saturation* and *Value* both range from 0 to 255, so the following bounds were used on the recommendation of online resources [2] as well as trial-and-error:

- Saturation: 100-255
- Value: 20-255

3.4 Image Color Detection

After determining the HSV boundary parameters for each color, the captured frames could then be filtered for one of the four colors. For a given OpenCV image (also known as a frame) we convert it to its HSV value. Next, we create masks to isolate a given color using the defined lower and upper bounds for a given color's HSV representation. Finally, we perform a bitwise-AND operation on the original BGR frame using the mask. This results in a BGR image that is mostly black except for the pixels in the image that have the desired color. An example of the filtering applied on a Rubik's cube[1] is presented below.

[1]Image URL: https://richardnewmansays.blogspot.com/2010/11/blog-post_13.html

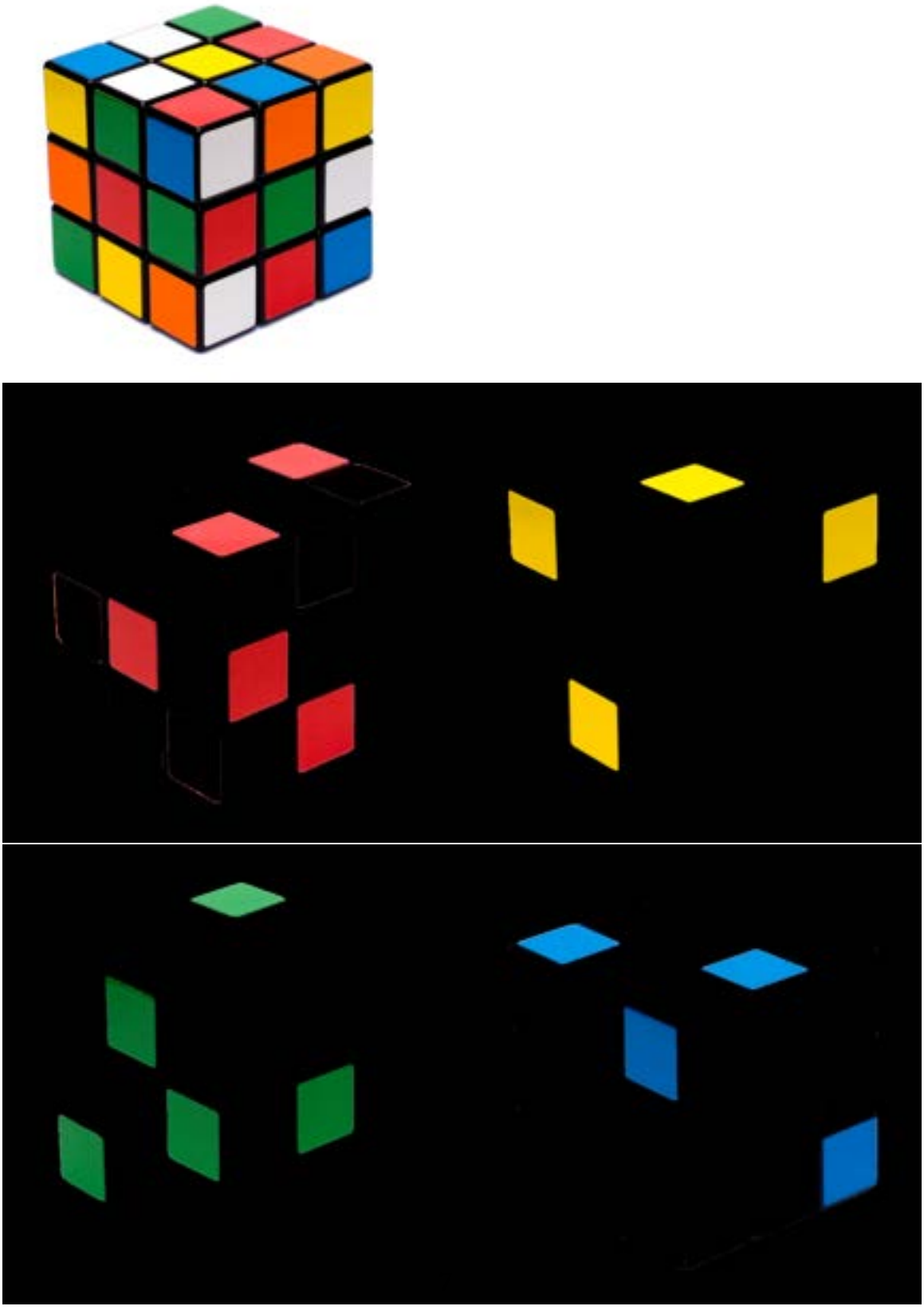


Figure 3: The output of the color detection program on a Rubik's Cube image. The colors filtered in order are red, yellow, green, and blue.

3.5 Further Implementation

With the basic implementation of color detection complete, the next objective was to apply this detection in real-time. To do so, OpenCV is used to capture frames from a laptop camera. A sub-region of the captured image is defined to be in the bottom-left section of the frame. This will be the region the user displays the desired input color. The aforementioned color detection algorithm is applied to this sub-region, with the program filtering for each color. The resulting intermediate output is an array of four masked images: red, yellow, green, and blue in that order. To determine if a color is present within the sub-region, the masked images are tested using the following: for a given sub-region, the center pixel and the surrounding one-pixel area are checked for the presence of color. Each pixel has a BGR value (B, G, R) and these values can be summed to determine if the pixel is within the color threshold.

Because the images are masked—flooring non-relevant pixels to black (B,G,R) = (0,0,0)—the sum for a given pixel will have a value of 0. Therefore, any non-zero value will correspond to the desired color. If all nine of the polled pixels have non-zero sums, one can be reasonably sure that the color is present in the sub-region. For greater assurance, the polled area can be expanded.

To prevent false-positives, the program must reach a threshold of consecutive color detections for a single color. Once this threshold is reached, the program determines the color is present and can theoretically communicate *which* color it is for further operation of the robot arm.

This result can be used for the following future implementation. If one of the user-defined colors are present, the program determines that this color corresponds to the color of the block's target platform. Then, combined with the knowledge of where the block is currently located, the robot can perform its pick-and-place operation, calculating the necessary joint angles to reach both the block's original position and its target. Of course, this assumes that the platforms are in an environment that is essentially isolated from colors that would introduce undesired noise to program's operation. For visual assistance, a flowchart model of the color detection algorithm is presented.

