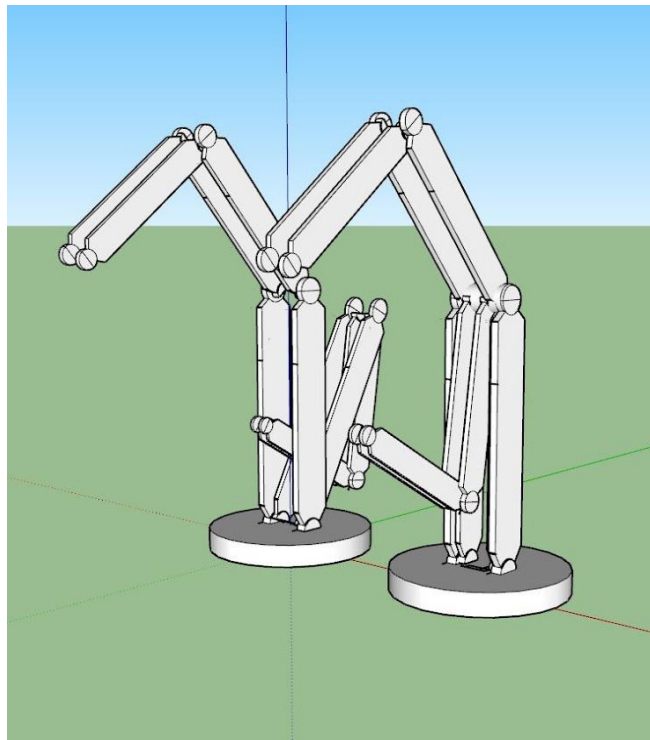


Automated Robotic Manipulator Integrated Engineering (A.R.M.I.E) Report

*Project On A Hollow Robotic Arm Pair Capable Of
Interaction With Environment With Increased
Workspace*



Erdong Xiao - N12456325
Armando Granado - N19941172
Sahil Kumar - N18120540

ME-GY 6933
Professor Vikram Kapila
Spring 19

INDEX

1. Background	3
2. Goal of the project	6
3. Methods	7
3.1 Design	7
3.2 Research	8
3.3 Kinematics	9
3.4 Inverse Kinematics	10
3.5 Workspace	10
3.6 Object Recognition Using OpenCV	11
3.7 Hardware Prototype	13
3.8 Components Used	14
3.9 Software Used	14
4. Conclusion	14
5. Applications	15
6. Future Work	15
7. Citation	15
8. Appendix	16
8.1 codes for the smaller arm	16
8.2 codes for the smaller arm	19

1. Background

Manipulator Arms:

An automated arm is a kind of mechanical arm, normally programmable, with comparative capacities to a human arm; the arm might be the aggregate of the system or might be a piece of a progressively perplexing robot. The connections of such a controller are associated by joints permitting either rotational movement, (for example, in an explained robot) or translational (straight) displacement. The connections of the controller can be considered to shape a kinematic chain. The end of the kinematic chain of the controller is known as the end effector and it is undifferentiated from the human hand



Types:

- Cartesian:

This robot arm has 3 prismatic joints.

- Cylindrical:

The axes of this robot forms a cylindrical coordinate system.

- Spherical robot:

It's axes form a polar coordinate system.

- SCARA robot:

Used for work such as pick and place by two parallel rotary joints.

- Articulated robot:

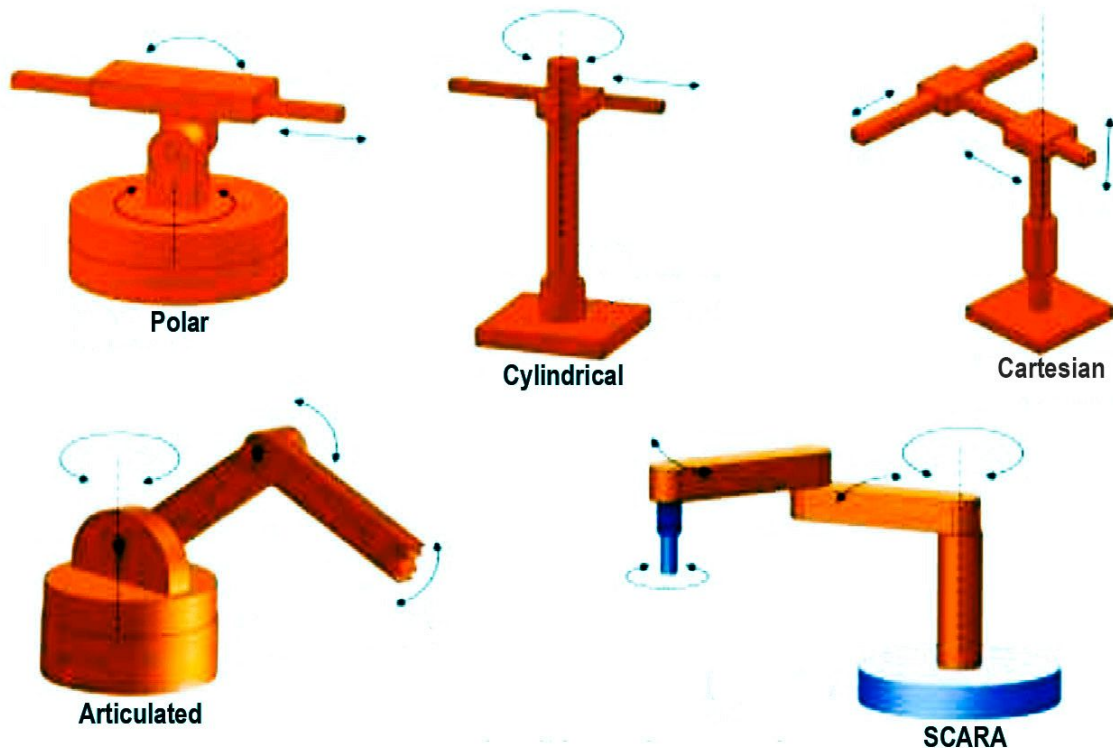
A robot whose arm has in any event three rotational joints.

