ROBOT ARM CONTROL WITH HAPTIC FEEDBACK

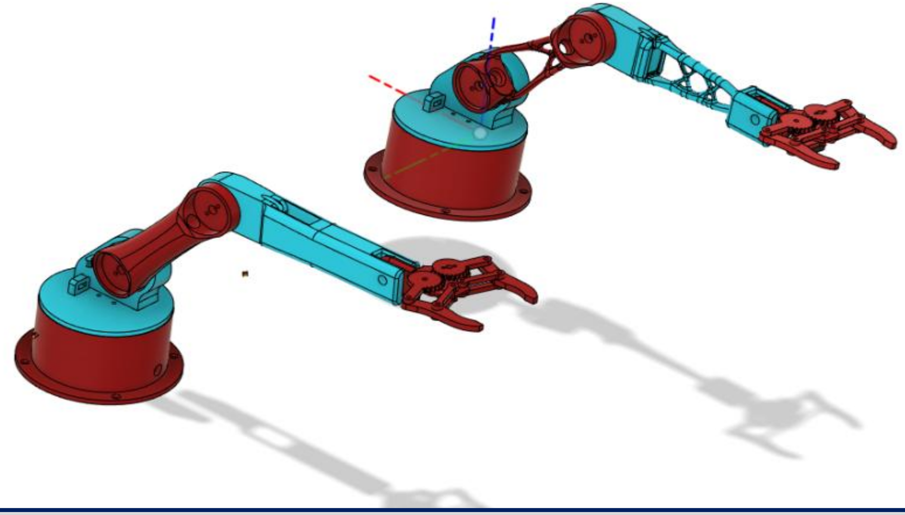
YILDIZ TECHNICAL UNIVERSITY

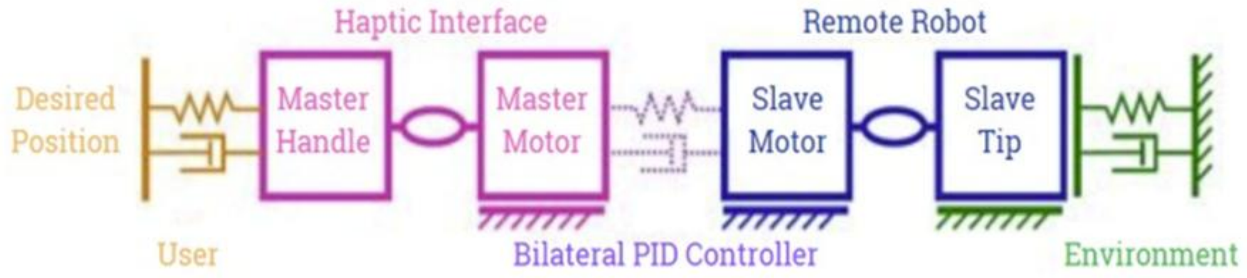
FACULTY OF MECHANICAL ENGINEERING

2006A601 Kadir Kadiroğlu   1806A038 Oğuzhan Can   1906A602 Özkan Yaşar

Project Advisor: Assoc. Prof. Hüseyin Ayhan Yavaşoğlu

INTRODUCTION





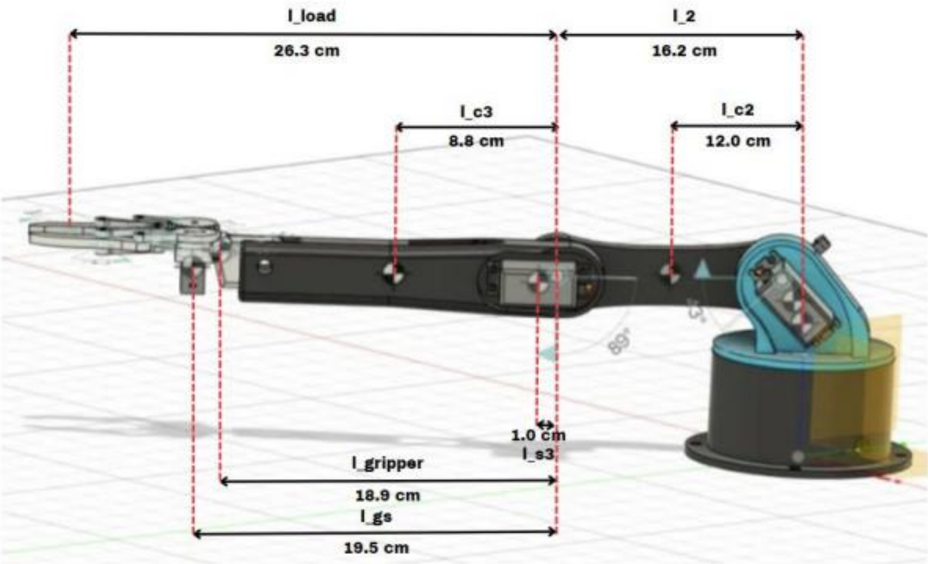
TECHNICAL REQUIREMENTS & DESIGN SPECIFICATIONS

Technical Requirements	Description
Force Feedback Integration	Maximum load capacity of 300 g and a maximum distance of 0.45 m are adhered to, thereby not surpassing the operator's natural force application capacity. The ability to adjust force transmission and position transmission ratios through different modes.
Enabling Remote Data Transmission	Fast (2.4 GHz) and reliable data transmission from a distance of 100 meters using RF modules.
Precision and Force Transfer	Achieving ±2 cm accuracy in linear movements, ±1 degree angular accuracy, and ±0.5 Newton force transmission accuracy

Optimization of Latency Duration	Minimizing the delay(approximately 35ms) between the operator's manipulations and the response of the controlled arm.
Security Protocols and Initialization Process	Safely determining the initial positions of the arms. Creating safety protocols and completing the initialization process within a maximum of 5 seconds.
Development of Control Algorithms	Implementing PID control algorithms to accurately transmit the operator's manipulations to the controller arm for haptic feedback integration Applying digital filters to dampen hand tremors(3Hz cut-off freq. -40dB/decade roll-off rate)
Integration and Compatibility	Facilitating easy integration of the robot arm with different devices Capability to be integrated into different platforms using standard attachments or adapters

Dynamic Equation Calculations

Design criteria for the most challenging scenario: The maximum pose of the arm



$$\tau = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} 0.05 \\ 1.25 \\ 0.85 \end{bmatrix} Nm$$

Based on these torque values:

- We've **selected** the needed actuators for each joint on the robot arms.
- We've **conducted** the stress analysis on the mechanical design part.

Bilateral Operation Equations

For the Slave Arm:

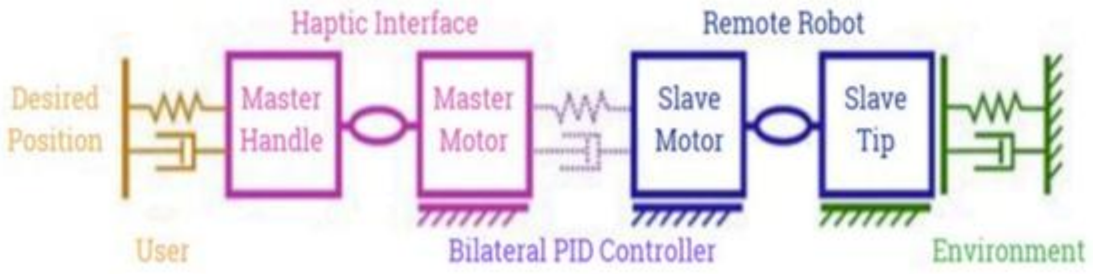
$$f_s(t) = k_{ps}(x_m - x_s) + k_{ds}(\dot{x}_m - \dot{x}_s) + k_{is}\int_0^t (x_m - x_s)\,dt$$

For the Master Arm:

$$f_m(t) = k_{pm}(x_s - x_m) + k_{dm}(\dot{x}_s - \dot{x}_m) + k_{im}\int_0^t (x_s - x_m)\,dt$$

Bilateral Teleoperation (Position Forward, Force Feedback):

$$f_m(t) = f_{env}$$



Battery Selection & Power Calculation

$$V_{Mg995} = 7.2V,\ I_{Mg995R_{stall-current}} = 1.2A$$

$$V_{SG90} = 7.2V,\ I_{SG90_{stall-current}} = 250\ mA$$

$$V_{AS5600} = 3.3V,\ I_{AS5600_{max-current}} = 10\ mA$$

$$V_{NRF24L01} = 3.3V,\ I_{NRF24L01_{max-current}} = 115\ mA$$

$$V_{Raspberry-Pi-Pico} = 3.3V,\ I_{Raspberry-Pi-Pico_{max-current}} = 300\ mA$$

$$3 * I_{Mg995R_{stall-current}} + 1 * I_{SG90_{stall-current}} + 4 * I_{AS5600_{max-current}} + 2 * I_{NRF24L01_{max-current}} + 2 * I_{Raspberry-Pi-Pico_{max-current}} = 4.72A$$