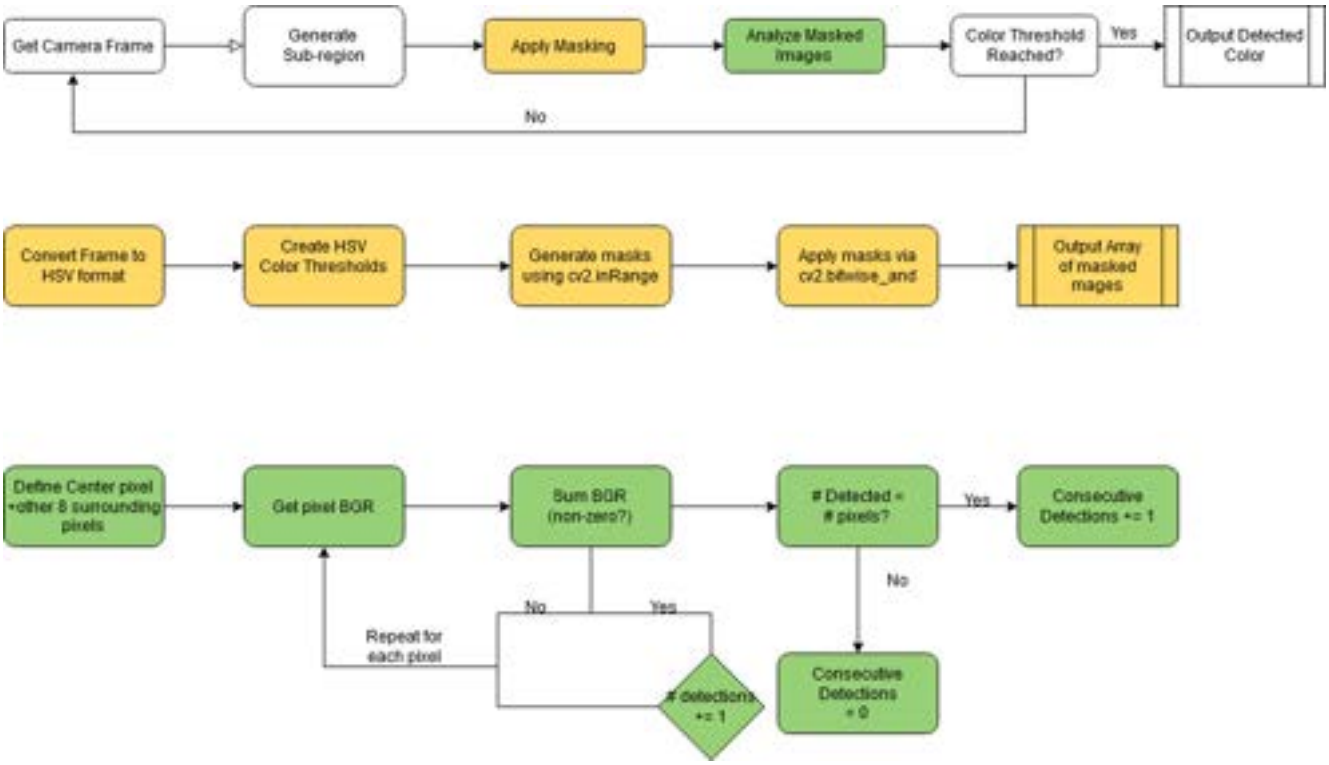


Figure 4: A flowchart of the overall color detection algorithm as well as sub-processes.

4 Lynxmotion Kinematics

4.1 Motivation

The robot should be able to calculate its inverse kinematics to determine the necessary angles each joint must have in order to reach a user-specified position within its range. It should also be able to calculate its forward kinematics, the robot's current position based on the status of its constituent servo motors. This feature will be useful in the case where the user or the robot itself manages to lose track of its current position.

4.2 Forward Kinematics

Though the 3 DoF arm has a mechanical makeup consisting of a four-bar linkage, the bot's behavior emulates a 2R planar bot on a rotating base and can therefore be modeled as such [5]. The robot's servos measure their respective angles relative to

the a-axis (servo 1) and the o-axis (servos 2 and 3), meaning at the RESET position, servo 1 rotates about the z-axis and servos 2 and 3 rotate about the y-axis. Knowing this, one can calculate the forward kinematics of the robot. Because the end-effector does not have its own rotational degree of freedom, we are not concerned with the effector orientation; it is always pointed downward (-z direction).

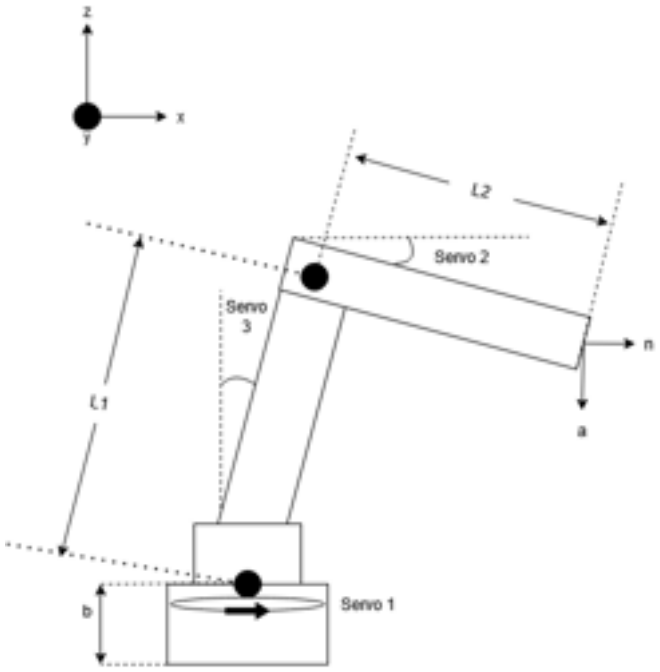


Figure 5: Simple schematic of the 2R with rotating base model representation of the Lynxmotion 3DoF robot arm.]

Using trigonometry and correcting for the behavior of the servos when using positive and negative angle entries, the forward kinematics for the robot end-effector is as follows:

$$ {}^{0}\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}_H = \begin{bmatrix} (l_2 \cos\theta_2 - l_1 \sin\theta_3)\cos\theta_1 \\ (-l_2 \cos\theta_2 + l_1 \sin\theta_3)\sin\theta_1 \\ l_2 \sin\theta_2 + l_1 \cos\theta_3 \end{bmatrix} \tag{1} $$

where $\theta_1, \theta_2, \theta_3$ are angles responding to servos 1, 2, and 3, respectively. The universal frame can be recovered by translating the end-effector along the z-axis by the height of the robot base (which we call 'b').

The validity of this model was verified using expected values for given angle configurations. For example, when all servo angles are at their zero values, one would

expect the robot to be in its reset position ${}^0\begin{bmatrix} p_x & p_y & p_z \end{bmatrix}_H^T = \begin{bmatrix} l_2 & 0 & l_1 \end{bmatrix}^T$ relative to the robot base frame. For the configuration where servo angles 1 and 3 are $-90°$, it is expected that the end-effector's position would be $\begin{bmatrix} 0 & l_1 + l_2 & 0 \end{bmatrix}^T$. This is the position where the robot arm would be completely horizontal while lying along the +y direction. In both cases, the forward kinematics equations result in the correct configuration and did so for other configurations not mentioned in this report.

4.3 Inverse Kinematics

The next step in the process was the derivation of the inverse kinematics. To derive the proper equations, the process presented in Dr. Rainer Hessmer's 2009 article, *Kinematics for Lynxmotion Robot Arm* [5] was followed. In this article, Dr. Hessmer uses a combination of the law of cosines, geometric and trigonmetric reasoning, and algebraic manipulation to calculate the angles for a given position in space. Notable about this article's application in this project is that the angles used in the articles are not necessarily correlative to those used by the Lynxmotion Servos. Geometric reasoning was used in order to relate the angles used in the article to the arm's servo angles.

The results of the calculation are the following for a given (p_x, p_y, p_z):

$$\theta_1 = -ATAN2(p_y, p_x)$$
$$\theta_2 = \theta_a + \theta_3 - \frac{\pi}{2}$$
$$\theta_3 = ATAN2(z', d) - ATAN2(k_2, k_1)$$
$$z' = p_z - b$$
$$d = \sqrt{p_x^2 + p_y^2}$$
$$k_1 = l_2 + l_1 \cos\theta_a$$
$$k_2 = l_1 \sin\theta_a$$
$$\cos\theta_a = \frac{d^2 + z'^2 - l_1^2 - l_2^2}{2 l_1 l_2}$$
$$\theta_a = ATAN2(\sqrt{1 - \cos^2\theta_a}, \cos\theta_a)$$

where θ_a is the angle link 2 makes with link 1. With these equations, the robot arm can now generate the necessary angles to move to specified locations within its angle range. A more detailed derivation of the forward and inverse kinematics are presented in the Appendix.

5 Object Detection

5.1 Motivation

The purpose of this module was to implement an object detection system to determine the position of the P&P object. The idea was that one could use the same camera used for color detection for this purpose.