- Parallel robot:

It's arms have concurrent prismatic or rotary joints.

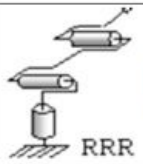


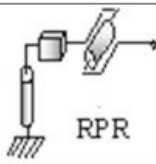
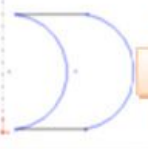


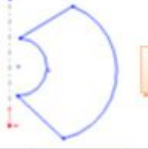

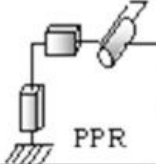
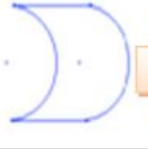

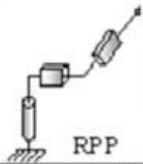


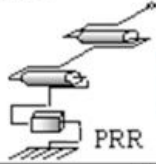


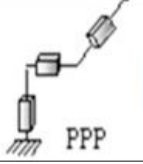
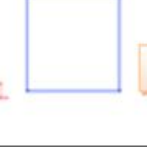
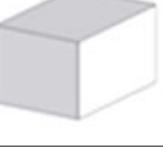
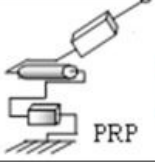
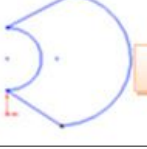

- Anthropomorphic robot:

A robot that is similar to a human hand.



Robot workspace:

- *Robot Workspace* is that volume of room that the end-effector of the robot or controller can reach. For an answer to exist, the predefined objective point must exist in the workspace. It is helpful to think about two meanings of workspace:
- *Dextrous workspace* is that volume of space that the robot end-effector can reach with all orientations. That is, at each point in the dextrous workspace, the end-effector can be arbitrarily oriented.
- *Reachable workspace* is that volume of space that the robot can reach in at least one orientation.
- Clearly, the dextrous workspace is a subset of the reachable workspace.
- The Monte Carlo Method is also a very famous method in the reinforcement learning, which can have a very powerful application when the transformation probabilities are not known. We can use the Monte Carlo method to generate the workspace.

Mechanism	2D Workspace	3D Workspace	Mechanism	2D Workspace	3D Workspace
 RRR			 RPR		
 RRP			 PPR		
 RPP			 PRR		
 PPP			 PRP		

2. Goal of the project

Build a robotic manipulator arm that is unique and advantageous all the same. A hollowed out robotic manipulator pair that can work together to not only conserve time and power but also improve the workspace that the robot can go to.

The robotic arm would also need to be smartly designed so as to capture images and use OpenCV to detect the object in front of the robot to be picked up and handed off to the other arm. This can have multifaceted uses and help in many ways.

OpenCV would pickup the location of the object and hand it over to the propeller which would then use inverse kinematics to go to that point and pickup the object and hand it off to the other arm behind it.

When not being used for this, the 2 arms would be capable of working individually.

3. Methods

- Design.
- Research
- Use DH convention to find kinematics.
- Find inverse kinematics using 2D planar model.
- Figuring out the workspace using the Monte Carlo method.
- Object recognition using OpenCV.
- Create hardware prototype.

3.1 Design

The design was originally designed using Simscape Multibody on MATLAB and then refined to give a clear understanding of how the robot would work in real life scenarios.