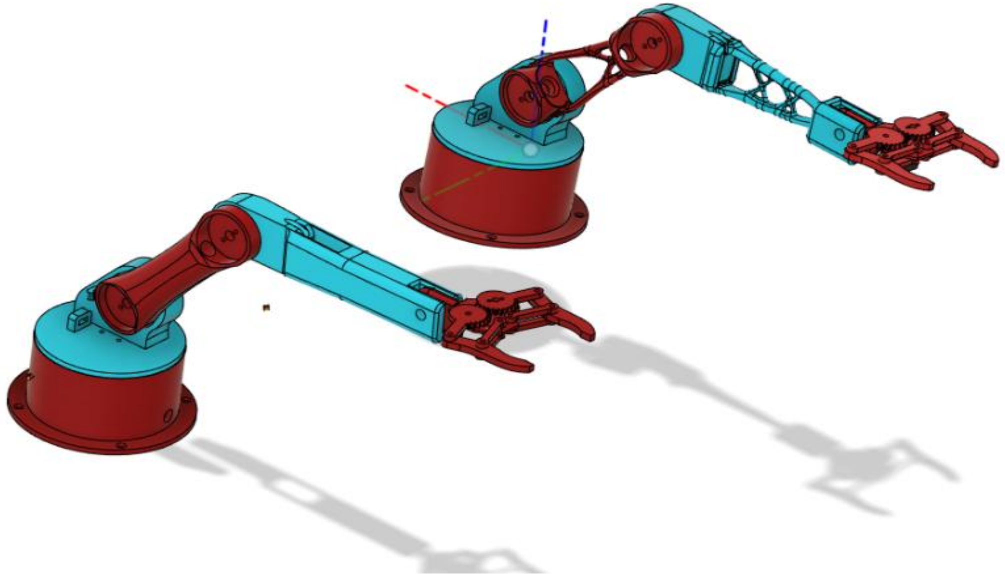
$$V_{Battery} = 7.4V$$

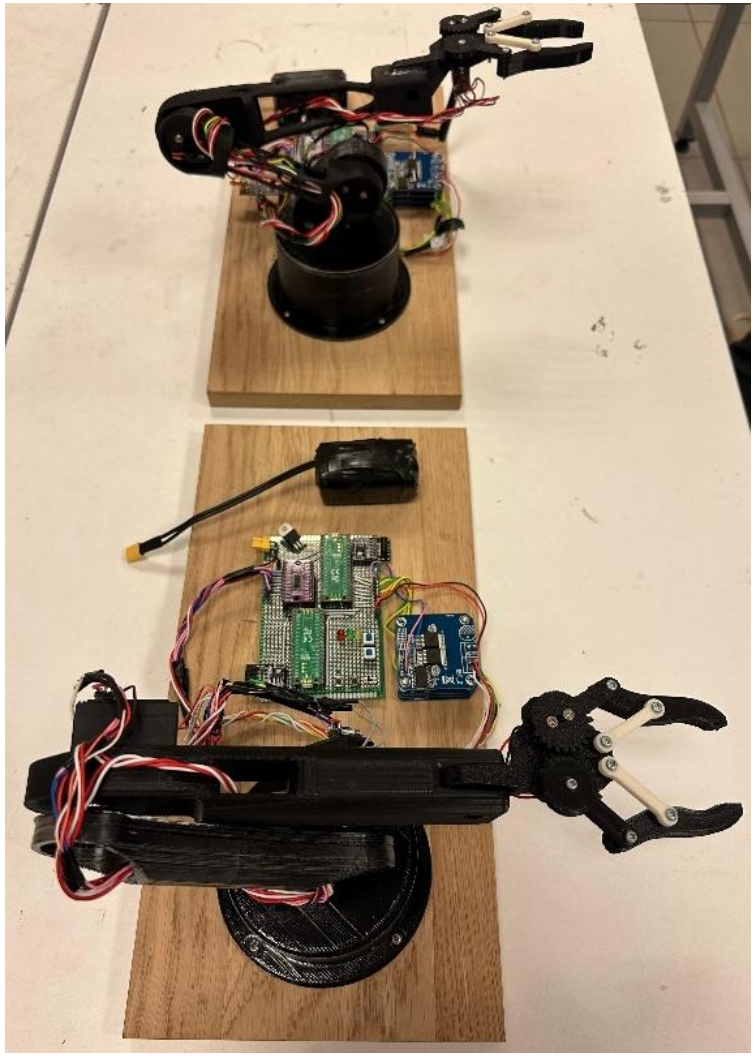
$$I_{Battery} = 6400mAh$$

$$P_{Battery} = I_{Battery} * V_{Battery}$$

$$P_{Battery} = 47.36W$$

Mechanical Design





Bilgi

Ad	PLA
Açıklama	Polylactic Acid
Anahtar Sözcükler	PLA
Tür	Plastik
Altsınıf	Termoplastik
Kaynak	Autodesk
Kaynak URL	

Temel Termal

Isıl İletkenlik	1,300E-01 W/(m·K)
Özgül Isı	1,800 J/(g·°C)
Isıl Genişleme Katsayısı	85,698 µm/(m·°C)

Nylon 12 (with Formlabs Fuse 1 3D Printer)

Density	1.015E-06 kg / mm^3
Young's Modulus	1850.00 MPa
Poisson's Ratio	0.35
Yield Strength	46.00 MPa
Ultimate Tensile Strength	50.00 MPa
Thermal Conductivity	3.500E-04 W / (mm C)
Thermal Expansion Coefficient	1.275E-04 / C
Specific Heat	1830.00 J / (kg C)

Figure 4.38 Nylon 12 Properties

Stress Analysis

⊟ Constraints

⊟ Fixed1

Type	Fixed
Ux	Fixed
Uy	Fixed
Uz	Fixed

⊟ Selected Entities

⊟ Loads

⊟ Gravity

Type	Gravity
Magnitude	9.807 m / s^2
X Value	0.00 m / s^2
Y Value	0.00 m / s^2
Z Value	-9.807 m / s^2

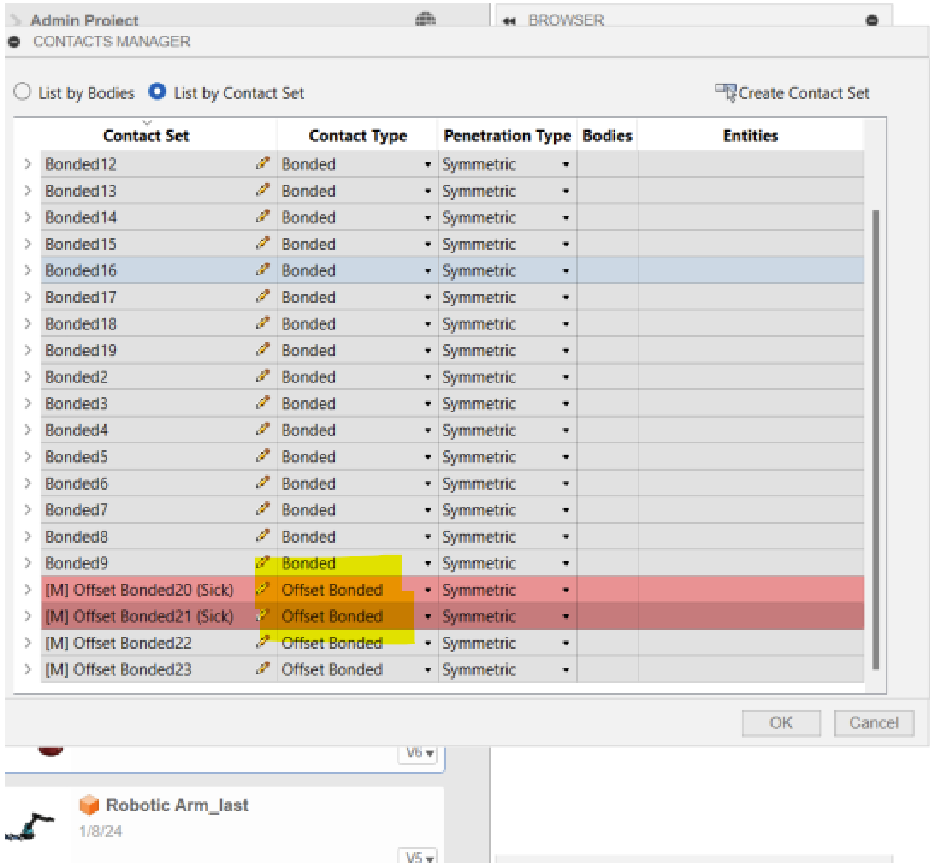
⊟ Selected Entities

Stress Analysis

⊟ **Force1**

Type	Force
Magnitude	3.00 N
X Value	0.00 N
Y Value	0.00 N
Z Value	-3.00 N
Force Per Entity	No