5.2 Decision to Jettison

While a starting point for object detection was achieved in the form of detecting rectangles using OpenCV, it was soon discovered that object detection was an unnecessary feature to implement for several reasons:

1. Rather than running an object detection algorithm to detect a cube for every frame captured by the camera, one could instead use variables to keep track of the location of the cube. Setting the cube location upon startup of the program and updating the position over the course of operation is much less resource-intensive than the alternative.

2. Time: Color detection, Robot Kinematics, and Object Detection were all unexplored territories for the author. After successfully achieving both color detection and kinematics functionality, it was decided that it would be better to use the remaining project time testing and reiterating these features than working with unknown frameworks such as TensorFlow or other deep-learning frameworks.

3. Resource Efficiency: As stated above, the object detection algorithm would introduce more resource overhead to the project, extending the running time of the program for a single operation. Practically speaking, the introduction and handling of two or three additional variables in the program code is more economical than running an algorithm to achieve the same purpose and would likely be the preferred choice in an industry setting.

As a result, I have elected to abstain from incorporating this feature into the project.

6 Block Diagram(s)

The color detection and kinematics features introduce the ability to both collect an input and operate the robot arm in a specified way. To give a brief outline of the idealized operation of the robot, a block diagram as well as enumerated description is included below:

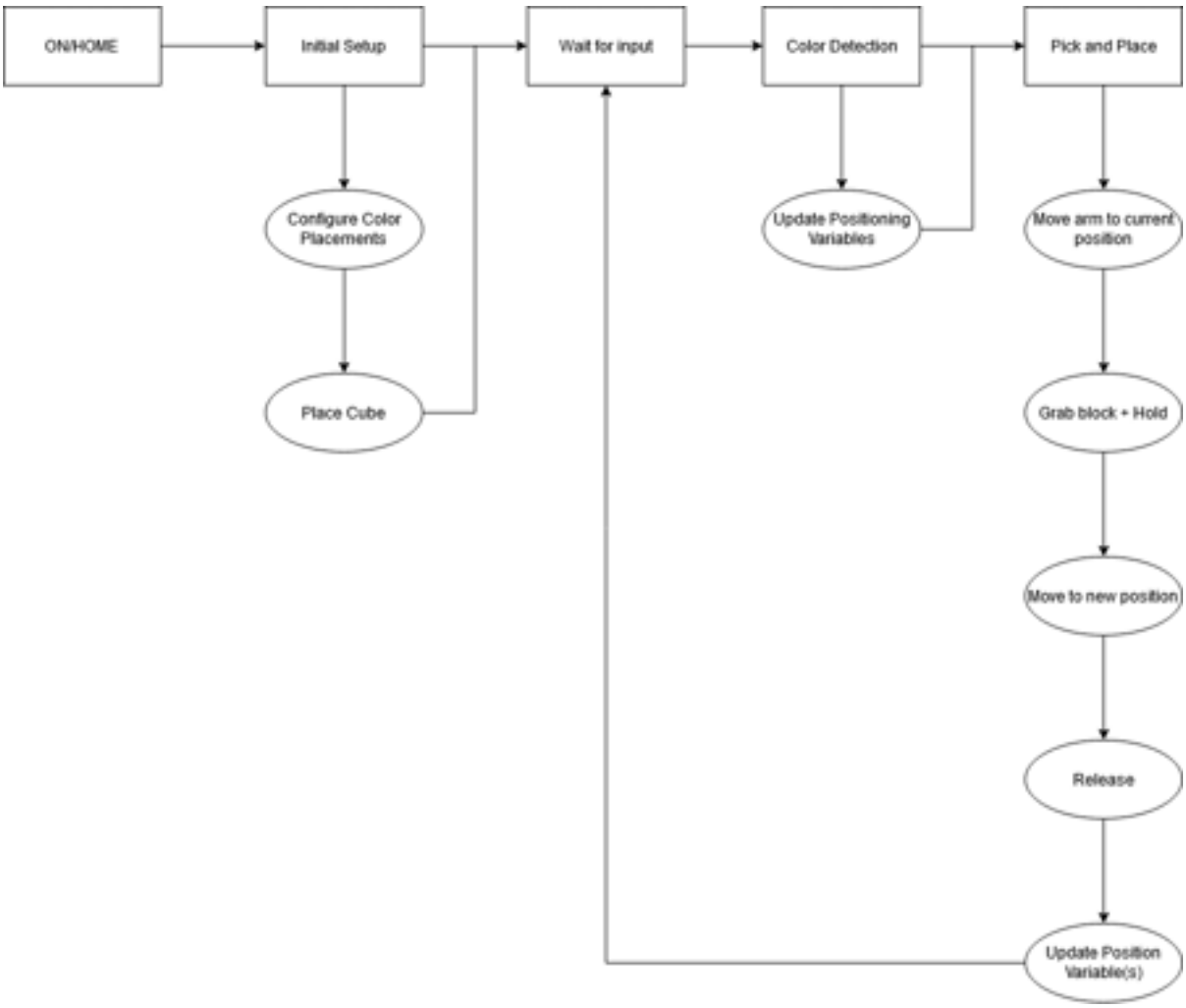


Figure 6: Flow overview of the robot P&P operation.

1. Power on Robot + move Home
2. Initial Setup
    - Configure target colored platforms
    - Place cube; set initial position
3. Wait for user color-input
4. Run color detection algorithm; update target variable
5. Perform P&P operation
    - Move arm to block's current position
    - Grab the block
    - Move to target position
    - Place block
    - Update block position variables.

Ultimately, this is how the robot should operate. With the core features programmed, the focus for project 3 will be the synthesis of these features and the practical setup of the operation environment.

7 Conclusion and Future Work

In this report, the color detection and kinematic calculations for the Lynxmotion 3DoF Robot Arm were discussed in-depth. Justification for the removal of the object detection feature from the plan was also provided. As the course moves to Project 3, the following must still be achieved:

- Proper coalescence of the aforementioned features (especially Python -> Arduino communication).
- Robot Operation Environment (colored platforms, cube, proper heights and positions, etc.)
- P&P testing (proper robot movement speeds, best object for reproducible results)

Project 2 was very much theory-based, so the third project will consist of using the theory and constructed framework for real-world use.

References

[1] Pysource. "Simple color recognition with opencv and python," YouTube. (2021), [Online]. Available: https://www.youtube.com/watch?v=t71sQ6WY7L4.

[2] CreepyD. "How to detect colors in opencv [python]," YouTube. (2021), [Online]. Available: https://www.youtube.com/watch?v=cMJwqxskyek.

[3] J. H. Bear, *What is the hsv (hue, saturation, value) color model?* 2020. [Online]. Available: https://www.lifewire.com/what-is-hsv-in-design-1078068.

[4] "Changing colorspaces," OpenCV Open Source Computer Vision. (2022), [Online]. Available: https://docs.opencv.org/3.4/df/d9d/tutorial_py_colorspaces.html.