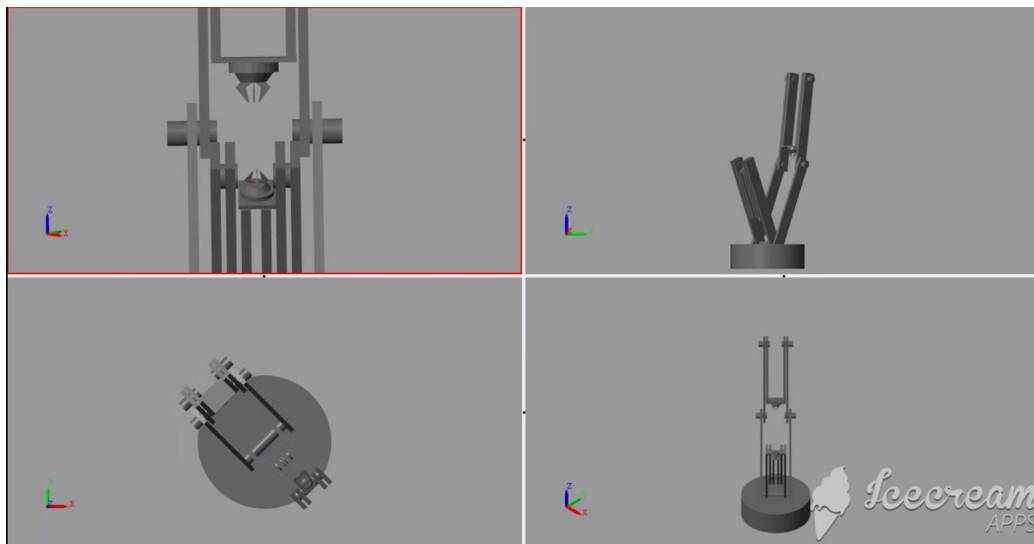
The arm links were designed with the following lengths:

Bigger Arm:

- 9 inches - Link 1
- 6.5 inches - Link 2
- 9 inches - Link 3 including gripper

Smaller Arm:

- 9 inches - Link 1
- 11.5 inches - Link 2 including gripper.



3.2 Research

For Materials and Structure, the research paper “Design, analysis and fabrication of robotic arm for sorting of multi-materials” was studied and it was found that metal is the common material in robots due to its strength

along with rigidity. Various papers were also used to find the stress factor, motor type and mechanisms to be taken into consideration.

For this project, acrylic plastic was used as the material due to its strength and ability to perform basic tasks and prove validity. Along with that standard servos with appropriate weight lifting capabilities were also used.

For the Physics calculations, research papers were used to consult for the torque calculations. As for the design process, a research paper named “Design Analysis And Fabrication Of Robotic Arm For Sorting Of Multi Materials” was used which gave insight on how big companies and how their robotic product development works. This was a huge step as it gave a step by step process of the right way to conduct such research.

3.3 Kinematics



The DH convention was used as shown to find the kinematic solution to this robot. We design a rotation joint in the base, and for the torso of the manipulators we design them as a 3 degree of freedoms planar arms. The DH parameters are as shown above. We multiply each of the

Transformation Matrix together, and then in the result matrix the upper left corner 3x3 matrix will reflect the orientation of the end-effector and the upper right corner 3x1 vector will reflect the position of the end-effector.

3.4 Inverse Kinematics

$${}^0T_3 = {}^0T_1 {}^1T_2 {}^2T_3 = \begin{bmatrix} C_1 & -S_1 & 0 & L_1 C_1 \\ S_1 & C_1 & 0 & L_1 S_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_2 & -S_2 & 0 & L_2 C_2 \\ S_2 & C_2 & 0 & L_2 S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_3 & -S_3 & 0 & L_3 C_3 \\ S_3 & C_3 & 0 & L_3 S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} C_{123} & -S_{123} & 0 & L_1 C_1 + L_2 C_{12} + L_3 C_{123} \\ S_{123} & C_{123} & 0 & L_1 S_1 + L_2 S_{12} + L_3 S_{123} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{cases} x^* = L_1 C_1 + L_2 C_{12} + L_3 C_{123} \\ y^* = L_1 S_1 + L_2 S_{12} + L_3 S_{123} \end{cases} \Rightarrow \begin{cases} x^* - L_3 C_{123} = L_1 C_1 + L_2 C_{12} \\ y^* - L_3 S_{123} = L_1 S_1 + L_2 S_{12} \end{cases}$$

Suppose $\begin{cases} x = x^* - L_3 C_{123} \\ y = y^* - L_3 S_{123} \end{cases}$

$$\Rightarrow \begin{cases} x = L_1 C_1 + L_2 C_2 \quad \text{--- (1)} \\ y = L_1 S_1 + L_2 S_2 \quad \text{--- (2)} \end{cases} \xrightarrow{\text{Square and add}} x^2 + y^2 = L_1^2 + L_2^2 + 2 L_1 L_2 C_2$$

$$\Rightarrow C_2 = \frac{x^2 + y^2 - L_1^2 - L_2^2}{2 L_1 L_2}, \quad S_2 = \sqrt{1 - C_2^2} \text{ (always allow up)}, \quad \text{det } \alpha_2 = \arcsin(S_2) \in [-\pi/2, \pi/2]$$