⊟ **Selected Entities**



Contact Set	Contact Type	Penetration Type	Bodies	Entities
Bonded12	Bonded	Symmetric		
Bonded13	Bonded	Symmetric		
Bonded14	Bonded	Symmetric		
Bonded15	Bonded	Symmetric		
Bonded16	Bonded	Symmetric		
Bonded17	Bonded	Symmetric		
Bonded18	Bonded	Symmetric		
Bonded19	Bonded	Symmetric		
Bonded2	Bonded	Symmetric		
Bonded3	Bonded	Symmetric		
Bonded4	Bonded	Symmetric		
Bonded5	Bonded	Symmetric		
Bonded6	Bonded	Symmetric		
Bonded7	Bonded	Symmetric		
Bonded8	Bonded	Symmetric		
Bonded9	Bonded	Symmetric		
[M] Offset Bonded20 (Sick)	Offset Bonded	Symmetric		
[M] Offset Bonded21 (Sick)	Offset Bonded	Symmetric		
[M] Offset Bonded22	Offset Bonded	Symmetric		
[M] Offset Bonded23	Offset Bonded	Symmetric		

Stress Analysis

⊟ **Mesh**

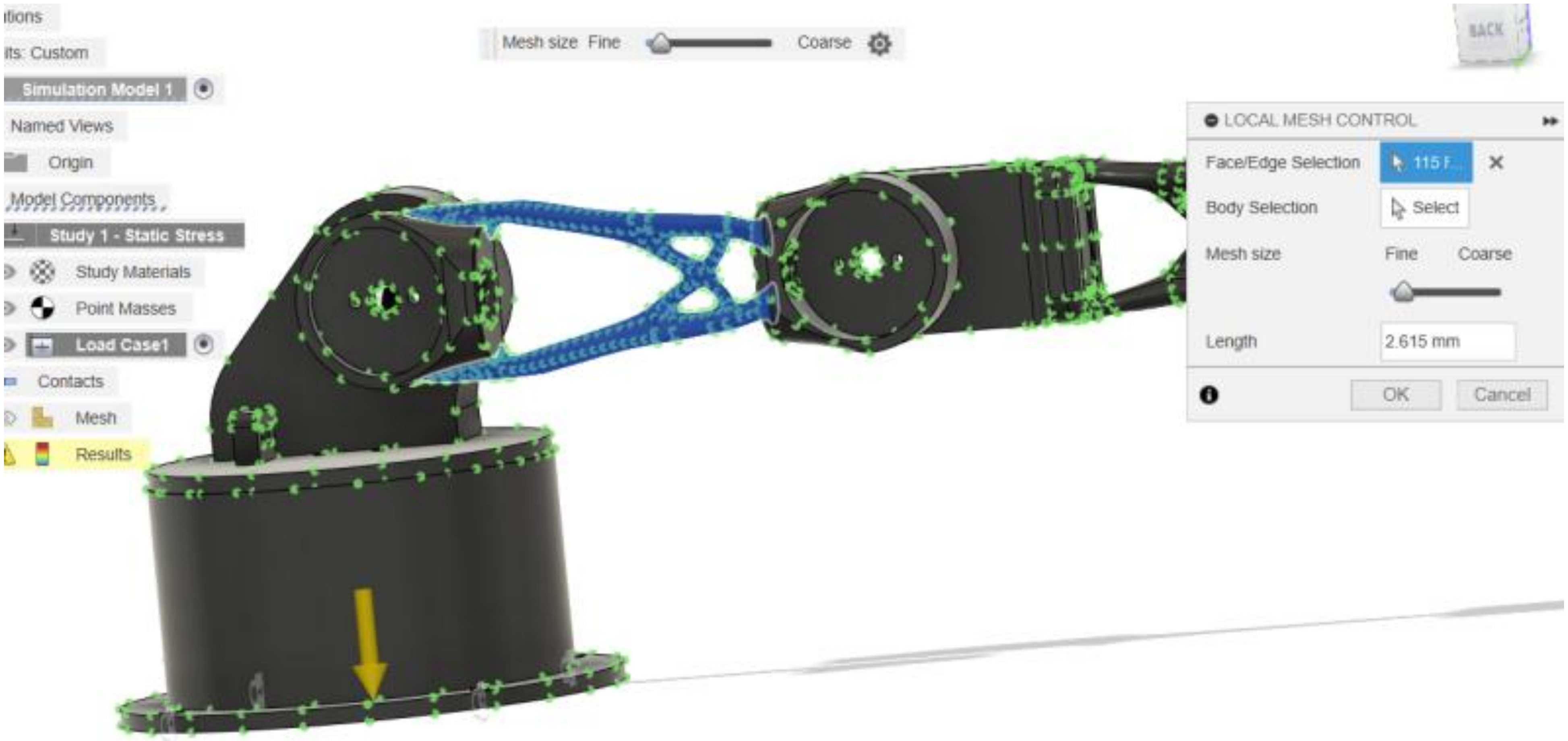
Average Element Size (% of model size)	
Solids	10
Scale Mesh Size Per Part	No
Average Element Size (absolute value)	-
Element Order	Parabolic
Create Curved Mesh Elements	Yes
Max. Turn Angle on Curves (Deg.)	60
Max. Adjacent Mesh Size Ratio	1.5
Max. Aspect Ratio	10
Minimum Element Size (% of average size)	20

⊟ **Adaptive Mesh Refinement**

Number of Refinement Steps	0
Results Convergence Tolerance (%)	20
Portion of Elements to Refine (%)	10
Results for Baseline Accuracy	von Mises Stress

⊟ **Mesh**

Type	Nodes	Elements
Solids	118921	68945



Stress Analysis

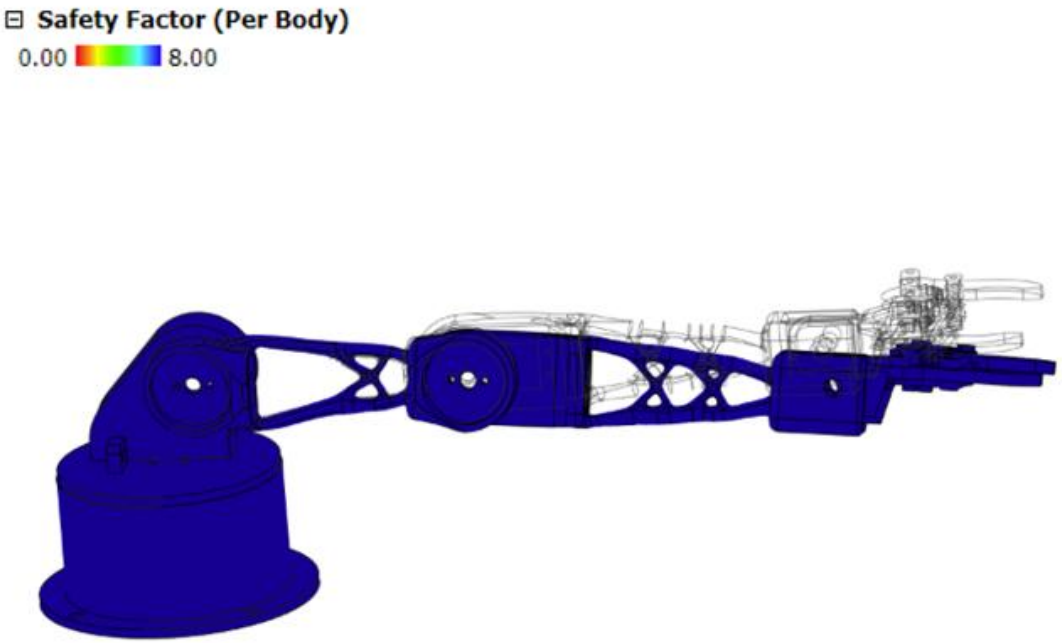


Figure 4.31 Safety Factor

Figure 4.32 Stress (von Mises)