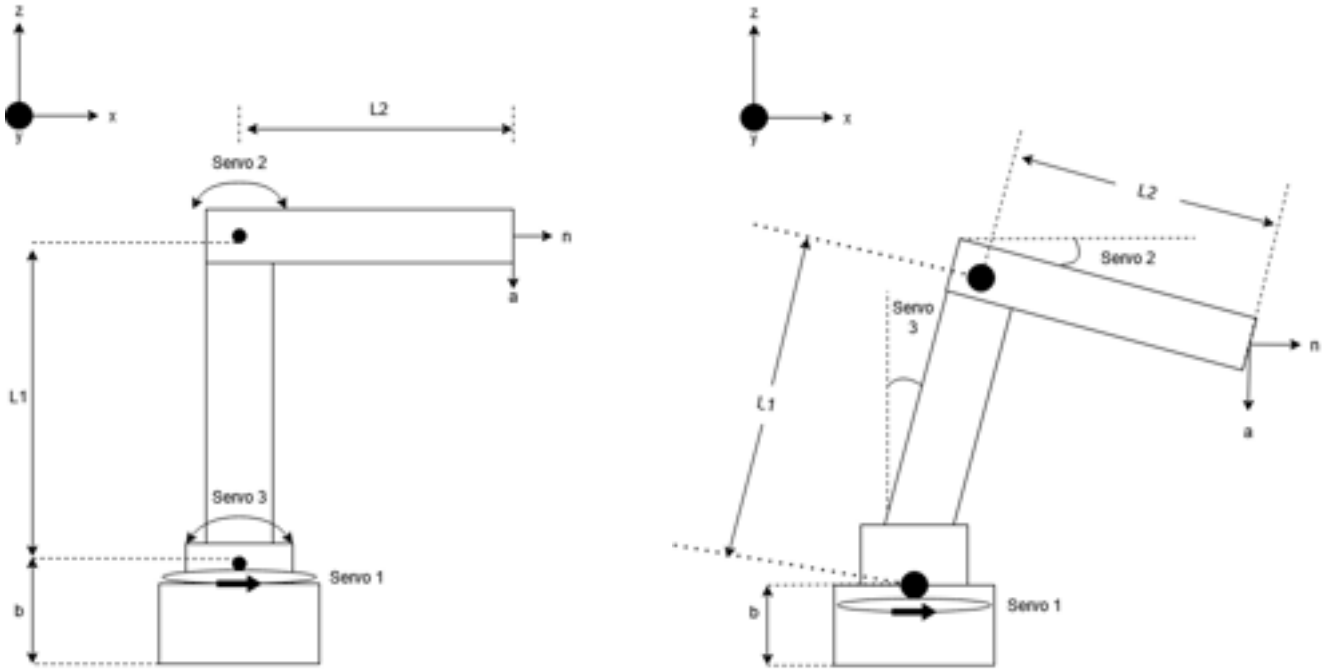
[5] R. Hessmer, *Kinematics for lynxmotion robot arm*, 2009. [Online]. Available: http://www.hessmer.org/uploads/RobotArm/Inverse%2520Kinematics%2520for%2520Robot%2520Arm.pdf.

Appendix

A Kinematics Derivations

A.1 Forward Kinematics

Deriving the forward kinematics is relatively straight-forward. We trigonometry to express the components of the end-effector's position in terms of the robot's link lengths.



(a) Reset Position

(b) Rotated Position

Figure A.1: Lynxmotion 3DoF Simplified Model

At $\theta_1 = 0$ and arbitrary θ_2, θ_3, the components of the end-effector position are:

$$x = l_1 \sin\theta_3 + l_2 \cos\theta_2$$
$$y = 0$$
$$z = l_1 \cos\theta_3 + l_2 \sin\theta_2$$

Based on how the coordinate system is defined, when $\theta_1 = \frac{-\pi}{2}$, the coordinates are:

$$
\begin{aligned}
x &= 0 \\
y &= l_1 \sin\theta_3 + l_2 \cos\theta_2 \\
z &= l_1 \cos\theta_3 + l_2 \sin\theta_2
\end{aligned}
$$

We can see that the x and y coordinates are similar save for the rotation angle of θ_1. This makes sense due to servo 1's circular rotation. Therefore, we can write:

$$
\begin{aligned}
x &= (l_1 \sin\theta_3 + l_2 \cos\theta_2)\cos\theta_1 \\
y &= -(l_1 \sin\theta_3 + l_2 \cos\theta_2)\sin\theta_1 \\
z &= l_1 \cos\theta_3 + l_2 \sin\theta_2
\end{aligned}
$$

However, from testing, we know that θ_3 has flipped behavior such that negative angles send the arm outward and positive angles make the arm contract. Noting this, the formula can be adjusted by $\theta_3 = -\theta_3$:

$$
\boxed{\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} (-l_1 \sin\theta_3 + l_2 \cos\theta_2)\cos\theta_1 \\ (l_1 \sin\theta_3 - l_2 \cos\theta_2)\sin\theta_1 \\ l_1 \cos\theta_3 + l_2 \sin\theta_2 \end{bmatrix}} \tag{A.1}
$$

A.2 Inverse Kinematics

The first step in the inverse kinematics derivation is to draw the figure with the angles used by Dr. Hessmer [5]. As shown in Figure A.2, θ_a is the angle that link 2 makes with link 1. The components of the end-effector position must be expressed using this angle in order to follow the method outlined in Dr. Hessmer's article.

We observe the case where θ_1, the angle responsible for rotation about the z-axis, is 0. For some arbitrary θ_3 and θ_a, we can express the x and z components of the end-effector's position as:

$$
\begin{aligned}
x &= l_1 \sin\theta_3 + l_2 \sin(\theta_3 + \theta_a) \\
z &= l_1 \cos\theta_3 + l_2 \cos(\theta_3 + \theta_a)
\end{aligned}
$$

We can use the law of cosines and the sum and difference trigonometry formulas to for simplification to receive:

$$
\begin{aligned}
x^2 + z^2 &= l_1^2 + l_2^2 + 2l_1 l_2 \cos\theta_a \\
\therefore \cos\theta_a &= \frac{x^2 + z^2 - l_1^2 - l_2^2}{2l_1 l_2}
\end{aligned}
$$

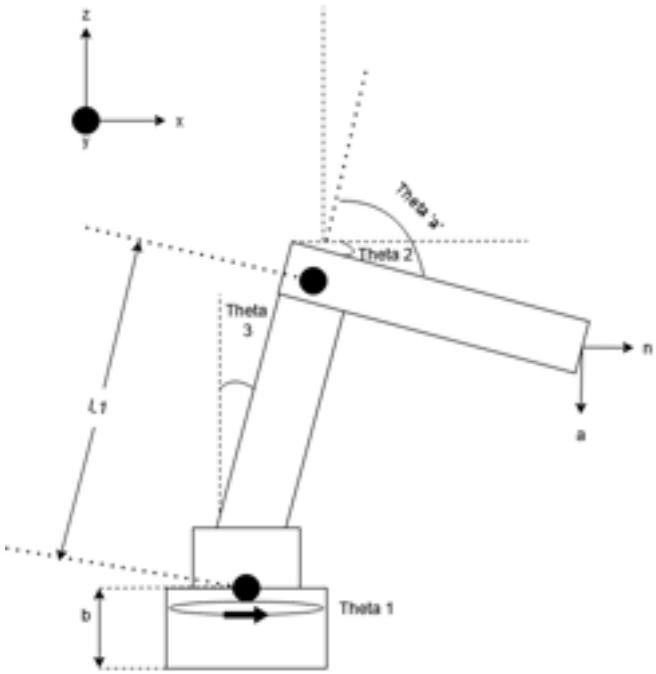


Figure A.2: Simplified Lynxmotion Model with relevant angles. Note the inclusion of θ_a which is the angle link 2 makes with link 1.

Rather than use the inverse cosine function, Dr. Hessmer uses ATAN2 to derive θ_a, noting that we can express $\sin\theta_a$ as $\pm\sqrt{1-\cos^2\theta_a}$:

$$\theta_a = ATAN2(\sqrt{1-\cos^2\theta_a}, \cos\theta_a) \tag{A.2}$$

After deriving θ_a, change-of-variables was used in order to isolate and directly solve for θ_3. Following Dr. Hessmer's process and using trial-and-error, it was determined that θ_3 is expressed as

$$\theta_3 = ATAN2(z, x) - ATAN2(k_2, k_1) \tag{A.3}$$

where $k_1 = l_2 + l_1\cos\theta_a$ and $k_2 = l_1\sin\theta_a$.