$$\begin{cases} x = L_1 C_1 - L_2 S_2 S_1 + L_3 C_1 \\ y = L_1 S_1 + L_2 C_2 S_1 + L_3 S_1 \end{cases} \Rightarrow \begin{cases} x = (L_1 C_1 + L_3) C_1 - L_2 S_2 S_1 \\ y = L_1 S_1 + (L_2 C_2 + L_3) S_1 \end{cases}$$

$$\Rightarrow \frac{y}{x} = \frac{(L_1 C_1 + L_3) C_1 + L_2 S_2}{(L_1 C_1 + L_3) C_1 - L_2 S_2 S_1} \Rightarrow -\left(\frac{y}{x}\right) L_2 S_2 S_1 + (L_1 C_1 + L_3) \left(\frac{y}{x}\right) = (L_1 C_1 + L_3) C_1 + L_2 S_2$$

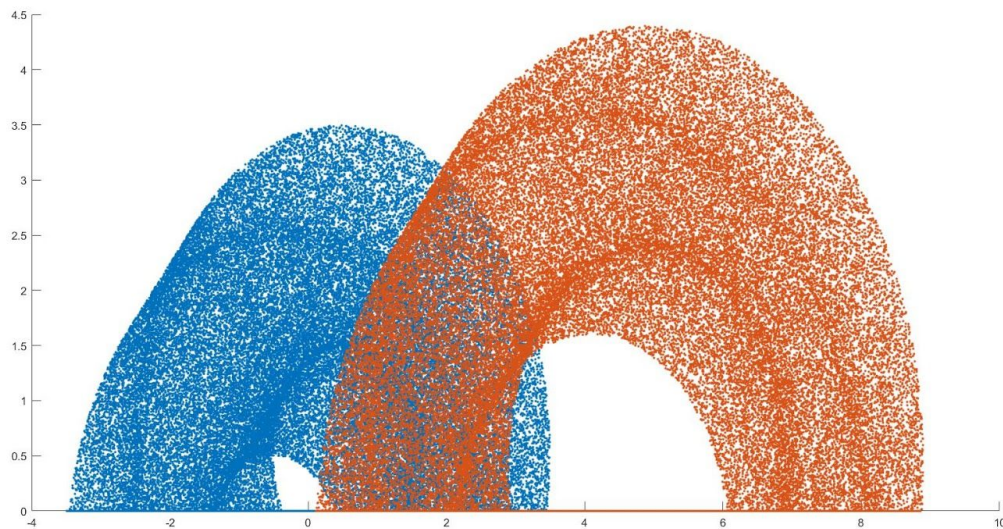
$$\Rightarrow t_1 = \frac{(L_1 C_1 + L_3) \left(\frac{y}{x}\right) - L_2 S_2}{(L_1 C_1 + L_3) + \left(\frac{y}{x}\right) L_2 S_2}, \quad \text{det } \alpha_1 = \arctan(t_1) \in (-\pi/2, \pi/2)$$

if $y < 0$, i.e. $y = -y$,
 $\text{det } \alpha_2 = -\arcsin(S_2), \quad \text{det } \alpha_1 = -\arctan(t_1)$

The 2D planar model was used as shown to find the inverse kinematic solution to this robot, since the third joint is designed to be actuated by DC motor, which is not convenient to do the position control. We supposed the third joint is just for doing the slight end-effector orientation adjustment, moreover, we suppose the desired orientation and the posture of the third link and joint are already known. Therefore, the theta1 and theta2 will be variables to be controlled and calculated. Based on that, we build a user interface, in which whenever we enter the desired position relative to the origins of the manipulators then the needed servo angles will be automatically calculated.

3.5 Workspace

The Monte Carlo method was used to determine the workspace using simulation. This method was based on a stochastic model, in this model the input is a large amount of random value joint variables arrays, since for each of the joint variables array we can use forward kinematics to calculate a possible position of the robot end-effector, the output is a point cloud cluttered by a lot of possible positions of the end-effector.



The strength of this model is we can distinguish the difference between the point in the workspace. For instance, for the locations where the points appear to be sparse, that reflect the singularity. On the contrary, for the locations where the points appear to be dense, that means in that places, the robot arm have higher flexibility.

The other strength of the Monte-Carlo method is it could make the workspace easier to be visualized and solved, compared to those traditional geometry methods.