Stress Analysis

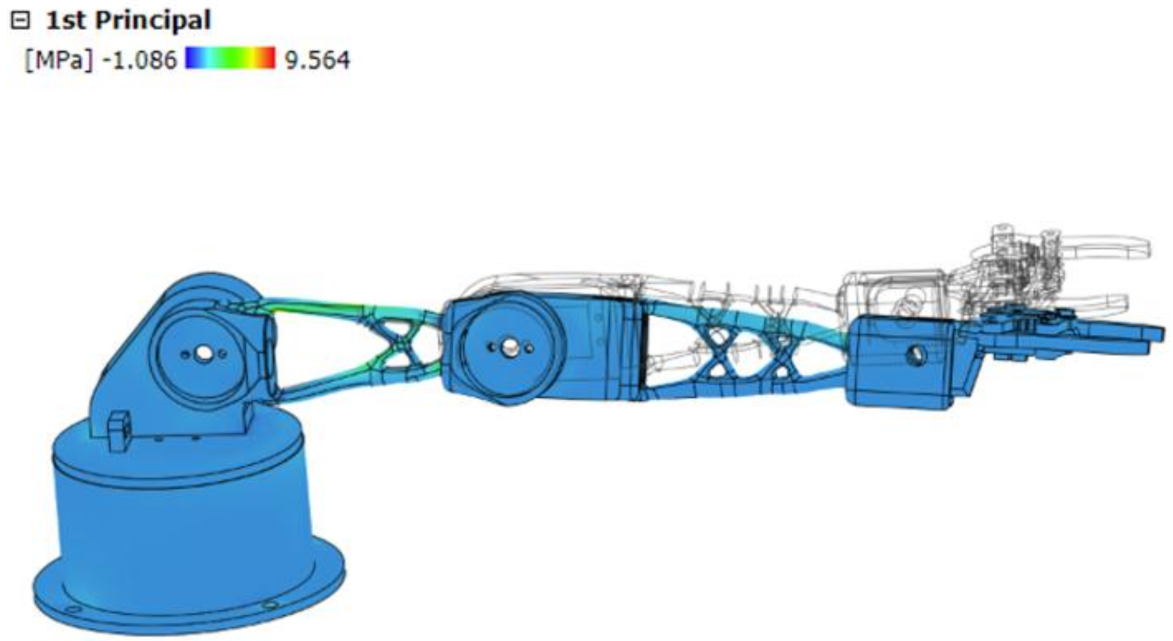


Figure 4.33 1st Principal

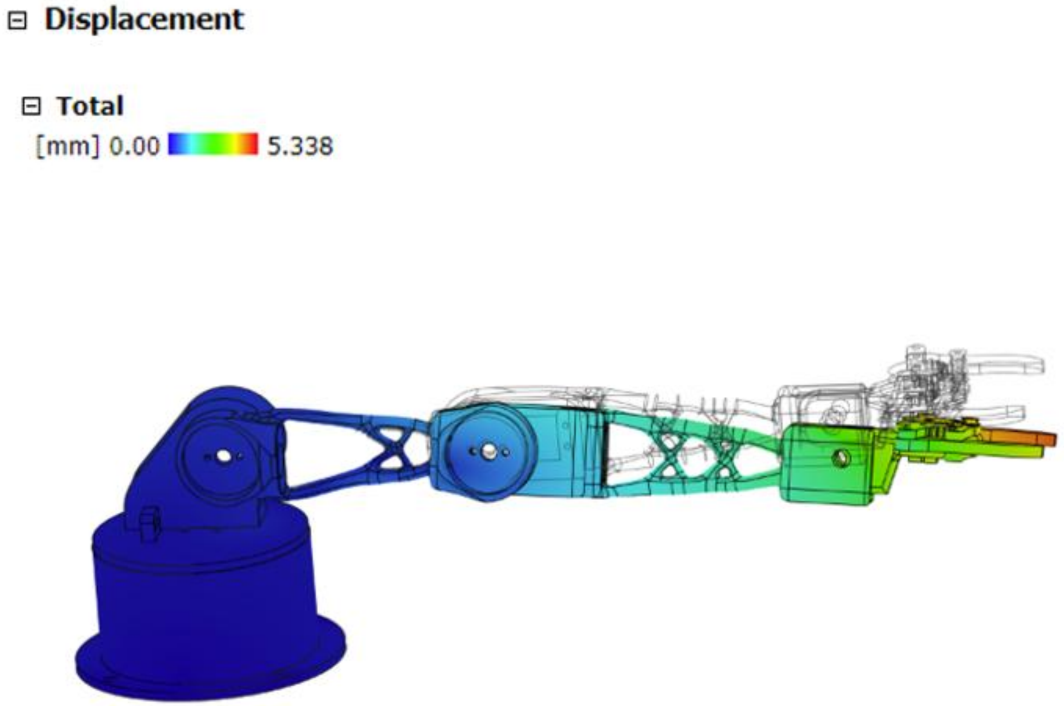


Figure 4.35 Displacement

Stress Analysis Results

Result Summary

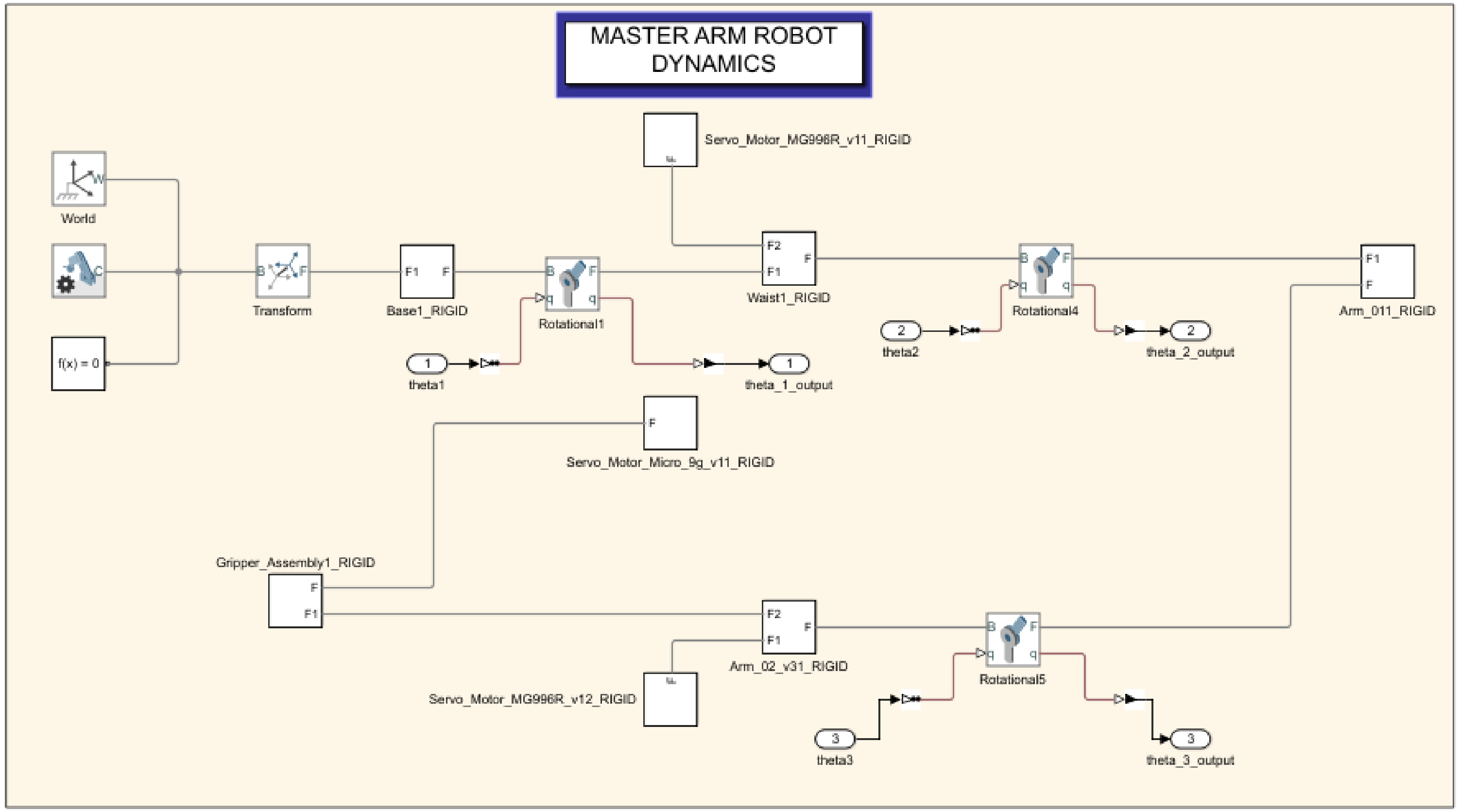
Name	Minimum	Maximum
Safety Factor		
Safety Factor (Per Body)	15.00	15.00
Stress		
von Mises	2.384E-06 MPa	3.048 MPa
1st Principal	-1.298 MPa	2.941 MPa
3rd Principal	-4.663 MPa	0.639 MPa
Normal XX	-3.421 MPa	1.59 MPa
Normal YY	-2.541 MPa	1.99 MPa
Normal ZZ	-2.087 MPa	1.253 MPa
Shear XY	-0.825 MPa	0.559 MPa
Shear YZ	-0.478 MPa	0.994 MPa
Shear ZX	-0.465 MPa	1.293 MPa
Displacement		
Total	0.00 mm	1.225 mm
X	-0.026 mm	0.078 mm
Y	-0.002 mm	0.193 mm
Z	-1.216 mm	0.031 mm
Reaction Force		
Total	0.00 N	1.984 N
X	-1.533 N	1.477 N
Y	-1.449 N	1.31 N
Z	-0.97 N	1.365 N
Strain		
Equivalent	0.00	0.003
1st Principal	0.00	0.002
3rd Principal	-0.003	0.00
Normal XX	-0.001	7.340E-04
Normal YY	-8.495E-04	6.566E-04
Normal ZZ	-4.175E-04	3.586E-04
Shear XY	-0.001	8.162E-04
Shear YZ	-6.976E-04	0.001
Shear ZX	-6.782E-04	0.002
Contact Pressure		
Total	0.00 MPa	0.687 MPa
X	-0.30 MPa	0.223 MPa
Y	-0.302 MPa	0.226 MPa
Z	-0.418 MPa	0.648 MPa
Contact Force		
Total	0.00 N	7.693 N
X	-4.96 N	3.217 N
Y	-3.276 N	3.586 N
Z	-7.659 N	6.19 N