For the Lynxmotion robot specifically, we can substitute, $z^{'} = p_z - b$ for **z** where 'b' is the height from the ground to the robot base and $d = \sqrt{p_x^2 + p_y^2}$ for **x**. Doing so, we determine that the equations for the Lynxmotion Inverse Kinematics are:

$$
\begin{aligned}
\theta_1 &= -ATAN2(p_y, p_x) \\
\theta_2 &= \theta_a + \theta_3 - \frac{\pi}{2} \\
\theta_3 &= ATAN2(z^{'}, d) - ATAN2(k_2, k_1) \\
z^{'} &= p_z - b \\
d &= \sqrt{p_x^2 + p_y^2} \\
k_1 &= l_2 + l_1 \cos\theta_a \\
k_2 &= l_1 \sin\theta_a \\
\cos\theta_a &= \frac{d^2 + z^{'2} - l_1^2 - l_2^2}{2 l_1 l_2} \\
\theta_a &= ATAN2(\sqrt{1 - \cos^2\theta_a}, \cos\theta_a)
\end{aligned}
\tag{A.4}
$$

for a user-specified (p_x, p_y, p_z). Fortunately, these equations are easily implemented into a Python program script making the calculation simple.

B Code

Included below are the code excerpts for both color detection and kinematics.

```
"""
Title: Color Detection
Author: Terrance Williams
Date: 31 October 2022
Description: This program takes input images using the user's laptop camera and tests said images for the presence of
red, yellow, green, and blue. Upon reaching the specified threshold, it outputs the detected color to the console.

Sources:
1. Pysource "Simple Color recognition with Opencv and Python" https://www.youtube.com/watch?v=t71sQ6WY7L4
2. CreepyD "How to Detect Colors in OpenCV [Python]" https://www.youtube.com/watch?v=cMJwqxskyek
3. OpenCV "Changing Color Spaces" https://docs.opencv.org/3.4/df/d9d/tutorial_py_colorspaces.html
"""

# Imports
import cv2 as cv
import numpy as np
import time


def send_color(pic_num):
    if pic_num == 0:
        return 'Red'
    if pic_num == 1:
        return 'Yellow'
    if pic_num == 2:
        return 'Green'
    if pic_num == 3:
        return 'Blue'


def img_mask(image):
    # input raw image
    # outputs array of masked images (red, yellow, green, blue, original)
```

```
34      img_hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)
35
36      # Generate HSV Thresholds
37      red_thresh1 = np.array([[0, SATURATION_LOWER,
    BRIGHTNESS_LOWER],
38                              [7, SATURATION_MAX,
    BRIGHTNESS_MAX]])
39      red_thresh2 = np.array([[165, SATURATION_LOWER,
    BRIGHTNESS_LOWER],
40                              [179, SATURATION_MAX,
    BRIGHTNESS_MAX]])
41      green_thresh = np.array([[40, SATURATION_LOWER,
    BRIGHTNESS_LOWER],
42                               [90, SATURATION_MAX,
    BRIGHTNESS_MAX]])
43
44      blue_thresh = np.array([[100, SATURATION_LOWER,
    BRIGHTNESS_LOWER],
45                              [140, SATURATION_MAX,
    BRIGHTNESS_MAX]])
46      yellow_thresh = np.array([[20, SATURATION_LOWER,
    BRIGHTNESS_LOWER],
47                                [35, SATURATION_MAX,
    BRIGHTNESS_MAX]])
48
49      # Generate Masks
50      red_mask1 = cv.inRange(img_hsv, red_thresh1[0],
    red_thresh1[1])
51      red_mask2 = cv.inRange(img_hsv, red_thresh2[0],
    red_thresh2[1])
52      # Combine red masks
53      red_mask = cv.bitwise_or(red_mask1, red_mask2,
    mask=None)
54
55      green_mask = cv.inRange(img_hsv, green_thresh[0
    ], green_thresh[1])
56      blue_mask = cv.inRange(img_hsv, blue_thresh[0],
    blue_thresh[1])
57      yellow_mask = cv.inRange(img_hsv, yellow_thresh[0
    ], yellow_thresh[1])
58
```

```
59      yellow_masked = cv.bitwise_and(image, image, mask
    =yellow_mask)
60      red_masked = cv.bitwise_and(image, image, mask=
    red_mask)
61      blue_masked = cv.bitwise_and(image, image, mask=
    blue_mask)
62      green_masked = cv.bitwise_and(image, image, mask=
    green_mask)
63
64      output = np.array([red_masked, yellow_masked,
65                         green_masked, blue_masked])
66
67      return output
68
69
70  if __name__ == '__main__':
71
72      # Define Globals
73      SATURATION_LOWER = 100
74      SATURATION_MAX = 255  # Max 'S' value in HSV
75      BRIGHTNESS_LOWER = 20
76      BRIGHTNESS_MAX = 255  # Max 'V' Value in HSV
77      consec_detections = 0  # Tracks how many
    consecutive images detect a given color
78      DETECTION_THRESH = 150  # Number of consecutive
    detects required to be sure the color is present.
79
80      # Set up webcam
81      cap = cv.VideoCapture(0)
82
83      # Capture images
84      while True:
85          _, img = cap.read()
86          img = cv.flip(img, 1)
87
88
    # ================================================
    ==========================
89          #     Detecting a color in a subregion:
90          #     1. Draw rectangle around desired
    subregion
```

```
91          #     2. Draw circle in center of subregion
    . The center of this circle is the
92          #     pixel we will analyze.
93          #     3. Segment the frame into a variable
    consisting solely of the subregion
94          #     4. Apply relevant masking and
    filtering
95          #     5. Perform bitwise AND operation to
    isolate the desired color.
96          #     6. If center pixel is not black, that
    color is present (in theory).
97          #
98
    # ================================================
    ============================
99
100         # 1. Draw rectangle
101         height, width, _ = img.shape
102         rect_TpLft = (0, int(7 * height / 8))
103         rect_BtmRght = (int(width / 4), height)
104         rect_color = (0, 255, 0)
105         rect_thickness = 2
106         image2 = np.copy(img)  # so we don't modify
    the original
107         image2 = cv.rectangle(image2, rect_TpLft,
    rect_BtmRght, rect_color,
108                               rect_thickness)
109         # 2. Draw circle
110         rect_center = (int(rect_BtmRght[0] / 2), int
    ((rect_TpLft[1] + height) / 2))
111         circle_rad = 20
112         circle_color = (255, 0, 255)
113         circle_thickness = 5
114         image2 = cv.circle(image2, rect_center,
    circle_rad, circle_color, circle_thickness)
115         # print(rect_TpLft, rect_center,
    rect_BtmRght)
116         # print(f'Center BGR is: {img[rect_center[-1
    ], rect_center[0]]}')
117
118         # 3. Define subregion and relevant
```

```
118 dimensions
119 
120         subregion = img[int(7 * height / 8):height,
    0:int(width / 4)]
121         sub_height, sub_width, _ = subregion.shape
122         sub_center = subregion[int(sub_height / 2),
    int(sub_width / 2)]
123         # print(sub_center)
124 
125         # 4. Apply masks & 5. bitwise AMD
126         subregion_hsv = cv.cvtColor(subregion, cv.
    COLOR_BGR2HSV)
127         masked_imgs = img_mask(subregion)
128 
129         # 6. Check center pixel and surrounding
    pixels to determine color sum
130 
131         i = 0  # Track which image has the color
132         # Define center
133         y, x = int(sub_height / 2), int(sub_width /
    2)
134         color_detected = False
135         for im in masked_imgs:
136             im_center = im[y, x]
137             # reach surrounding pixels from center
138 
139             im_area = np.array([im[y - 1, x - 1], im
    [y - 1, x], im[y - 1, x + 1],
140                                 im[y, x - 1],
    im_center, im[y, x + 1],
141                                 im[y + 1, x - 1], im
    [y + 1, x], im[y + 1, x + 1]])
142             num_detections = 0
143             for pixel in im_area:
144                 if np.sum(pixel) > 0:
145                     num_detections += 1
146                 # print(f'Pixels with color on Image
     {i}: {num_detections}')
147             if num_detections == len(im_area):  # do
     all pixels have a non-black color?
148                 # Color found
```

```
149                     color_detected = True
150                     break
151                 i += 1
152             if color_detected:
153                 consec_detections += 1
154                 color = send_color(i)
155                 print(f'Consecutive Detections:{
    consec_detections}. Color: {send_color(i)}')
156             else:
157                 print('No color found.')
158                 consec_detections = 0
159             subregion = cv.circle(subregion, (x, y), 15
    , circle_color, 5)
160 
161             # cv.imshow("Original", img)
162             # cv.imshow("Rectangle", image2)
163             cv.imshow("Subregion", subregion)
164             cv.moveWindow("Subregion", 0, 0)
165             cv.waitKey(1)
166             if consec_detections >= DETECTION_THRESH:
167                 print(f'Color detected on Image {i}:
    Color: {send_color(i)}.')
168                 break
169     print("Exiting...")
170     cv.destroyAllWindows()
171     cap.release()
172 
```

```
"""
Title: Lynxmotion 3DoF Kinematics
Author: Terrance Williams
Date: 31 October 2022
Description: Program functions which calculate the forward kinematics and inverse kinematics of the 3DoF Lynxmotion Robot Arm.
"""

import numpy as np

global l1, l2, b  # lynxmotion variables (link1 length, link2 length, ground to robot base distance)


def lynxmotion_fk(servo1, servo2, servo3, mode='rad'):
    if mode == 'deg':
        servo1 = np.radians(servo1)
        servo2 = np.radians(servo2)
        servo3 = np.radians(servo3)
    x = (-l1*np.sin(servo3) + l2*np.cos(servo2))*np.cos(servo1)
    y = (-l1*np.sin(servo3) + l2*np.cos(servo2))*np.sin(servo1)
    z = (l1*np.cos(servo3) + l2*np.sin(servo2))

    return np.array([x, y, z+b])


def lynxmotion_ik(px, py, pz):
    d = np.sqrt(px**2 + py**2)
    z_prime = pz - b
    cos_a = ((d**2 + z_prime**2 - l1**2 - l2**2)/(2*l1*l2)).round(5)
    # Rounded to prevent weird case for 45 degree angles
    # Servos 1, 2, 3 angles:  [  0.  45. -45.]

    theta_a = np.arctan2(np.sqrt(1-(cos_a**2)), cos_a)
```

```
33      theta_1 = -np.arctan2(py, px)
34
35      k1 = l2 + l1*cos_a
36      k2 = l1*np.sin(theta_a)
37
38      theta_2 = np.arctan2(z_prime, d) - np.arctan2(k2
    , k1)
39      theta_3 = theta_a + theta_2 - (np.pi/2)
40
41      return np.array([theta_1, theta_2, theta_3])
42
```