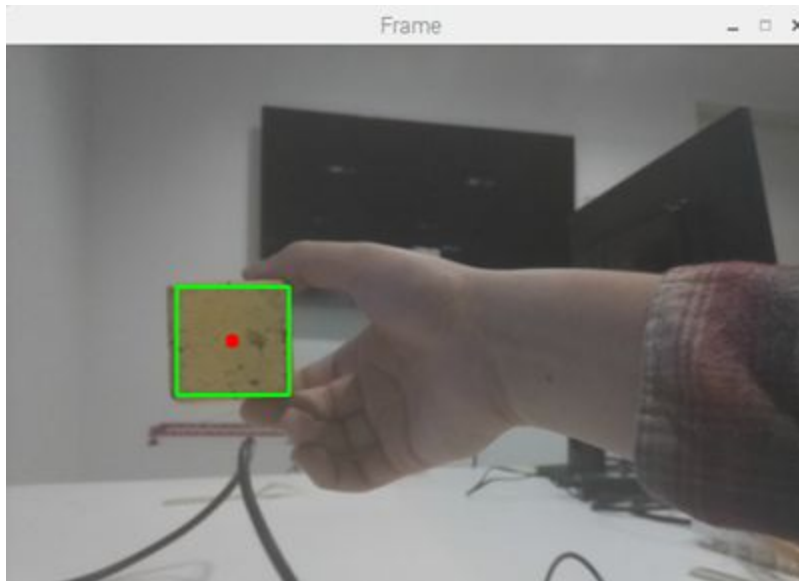
3.6 Object Recognition Using OpenCV

Using opencv we were able to blur the camera frames and resize images taken from pycam. The raspberry pi was able to do this in real time. Its maximum detection was 9 frames per second, but it was enough for our system. After the frame was blurred we can convert it to hsv color space. Next we perform a mask to only retain hsv ranges to allow yellow, green, and red objects. To improve accuracy we then erode and dilate this same frame.

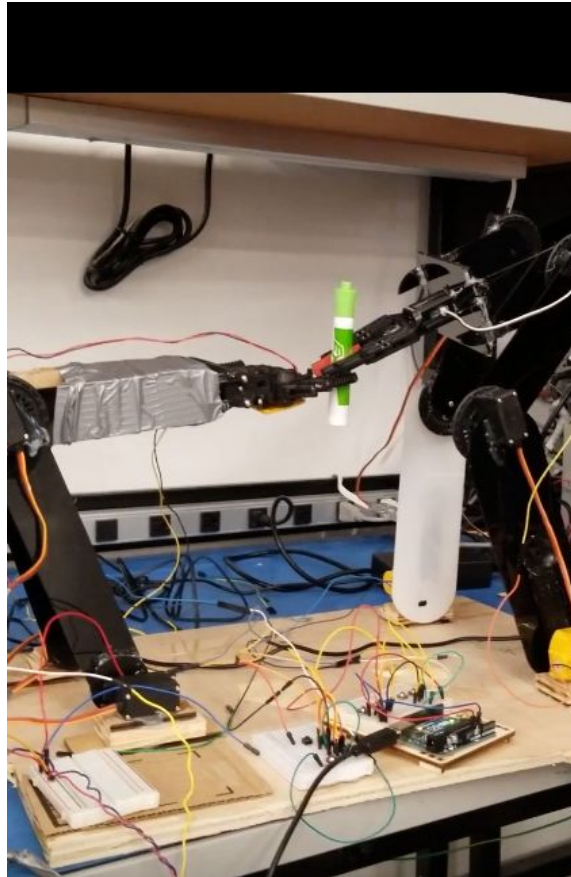


From here opencv is able to detect the contours of the above image. After this opencv is able to fit a square around the object. Then a small calculation can be done to calculate the center of the object from the rectangle's position, width, and height. However, the center location found by opencv is the pixel location. Therefore, the camera must be fixed to include the proper workspace of the initial robot arm. Once the proper camera location is found, which encapsulates the whole workspace, it is recorded. Then after taking four measurements, we can create a proper equation to convert pixel location to real world (x,y) positions relative the base frame. $0.0332 \cdot X_{\text{pixel}} - 2.6758 = X_{\text{location (in.)}}$, and $-0.0424 \cdot Y_{\text{pixel}} + 17.456 = Y_{\text{location(in.)}}$. Now that the center is calculated we are able to

publish the center point data to Propellor. By using asynchronous communication via the rx and tx pins on raspberry pi.



3.7 Hardware Prototype



3.8 Components Used

- 10 Acrylic plastic links.
- 8 Standard servo motors.
- 2 DC motors.
- 2 DC motor enabled grippers.
- 1 Arduino
- 1 Raspberry Pi
- 1 RaspiCam

3.9 Software Used

- Propeller IDE

- Arduino IDE
- OpenCV
- Python

4. Conclusion

A hollow robotic manipulator was the main objective. The application of the arm would be great for environments in which two manipulators are needed. We discuss this more in depth in the next part. However, the robot requires much power to actuate. Also, stronger motors and a redesign are need for this prototype. mounting the motors on the link adds too much weight to the system. If the motors could be mounted on the floor and a system was created to actuate the joints the robot would require less power, and have a greater size workspace. If a rigid link could connect the parallel joints, then half motors would be needed as well. The rigid joint would still preserve the hollowness, which would allow one arm to go in between the other.