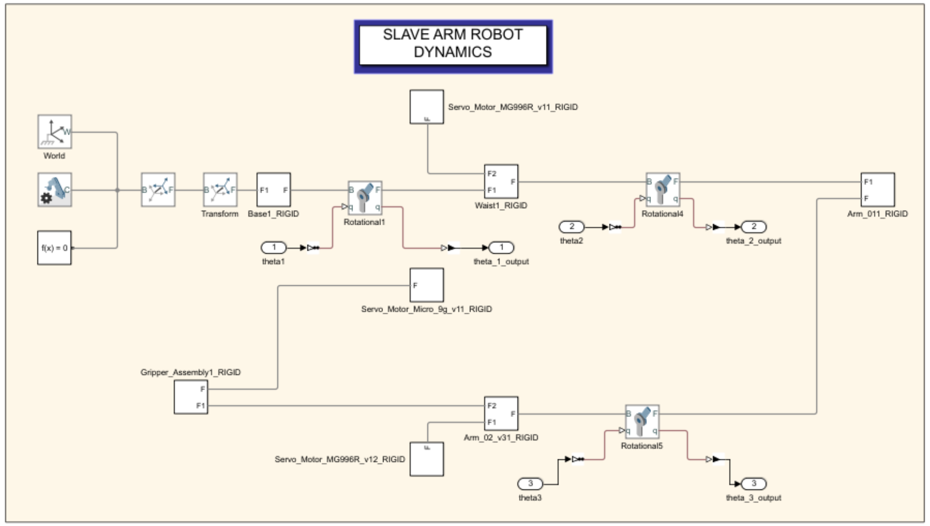
Result Summary

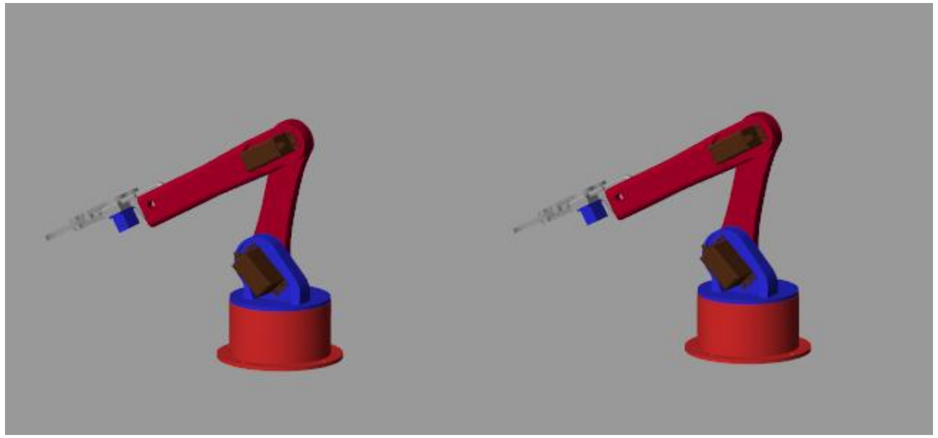
Name	Minimum	Maximum
Safety Factor		
Safety Factor (Per Body)	5.497	15.00
Stress		
von Mises	4.104E-05 MPa	8.368 MPa
1st Principal	-1.086 MPa	9.564 MPa
3rd Principal	-9.439 MPa	0.97 MPa
Normal XX	-9.069 MPa	9.252 MPa
Normal YY	-2.345 MPa	2.147 MPa
Normal ZZ	-2.083 MPa	2.295 MPa
Shear XY	-2.122 MPa	1.86 MPa
Shear YZ	-1.018 MPa	0.852 MPa
Shear ZX	-2.516 MPa	1.733 MPa
Displacement		
Total	0.00 mm	5.338 mm
X	-0.545 mm	0.331 mm
Y	-2.383 mm	0.319 mm
Z	-4.803 mm	0.37 mm
Reaction Force		
Total	0.00 N	1.978 N
X	-1.403 N	1.373 N
Y	-1.15 N	1.365 N
Z	-0.679 N	1.224 N
Strain		
Equivalent	0.00	0.005
1st Principal	-7.090E-06	0.005
3rd Principal	-0.005	2.045E-05
Normal XX	-0.004	0.004
Normal YY	-0.001	0.001
Normal ZZ	-0.002	0.002
Shear XY	-0.003	0.003
Shear YZ	-0.001	0.001
Shear ZX	-0.004	0.003
Contact Pressure		
Total	0.00 MPa	1.182 MPa
X	-1.022 MPa	1.011 MPa
Y	-0.889 MPa	0.761 MPa
Z	-0.989 MPa	0.746 MPa
Contact Force		
Total	0.00 N	6.723 N
X	-3.52 N	3.685 N
Y	-2.461 N	2.23 N
Z	-4.369 N	6.403 N