EML 6805 Project 3 Report

Terrance Williams

November 28, 2022

Contents

1 Introduction

In Project 1, the Lynxmotion 3DoF Robot Arm was assembled and tested for operation behavior. For Project 2, the robot's kinematic equations were derived and implemented in the form of a Python calculator script; a means for color detection using a laptop camera was also implemented. With the completion of these two projects, the robot system was primed for Project 3.

2 Project 3 Objectives

The main objective of Project 3 was the synthesis of the previous two projects. Since the robot was developed to be controlled via Arduino in Project 1, and the color detection system was implemented via Python in Project 2, the two components need the ability to interact with one another. This was achieved using serial communication.

3 Arduino-Python Communication

To establish serial communication between the two components, I developed the two programs' interactions separately. Meaning, both the Python program and the Arduino sketch's behavior when receiving/sending serial data were developed divorced from one another, tested, and finally—like two puzzle pieces—linked together once each program worked as expected[1].

3.1 Python

The Python end of the serial link was programmed using the *pySerial* library [1]. The pySerial object was configured to work at a 19200 baud rate using a laptop USB port as the COM channel. It then enters a listening loop, waiting for the Arduino to connect to the serial channel. After some initial testing, it was discovered that the two systems needed some form of synchronization; otherwise, the two programs have the chance of "dropping" communication, entering into endless loops. To perform this synchronization, the Python program is set to enter a *while* loop in which it checks for available serial data every two seconds. If the data is of the form: "LYNX: Associate Colors.\n", the program then establishes that the robot program is to start zmc exits the loop.

[1] I refer to this Python-Arduino communication as "pyduino"

From here, the Python program enters another *while* loop, this time for the duration of the Lynxbot's operation. In this loop, the program listens to the COM port for available serial data. If the data received is the user-defined signal conveying the Arduino is ready for input ('I'), the program enters its Color Detection mode where it captures video frames from the user's laptop camera until a color (red, yellow, green, or blue) is detected 110 consecutive times. Upon reaching this threshold, the program then sends a byte that communicates which color was detected. The correlation is as follows: [red, yellow, green, blue] = [0, 1, 2, 3] where the numeric values are transmitted as ASCII characters. After transmitting the data, the program returns to listening for the Arduino READY signal.

3.2 Arduino

Since serial communication is easily implemented in Arduino, the Arduino side of the pyduino serial communication was as simple as beginning the Serial communication at the pyduino baud rate (19200) and sending the aforementioned PROGRAM_START and READY signals to through the channels. The problem, however, came with the Lynxbot system's serial communication requirements.

The Arduino model used for this project is an Arduino Uno, which has only one hardware serial pinout. The Lynxbot system however, requires at least two serial communication channels: one for the Lynxmotion Smart Servo (LSS) serial bus and the other for the pyduino communication. Since the pyduino communciation occurs via USB, a hardware serial connection, it was necessary to implement another serial port for the LSS bus using the *SoftwareSerial* Arduino library [2][3].

Pins 8 and 9 on the Arduino were then reserved for the RX and TX lines of the SoftwareSerial communication, allowing the Arduino to both control the Lynxmotion servos (and the RC mini-gripper servo) on this serial line (115200 baud) and interact with the Python program[2].

Figure 1 is a screenshot of the Arduino program at work. Arduino IDE's Serial Monitor was used to pass inputs to the Lynxbot in place of the Python code. Many additional printouts were added as well to aid debugging and ensure proper functionality. The image shows the program starting with color association (elaborated on in the next section) which then begins its operation after the fourth position is filled. It then depicts one pick and place operation in which the color code for

[2]A large "thank you" to user-moderator *dialfonzo* on the RobotShop Community forum for his assistance in implementing this properly without system failure. His timely assistance on this matter was crucial to the success of this project.

the Red color was provided (the program assumes the object starts at the position associated with Green). It then outputs the servo angles associated with the object's current position and its target (using test values). Finally, after a RESET, Red was input again as the target color to which the program informs the user that the object is there already.

4 Project Operation Flow

The full program works as follows: Begin by running the Python code (let's call it Pybot). Pybot creates the necessary variables for the color detection and serial connection, entering into the first serial "checkpoint" afterward. At this point, the Arduino can be connected to the laptop via USB. The Arduino defines its necessary variables and creates the objects for controlling the LSS servos and the mini-gripper. During the setup function, the Arduino initializes the SoftwareSerial bus (toggling an LED on the system as a visual indicator) and moves the robot to its RESET position (all servos at 0°).

The Arduino then connects to the pyduino channel. At this point, Pybot detects this connection, exiting its first checkpoint and entering the second in which it waits for the PROGRAM_START signal. The Arduino sends this signal and then sends its READY signal after a brief delay. Pybot gets the PROGRAM_START message and exits checkpoint two. It then enters a perpetual loop of waiting for the READY signal, detecting a color, and transmitting the detected color data.