Propellor is more than sufficient in powering the motors. By utilizing the cogs, we can actuate all motors at the same time. thus reducing lag and maintain rigidness between parallel joints. Arduino was proper to control the h-bridges needed for the system's DC motors. Finally, raspberry pi had enough ram to process the center point of the object and send it propellor. If more time was had the integration of all three would work smoothly.

5. Applications

- Required Mobility (KUKA Omnirob),
- Required Precision (Da Vinci Surgical Robot),
- Quality and quantity of work for low cost,
- Low power consumption.
- Surgical robots - Doctor Nurse Pair.

- Some applications might require just speed (Smaller arm), some just strength (Bigger arm) and some both, with precision.

6. Future Work

- Making it structurally sound.
- Further research on applications and parts.
- Mount motors on the base, to avoid unnecessary added weight.
- rigid links between joints to half the amount of servo motors needed.

7. Citation

- [1] https://www.researchgate.net/publication/281642602_Design_analysis_and_fabrication_of_robotic_arm_for_sorting_of_multi-materials
- [2] https://www.researchgate.net/publication/289674591_DESIGN_AND_DEVELOPMENT_OF_A_MECHANISM_OF_ROBOTIC_ARM_FOR_LIFTING_PART5
- [3] http://www.cs.cmu.edu/~ylpark/publications/Shin_SAGE_IJRR_2010.pdf
- [4] http://vigir.missouri.edu/~gdesouza/Research/Conference_CDs/IFAC_ICINCO_2010/ICINCO/ICINCO/Robotics%20and%20Automation/Posters/ICINCO_2010_220_CR.pdf
- [5] http://users.ox.ac.uk/~kneabz/Stress4_mt07.pdf
- [6] <https://www.amci.com/industrial-automation-resources/plc-automation-tutorials/stepper-vs-servo/>
- [7] <http://www.robotoid.com/howto/materials-for-robot-building-an-introduction.html>
- [8] <https://skyciv.com/education/types-of-supports-in-structural-analysis/>
- [9] <http://www.robotics.stanford.edu/~ang/papers/icra11-LowCostCompliantManipulator.pdf>
- [10] <https://www.builditsolar.com/References/Glazing/physicalpropertiesAcrylic.pdf>
- [11] https://www.engineersedge.com/strength_of_materials.htm

8.Appendix

8.1 codes for the smaller arm

```
#include "simpletools.h" // Include simple tools
#include "servo.h"
unsigned int stack1[40+25];
static volatile int Acc,pin,angle1,angle2,angle11,angle22,ag0,ag1;
unsigned int stack2[40+25];
unsigned int stack3[40+25];
unsigned int stack4[40+25];

void motor1(void *par);
void motor2(void *par);

int main() // Main function
{
    set_direction(0,1);
    int gripper;

    // Add startup code here.
    ag0=900;
    ag1=900;
    //servo_angle(14,ag0);
    //servo_angle(15,ag0);
    //servo_angle(16,ag1);
    //servo_angle(17,ag1);

    while(1)
    {
```



```

float theta1;
float theta2;
float s2;
float c2;
float t1;
float L1;
float L2;
float x;
float y;
L1=9;L2=11.5;
//L1=6.5;L2=9;
print("Please enter the desired position:\n");
print("x=");
scanf("%f",&x);
print("y=");
scanf("%f",&y);

if(y>0)
{
c2=(pow(x,2)+pow(y,2)-pow(L1,2)-pow(L2,2))/(2*L1*L2);
s2=sqrt(1-pow(c2,2));
t1=((L2*c2+L1)*(y/x)-L2*s2)/((L2*c2+L1)+(y/x)*L2*s2);
theta2=asin(s2)*180/PI;
theta1=atan(t1)*180/PI;
}
if(y<0)
{
y=-y;
c2=(pow(x,2)+pow(y,2)-pow(L1,2)-pow(L2,2))/(2*L1*L2);
s2=sqrt(1-pow(c2,2));
t1=((L2*c2+L1)*(y/x)-L2*s2)/((L2*c2+L1)+(y/x)*L2*s2);
theta2=-asin(s2)*180/PI;
theta1=-atan(t1)*180/PI;
}

```

```

}
if(c2>1||c2<-1)
{print("the position is outside the workspace\n\n");}
else
{
print("the desired servo angles are:\n");
print("theta1=%f\n",theta1);
print("theta2=%f\n",theta2);
ag0=900+10*theta1;
ag1=900+10*theta2;
// cogstart(&motor1, NULL, stack3,sizeof(stack3));
// cogstart(&motor2, NULL, stack4,sizeof(stack4));
print("Do you want to use the gripper? If yes type 1, if no type 0.\n");
print("gripper=");
scanf("%d",&gripper);
print("\n");
if(gripper==1)
{
high(0);
high(1);
print("Sent\n");
pause(100);
low(1);
low(0);
}
if(gripper==0)
{
high(0);
low(1);
print("Sent2\n");
pause(100);
low(1);
low(0);
}