Figure 4.36 Normal & Generative Design Comparison

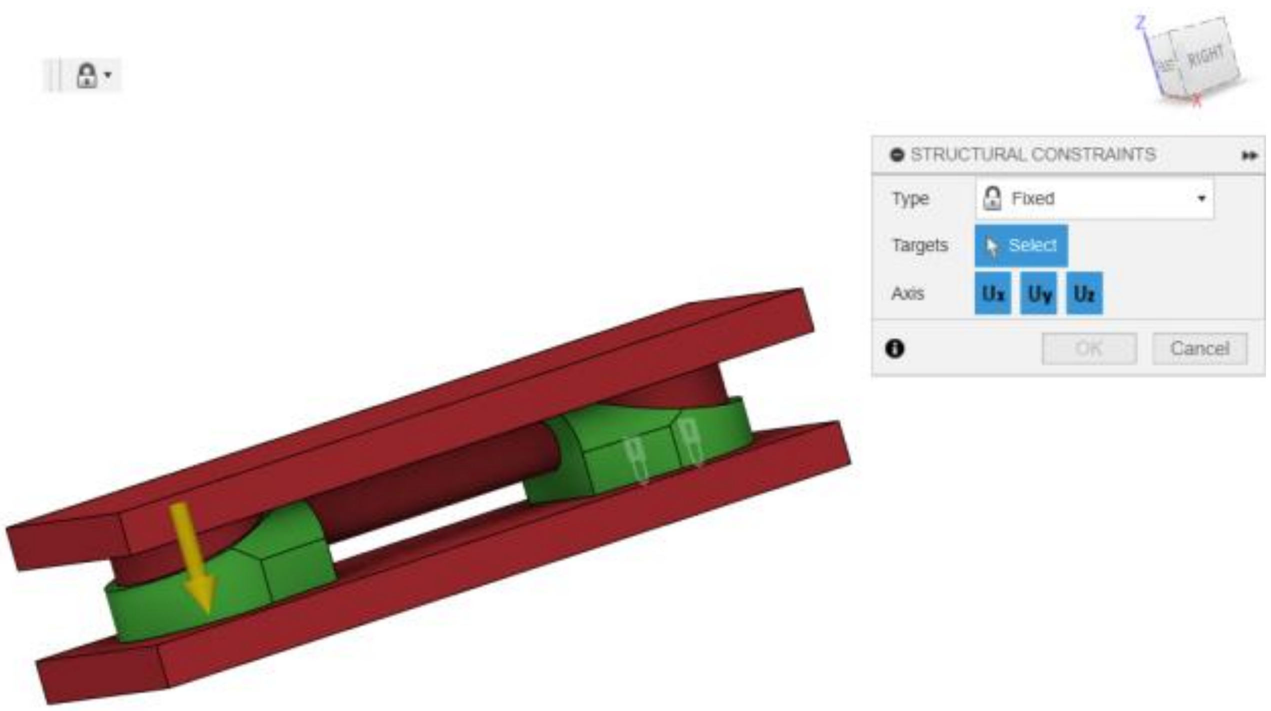
MULTIBODY ROBOT DYNAMICS & SIMSCAPE SIMULATION OUTPUT

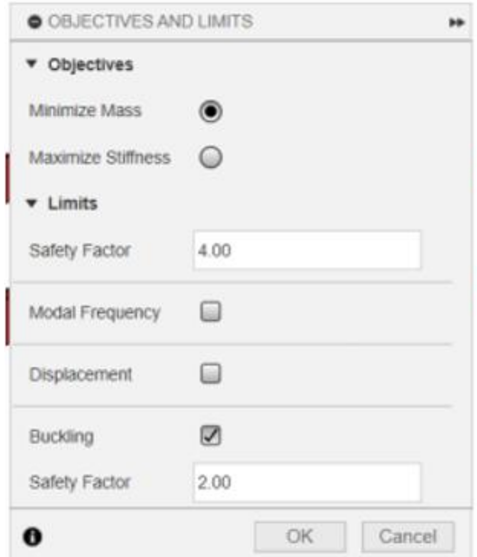


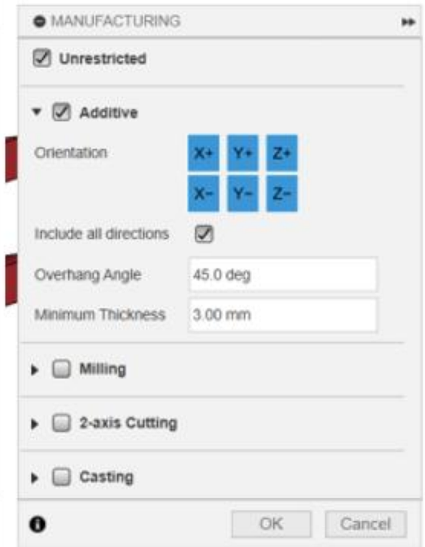


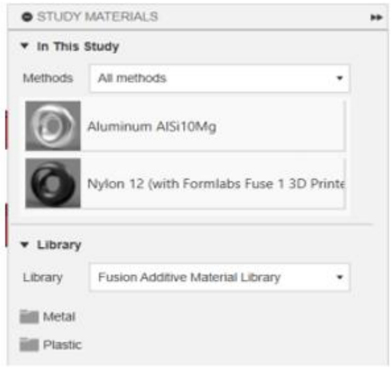


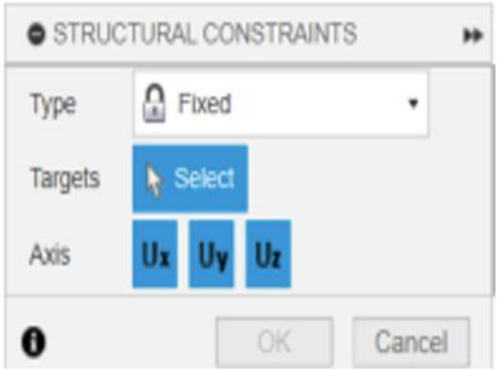
CONCEPTUAL DESIGN & UNIQUENESS











CONCEPTUAL DESIGN & UNIQUENESS

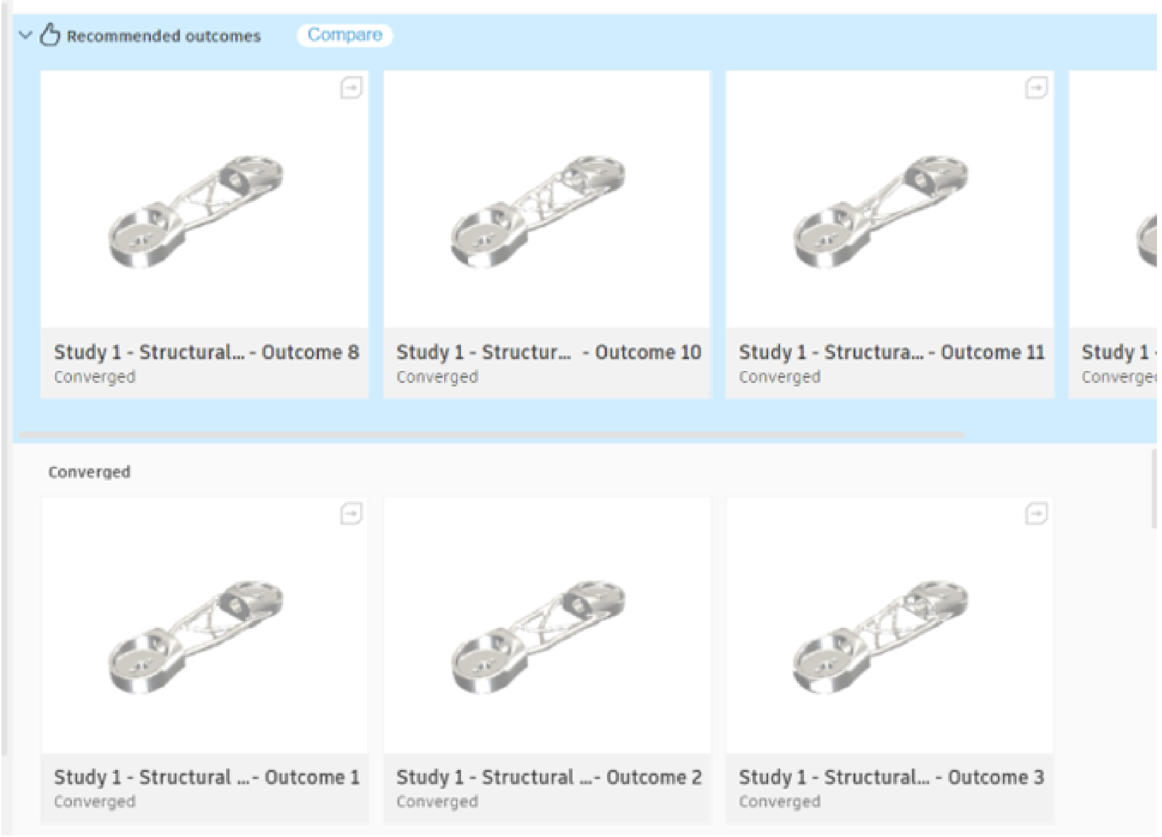
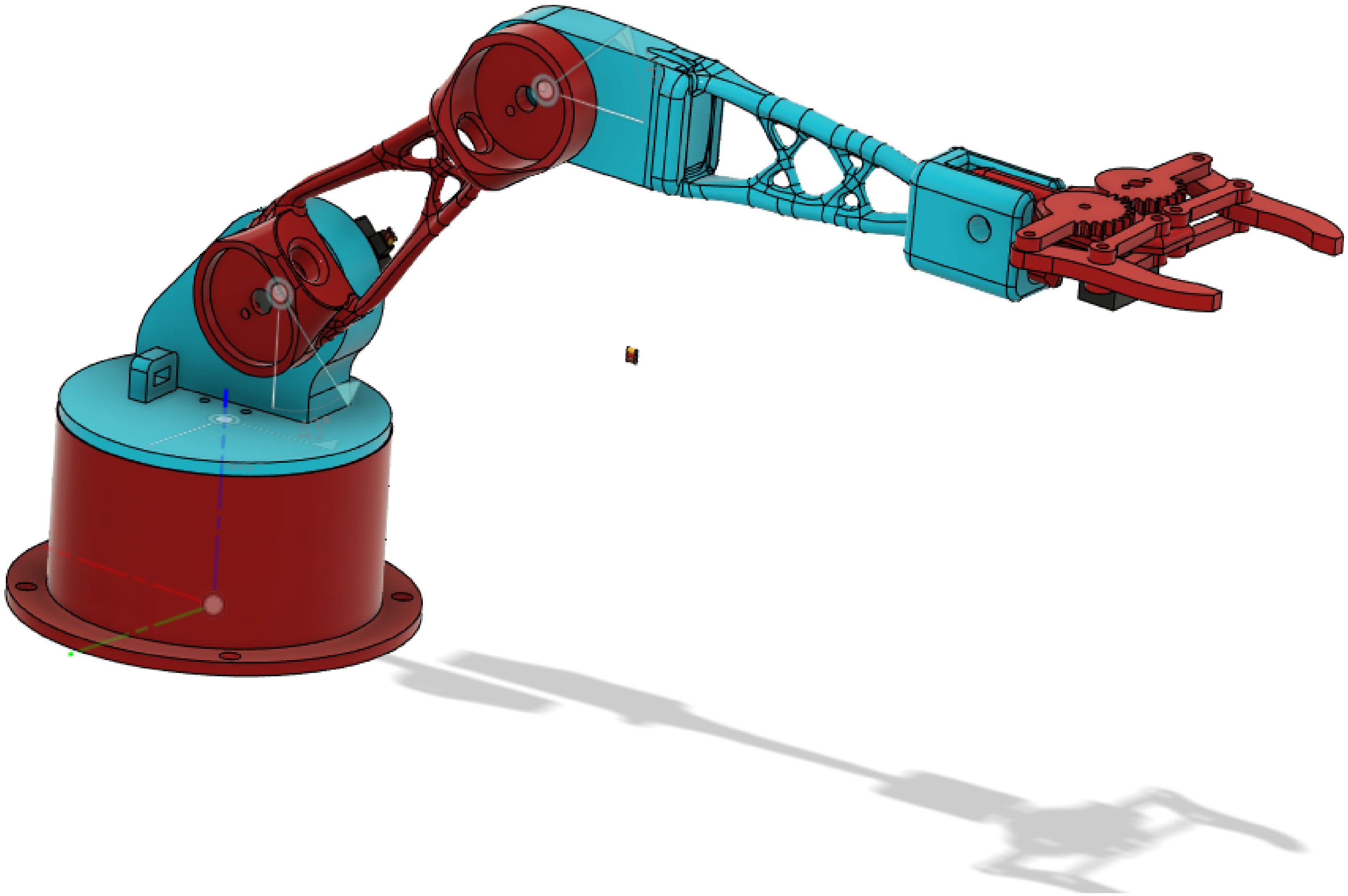