Arduino then associates the pick and place positions with a color. It begins with Position 1, then proceeds to define positions 2, 3, and 4. Each position gets a color (Red, Yellow, Green, or Blue)[3]. With this step, the "setup" is complete and the Arduino enters its loop function.

In said loop function, the Arduino listens to the pyduino channel for color detection data. If the color received is associated with the pick and place object's current position, the program does nothing and effectively discards this data. Otherwise, the program performs a pick and place operation, moving the object from its current position to the target position based on the provided color. The program then updates the pick and place object's position and sends the READY signal to Pybot for another color input. This cycle repeats as the user wishes. A high-level overview of the program flow is presented in Figure 2.

[3] At this time, I have not implemented a check for multiple positions having the same associated color for the setup phase. However, this problem is mitigated by user action, so it is not an issue for the current purpose.

```
18:42:53.794 -> LYNX: Associate Colors.
18:42:55.773 -> IPositions filled: 0
18:42:58.513 -> Data detected
18:42:58.603 -> Position 1 filled
18:42:58.644 -> IPositions filled: 1
18:42:58.804 -> Data detected
18:42:58.924 -> Position 2 filled
18:42:58.924 -> IPositions filled: 2
18:42:59.118 -> Data detected
18:42:59.203 -> Position 3 filled
18:42:59.238 -> IPositions filled: 3
18:42:59.434 -> Data detected
18:42:59.508 -> Position 4 filled
18:42:59.508 -> ICurrent position color is: GREEN
18:43:15.525 -> Target position color is: RED
18:43:16.215 -> Reset lynx servo positions and move time:
18:43:16.253 -> 0
18:43:16.253 -> 0
18:43:16.253 -> 0
18:43:16.253 -> 150
18:43:18.333 -> Servo 1 Angle (Lynx, current, target): 0, 0, 0
18:43:18.493 -> Servo 2 Angle: 0, 0, -300
18:43:18.693 -> Servo 3 Angle: 0, -300, 500
18:43:26.374 -> Reset lynx servo positions and move time:
18:43:26.374 -> 0
18:43:26.413 -> 0
18:43:26.413 -> 0
18:43:26.413 -> 150
18:43:28.494 -> ICurrent position color is: RED
18:43:28.893 -> Object already there.
18:43:28.893 -> I
```

Figure 1: Output for the test version of the Arduino code. The 'I's throughout the image are the READY signals transmitted by the Arduino.

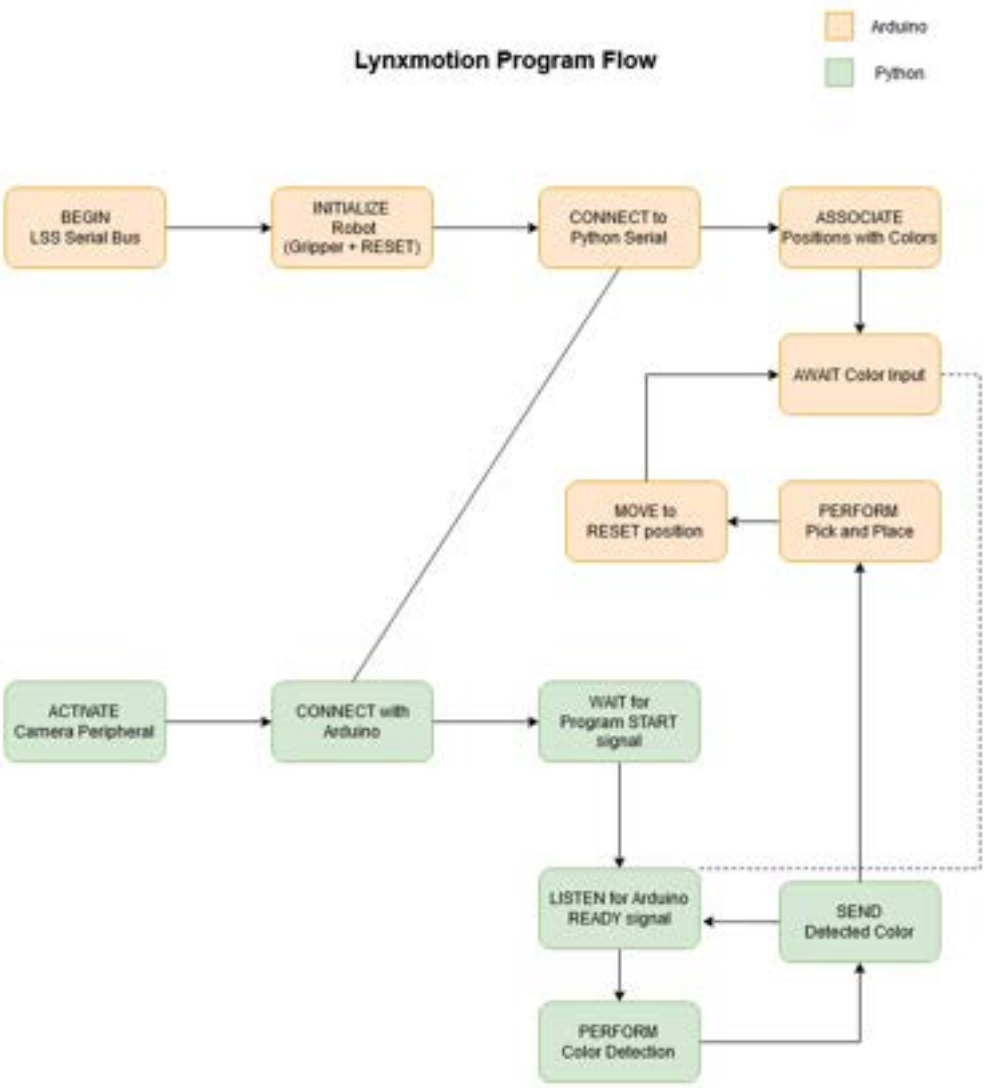


Figure 2: The figure shows the high-level programmatic flowchart for the entire Lynxmotion Robot Arm system. The Python portions are colored green, and the Arduino portions are orange. Note that the two programs are mostly isolated save for two connection points: the serial connection (solid, no arrow) and the READY signal point (dotted)

5 Operation Setup

The robot's environment construction had three components: base construction, pick and place object selection, and position defining. The project's base is a tri-fold cardboard presentation board. This was chosen due to the easy modification and inexpensive replacement of cardboard. Through testing done in the previous projects, it was observed that the robot would undergo extraneous motion during operation—such as the robot rotating about its o-axis—due to high torque. This movement would make calibration of the end-effector ineffective if not impossible because the robot's position in the universe frame would change at unknown values. As a result, the robot was secured in place by pinning the bot to a block of (level) wood via three nails that were hammered through the robot base's pinning holes. The fourth pinning hole was inaccessible due to the Arduino/LSS Adapter configuration, so this pin was secured via duct tape.

Figure 3: An image of the Lynxbot pinned to the cardboard.

The next step in the operation setup was the choice of the pick and place object. For this, it was desired that the object be rectangular to be easily handled by the robot and remain flat on the environment surface. The object also must be dense enough the prevent easy displacement but light enough to be manipulated consistently by the robot without straining the motors. With these conditions in mind, the object chosen for the project was a rubber eraser.

Finally, the positions (in universe frame) onto which the object will be placed were chosen. This step included much trial-and-error, accounting for the ease with which the robot arm could reach the position in terms of servo load while maintaining

visual spatial variety to showcase the robot's range. The inverse kinematics[4] developed in the previous project helped this process proceed smoothly, facilitating the testing of numerous positions until suitable ones were found. The selected positions and their corresponding servo angles are tabulated below. The position with decimal y-value (16, 13.6, 5.5) was chosen because it lies on a circle of a defined radius (21cm). Other positions were chosen to showcase the robot's ability to sweep to a range of positions.