```

```

}
}
// Add main loop code here.
}

```

```

}

void motor1(void *par1)
{
servo_angle(14,ag0);
servo_angle(15,1800-ag0);
pause(10);
}
void motor2(void *par2)
{
servo_angle(16,ag1);
servo_angle(17,1800-ag1);
pause(10);
}

```

8.2 codes for the bigger arm

```

#include "simpletools.h" // Include simple tools
#include "servo.h"
unsigned int stack1[40+25];
static volatile int Acc,pin,angle1,angle2,angle11,angle22,ag0,ag1;
unsigned int stack2[40+25];
unsigned int stack3[40+25];
unsigned int stack4[40+25];

void motor1(void *par);

```

```

void motor2(void *par);

int main() // Main function
{
    set_direction(0,1);
    int gripper;

    // Add startup code here.
    ag0=900;
    ag1=900;
    //servo_angle(14,ag0);
    //servo_angle(15,ag0);
    //servo_angle(16,ag1);
    //servo_angle(17,ag1);


    while(1)
    {
        float theta1;
        float theta2;
        float s2;
        float c2;
        float t1;
        float L1;
        float L2;
        float x;
        float y;
        //L1=9;L2=11.5;
        L1=6.5;L2=9;
        print("Please enter the desired position:\n");
        print("x=");
        scanf("%f",&x);
    }
}

```

```

print("y=");
scanf("%f",&y);

if(y>0)
{
c2=(pow(x,2)+pow(y,2)-pow(L1,2)-pow(L2,2))/(2*L1*L2);
s2=sqrt(1-pow(c2,2));
t1=((L2*c2+L1)*(y/x)-L2*s2)/((L2*c2+L1)+(y/x)*L2*s2);
theta2=asin(s2)*180/PI;
theta1=atan(t1)*180/PI;
}
if(y<0)
{
y=-y;
c2=(pow(x,2)+pow(y,2)-pow(L1,2)-pow(L2,2))/(2*L1*L2);
s2=sqrt(1-pow(c2,2));
t1=((L2*c2+L1)*(y/x)-L2*s2)/((L2*c2+L1)+(y/x)*L2*s2);
theta2=-asin(s2)*180/PI;
theta1=-atan(t1)*180/PI;
}
if(c2>1||c2<-1)
{print("the position is outside the workspace\n\n");}
else
{
print("the desired servo angles are:\n");
print("theta1=%f\n",theta1);
print("theta2=%f\n",theta2);
ag0=900+10*theta1;
ag1=900+10*theta2;
// cogstart(&motor1, NULL, stack3,sizeof(stack3));
// cogstart(&motor2, NULL, stack4,sizeof(stack4));
print("Do you want to use the gripper? If yes type 1, if no type 0.\n");
print("gripper=");

```

```
scanf("%d",&gripper);
print("\n");
if(gripper==1)
{
high(0);
high(1);
print("Sent\n");
pause(100);
low(1);
low(0);
}
if(gripper==0)
{
high(0);
low(1);
print("Sent2\n");
pause(100);
low(1);
low(0);
}
}
// Add main loop code here.
}
```

```
}
```

```
void motor1(void *par1)
{
servo_angle(14,ag0);
servo_angle(15,ag0);
pause(10);
```

```

}
void motor2(void *par2)
{
  servo_angle(16,ag1);
  servo_angle(17,1800-ag1);
  pause(10);
}

```

8.3 codes for the grippers

```

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,INPUT);
  pinMode(9,INPUT);
  pinMode(10,INPUT);
  pinMode(10,INPUT);
  int i=0;
  int k=0;
  int a=0;
  int b=0;
  int c=0;
  int d=0;
  //digitalWrite(3,HIGH);
  //digitalWrite(2,LOW);
  //Serial.print(a);
  //delay(750);
}

```

```
    digitalWrite(2,LOW);
    digitalWrite(3,LOW);
    digitalWrite(4,LOW);
    digitalWrite(5,LOW);

}

void loop()
{
    digitalWrite(6,HIGH);
    digitalWrite(7,HIGH);

    //int a;
    int a = digitalRead(8);
    int b = digitalRead(9);
    int c = digitalRead(10);
    int d = digitalRead(11);

    int i=1;
    int k=1;
    // put your main code here, to run repeatedly:

    if(a==HIGH && b==HIGH)
    {
        int a = digitalRead(8);
        int b = digitalRead(9);

        Serial.print(a);
        Serial.print(b);

        digitalWrite(3,LOW);
        digitalWrite(2,HIGH);
        Serial.print(a);
```



```
delay(100);  
digitalWrite(2,LOW);  
digitalWrite(3,LOW);  
i=2;  
}
```

```
else if(a==HIGH && b==LOW)  
{int a = digitalRead(8);  
int b = digitalRead(9);
```

```
Serial.print(a);  
Serial.print(b);
```

```
digitalWrite(3,HIGH);  
digitalWrite(2,LOW);  
Serial.print(a);  
delay(100);  
digitalWrite(2,LOW);  
digitalWrite(3,LOW);  
i=2;  
}  
if(c==HIGH && d==HIGH)  
{  
int c = digitalRead(10);  
int d = digitalRead(11);
```

```
Serial.print(c);  
Serial.print(d);
```

```
digitalWrite(4,LOW);  
digitalWrite(5,HIGH);  
//Serial.print(c);  
delay(100);
```

```
digitalWrite(4,LOW);  
digitalWrite(5,LOW);  
k=2;  
}
```

```
else if(c==HIGH && d==LOW)  
{int c = digitalRead(10);  
int d = digitalRead(11);
```

```
Serial.print(c);  
Serial.print(d);
```

```
digitalWrite(4,HIGH);  
digitalWrite(5,LOW);  
//Serial.print(c);  
delay(100);  
digitalWrite(4,LOW);  
digitalWrite(5,LOW);  
k=2;  
}
```

```
}
```

8.4 Codes for OpenCv

```
from collections import deque  
from imutils.video import VideoStream  
import numpy as np  
import argparse  
import cv2  
import imutils
```

```

import time
import serial

ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video",
    help="path to the (optional) video file")
ap.add_argument("-b", "--buffer", type=int, default=64,
    help="max buffer size")
args = vars(ap.parse_args())
ser = serial.Serial(
    port='/dev/ttyS0',
    baudrate = 115200,
    parity = serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=1
)
# Boundaries
Lower = (15,86,6)
Upper = (180,255,255)
pts = deque(maxlen=args["buffer"])

vs = VideoStream(src=0).start()
time.sleep(2.0)

while True:

    frame = vs.read()

    frame = frame[1] if args.get("video", False) else frame

    if frame is None:

```

break

```
frame = imutils.resize(frame, width=600)
blurred = cv2.GaussianBlur(frame, (11, 11), 0)
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv, Lower, Upper)
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)
#cv2.imshow("Frame", mask)
cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
center = None
```

only proceed if at least one contour was found

if len(cnts) > 0:

find the largest contour in the mask

c = max(cnts, key=cv2.contourArea)

x, y, w, h = cv2.boundingRect(c)

cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

center = ((x+w/2),(y+h/2))

#print(center[0],center[1])

x2= center[0]*.0332-2.6578#-(800))/301

y2 = center[1]*(-.0424)+17.4576

x2 = str(x2)

x2 = x2[0:5]

y2 = str(y2)

y2 = y2[0:5]

buffer = (x2+": "+y2+":")

print(buffer)

ser.write(buffer.encode())

```

    RealCenter=(x2,y2);
    # print(RealCenter)

    cv2.circle(frame, center, 5, (0, 0, 255), -1)

    pts.appendleft(center)
    key = cv2.waitKey(1) & 0xFF

    # if the 'q' key is pressed, stop the loop
    if key == ord("q"):
        break
    # if we are not using a video file, stop the camera video stream
    vs.stop()
    # close all windows
    cv2.destroyAllWindows()

```

8.5 codes for the Monte Carlo Workspace

```

clear all;
clc;
theta1=-pi+2*pi*rand(1,100000);
theta2=-pi+2*pi*rand(1,100000);
theta3=-pi+2*pi*rand(1,100000);
% theta4=0+pi*rand(1,100000);

x=2*cos(theta1)+1*cos(theta1+theta2)+0.5*cos(theta1+theta2+theta3);
y=0.5*(2*sin(theta1)+1*sin(theta1+theta2)+0.5*cos(theta1+theta2+theta3))+
abs(2*sin(theta1)+1*sin(theta1+theta2)+0.5*cos(theta1+theta2+theta3)));
%
x=-3*sin(theta1)+1.5*cos(theta1).*cos(theta2)+1*cos(theta1).*cos(theta2+th
eta3)+0.4*cos(theta1).*cos(theta2+theta3+theta4);

```

```
%  
y=3*cos(theta1)+1.5*sin(theta1).*cos(theta2)+1*sin(theta1).*cos(theta2+the  
ta3)+0.4*sin(theta1).*cos(theta2+theta3+theta4);  
% z=1.5*sin(theta1)+1*sin(theta2+theta3)+0.4*sin(theta2+theta3+theta4);
```

```
scatter(x,y,'');  
hold on;  
x=3*cos(theta1)+1*cos(theta1+theta2)+0.4*cos(theta1+theta2+theta3)+4.5;  
y=0.5*(3*sin(theta1)+1*sin(theta1+theta2)+0.4*cos(theta1+theta2+theta3)+  
abs(3*sin(theta1)+1*sin(theta1+theta2)+0.4*cos(theta1+theta2+theta3)));  
  
scatter(x,y,'');
```