Position (x, y, z) (cm)	Servo Angles (°) (Servo 1, Servo 2, Servo 3)
(15, -10, 5.5)	(33.7, -42.7, 17.5)
(16, 13.6, 5.5)	(-40.4, -39, 27.5)
(19, 0, 5.5)	(0, -41.6, 20.8)
(24, 9, 7)	(-20.5, -27.4, 40.5)

Table 1: Chosen positions and their corresponding angles.

[4]It was observed that for greater distances in the x-direction (Ex. 24cm), the end-effector traveled further vertically than shorter distances (16cm). As a result, greater distances were calculated at increased z-values such as 6.5cm or 7 cm rather than 5.5cm. It is unknown why this behavior occurred; perhaps the link measurements require more accuracy and precision.

With the positions defined, the user can choose the colors to associate with these positions and operate the robot as discussed previously. Diagrams displaying example configurations are displayed below.

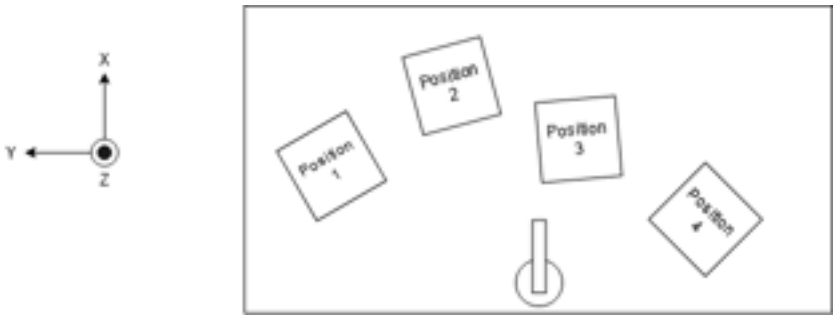
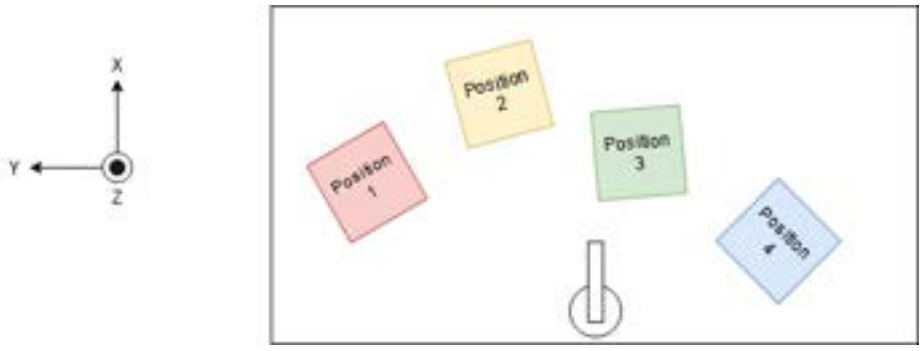
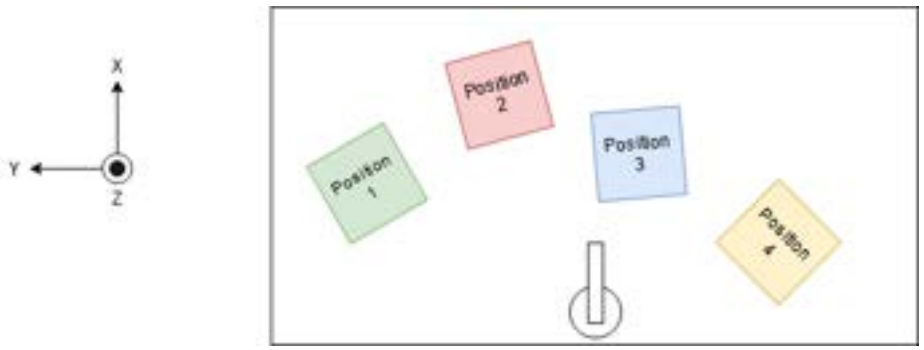


Figure 4: Basic project setup (Top View).



(a) Example color configuration.



(b) Alternate color configuration

6 Results

After completing the code and constructing the environment, the project was tested for operation. While the robot was able to interact when the eraser when the eraser was properly aligned, the orientation of the eraser had a larger margin for error due to its width. As a result, the eraser is now used such that it rests on its longest face, minimizing the width of the piece that the gripper must hold. This resulted in more consistent pick and place results that allowed room for user placement errors.

The robot was tested in a number of color configurations. It behaved as desired for each configuration with little problems. Occasionally, Servo 3 did result in the current overload, but this is a problem that has appeared sparingly since Project 1. After a reset (unplugging the Arduino from the laptop and toggling the 12V robot power supply), this problem appears to disappear.

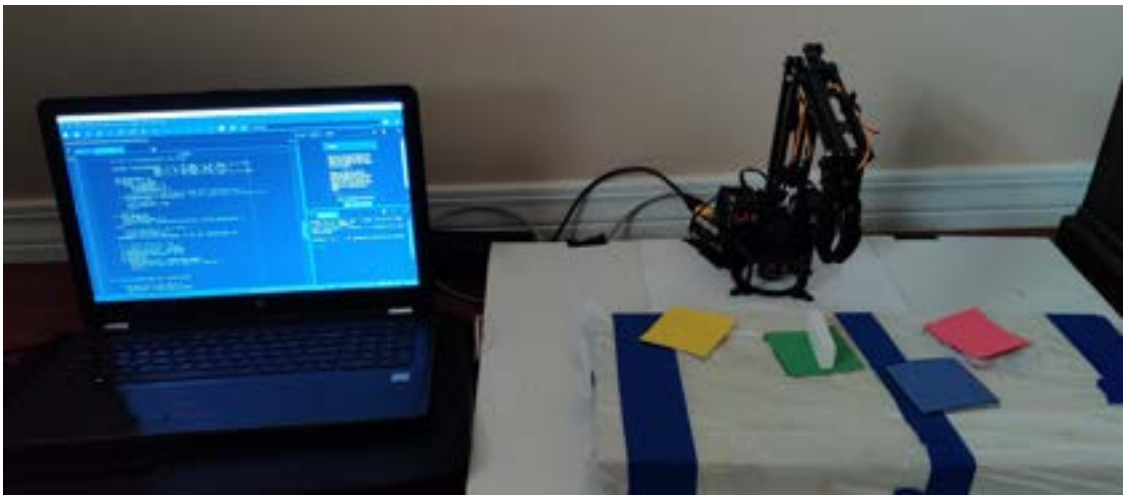


Figure 6: An image of the Lynxbot system with Color Detection.

7 Conclusion

Overall, the robot system works as expected, being able to asscociate colors with the defined positions, manipulate the eraser object between positions, and doing so consistently while maintaining satisfactory operating conditions such as LSS, mini-gripper, and Arduino temperatures. As a result, this project may be considered a success.
The code for both components of pyduino as well the kinematics may be found at the following link: https://github.com/tjdwill/Lynxbot

References

[1] C. Liechti, *Pyserial api*, 2020. [Online]. Available: https://pyserial.readthedocs.io/en/latest/pyserial_api.html.

[2] Arduino, *Softwareserial library: Arduino documentation*, 2022. [Online]. Available: https://docs.arduino.cc/learn/built-in-libraries/software-serial.

[3] dialfonzo of RobotShop, *Lynxmotion 3dof robot arm: Replacing 2io adapter w/ arduino*, 2022. [Online]. Available: https://community.robotshop.com/forum/t/lynxmotion-3dof-robot-arm-replacing-2io-adapter-w-arduino/94416.