Figure 4.45 Conceptual Designs

Figure 4.46 Second Link Generative Design Output

UNIQUE DESIGN WITH GENERATIVE DESIGN



Electronical Design Realization

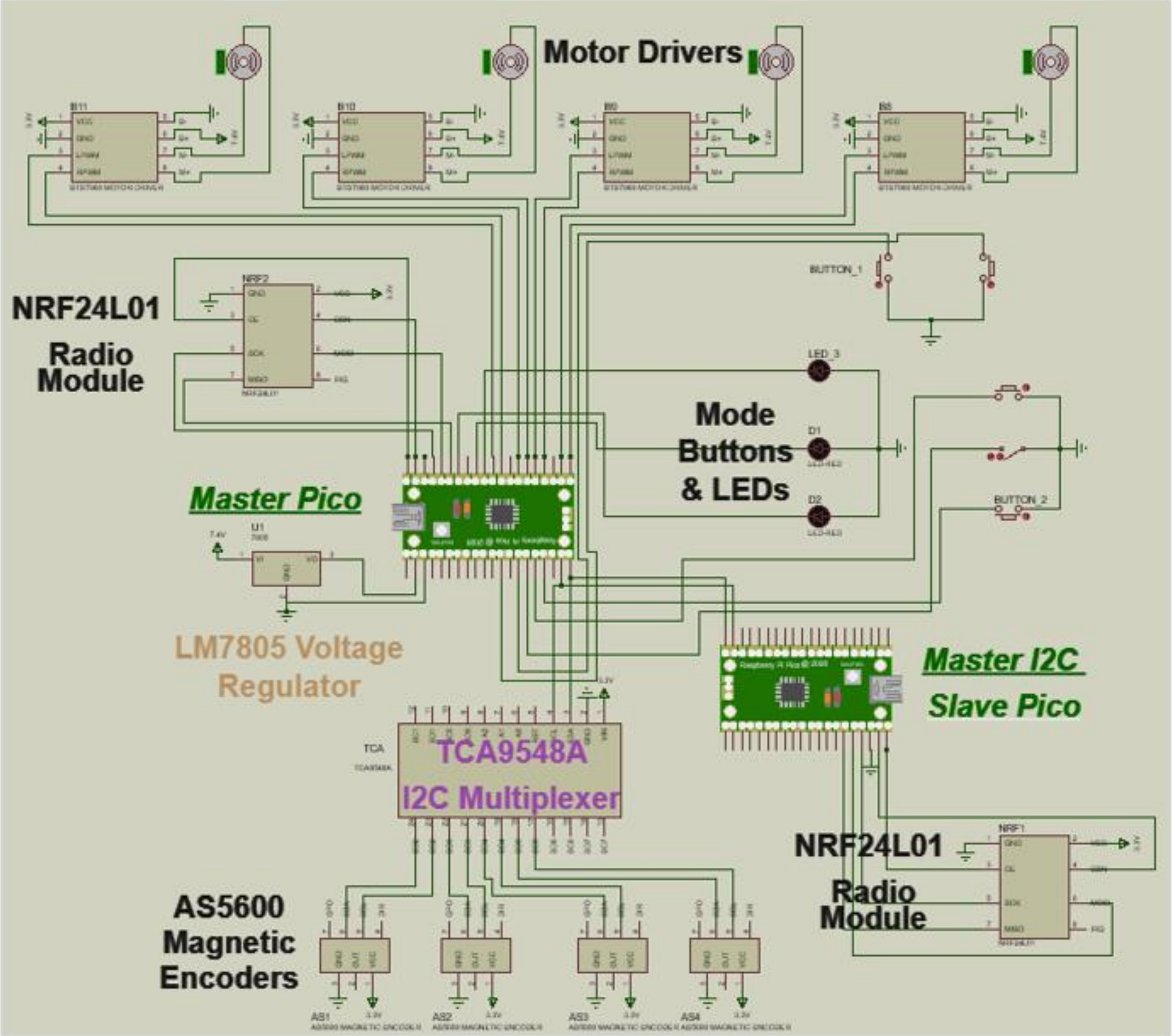


Figure 4.61 – Controller Arm (Master Arm) Circuit Design on Proteus

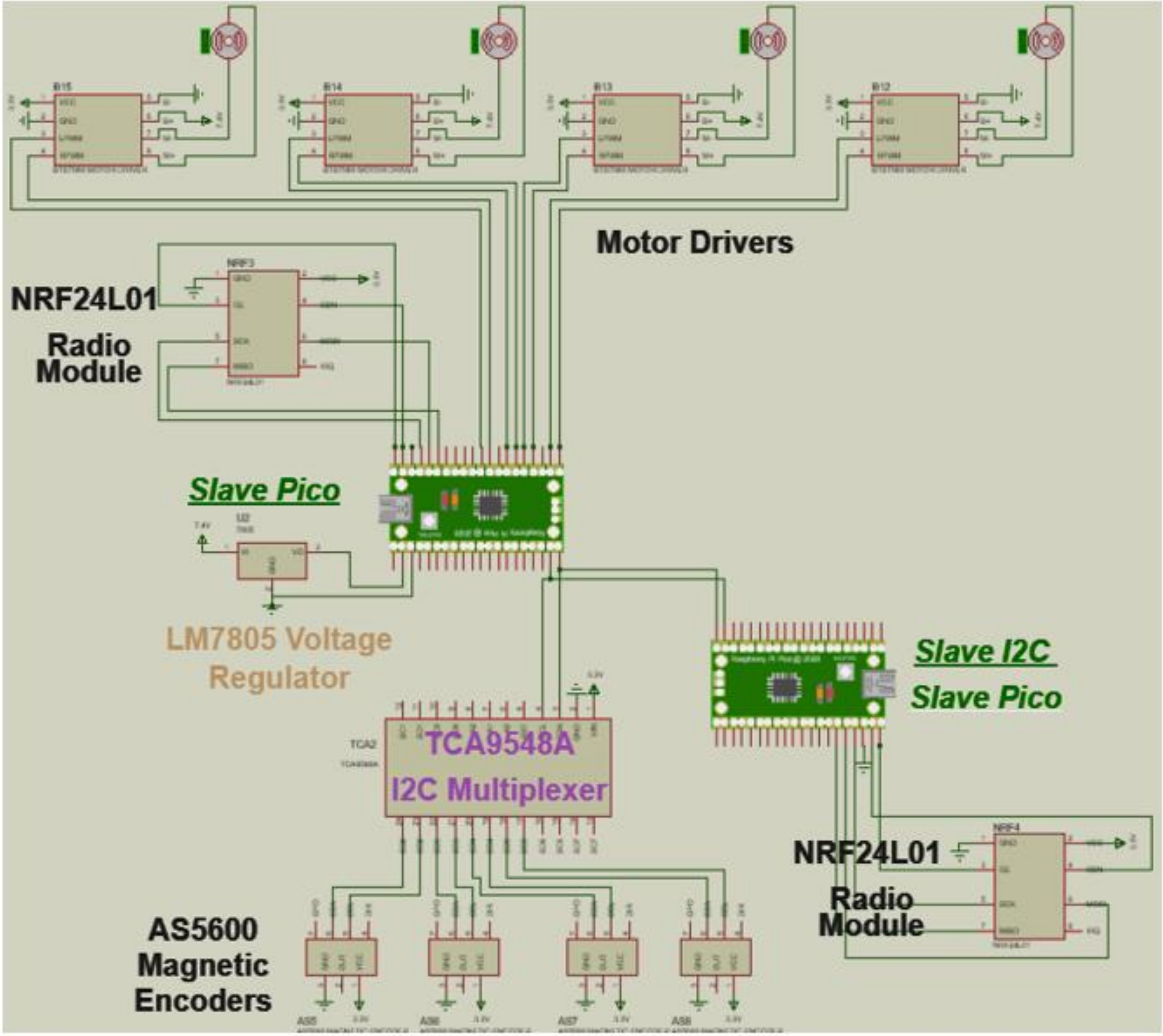


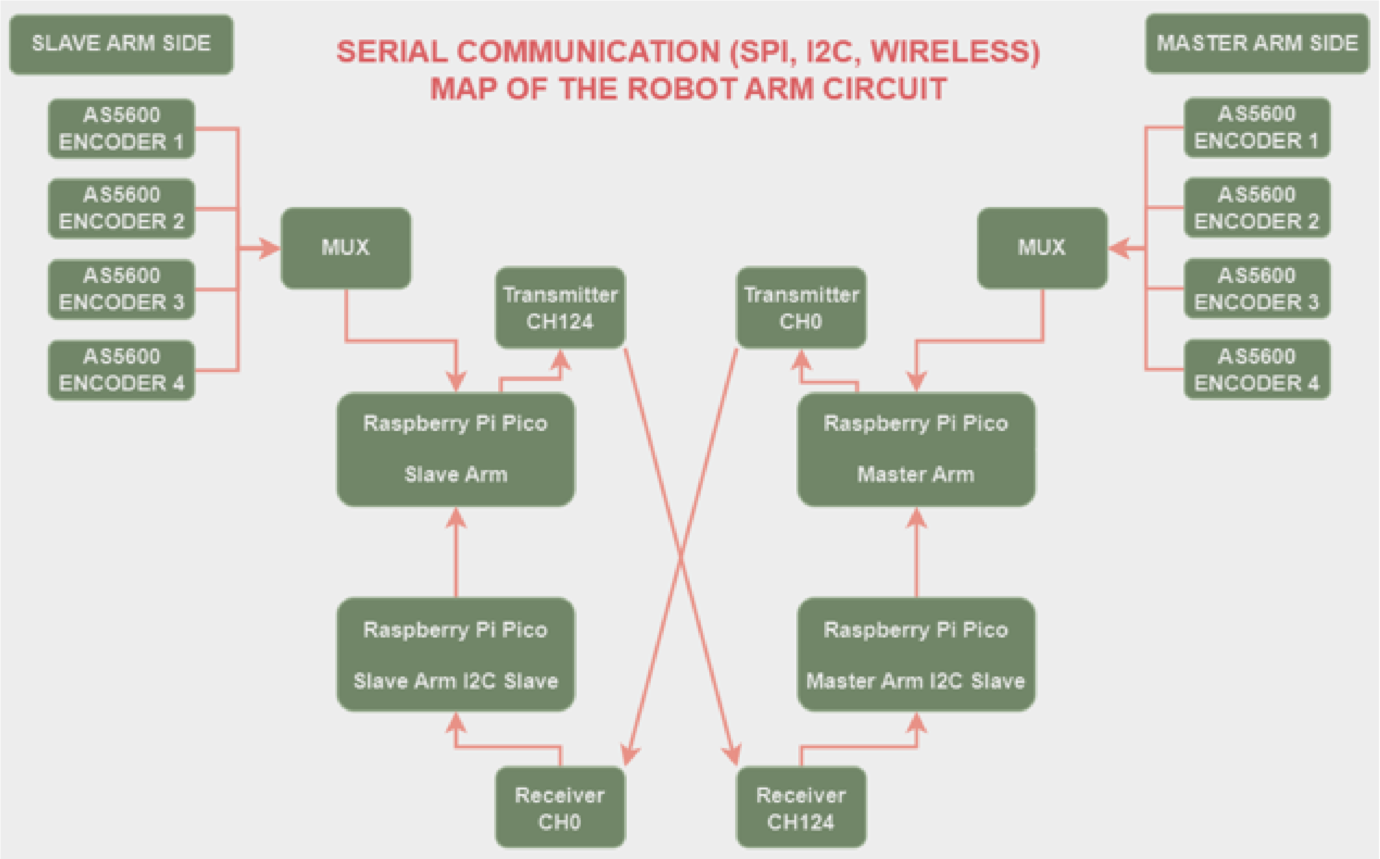
Figure 4.62 - Controlled Arm (Slave Arm) Circuit Design on Proteus

Electronical Design Realization

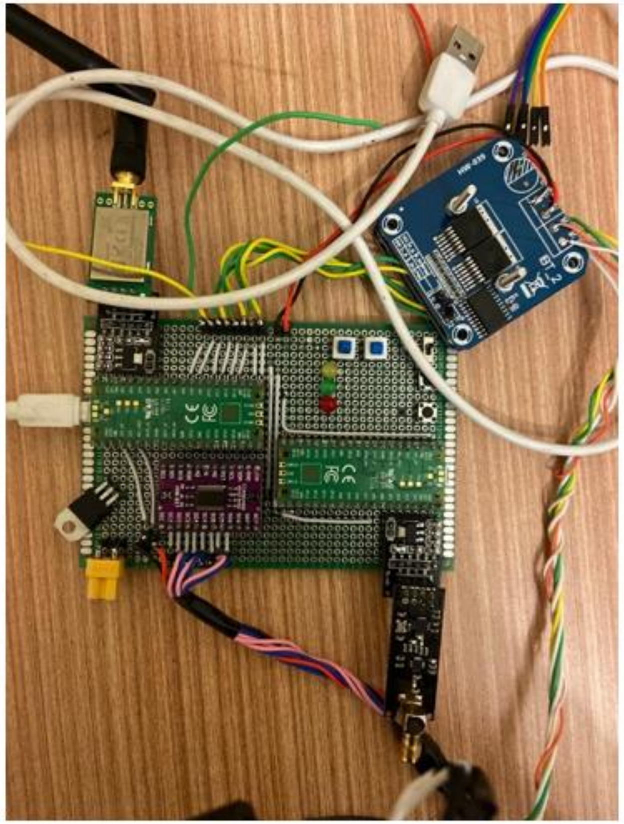
Table 4-1 Component List

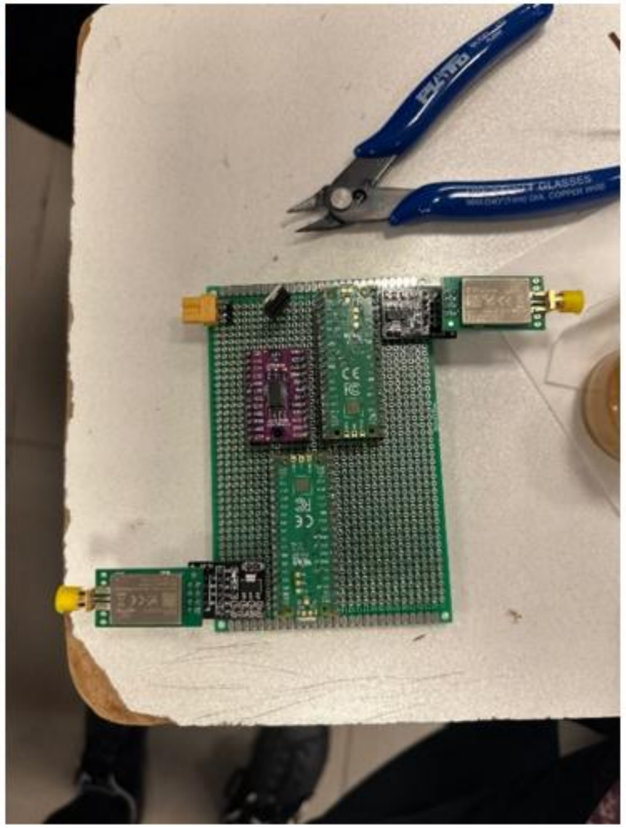
COMPONENT	DESCRIPTION	BASE QTY.
Battery	Orion 18650 3.7V 3200mah 3c Rechargeable Li-ion	8
Servo Motor	6 pcs. MG995R 13kg & 2 pcs. SG90 9G Mini	8
Magnetic Encoder	AS5600 (12-bit resolution)	8
Motor Driver	BTS7960 40A	8
Voltage Regulator	LM7805 3.3V	2
Microcontroller	2 pcs. Raspberry Pi Pico & 2 pcs. Raspberry Pico W	4
Wireless Communication Module	NRF24L01+PA+LNA SMA Antenna 2.4GHz 5km	4
Multiplexer	I2C TCA9548A	2
Mechanical Parts	Will be built-in 3D Printer (PLA Filament)(QTY in kg)	2
LED & Button	3 pcs. LED & 6 pcs. button	9
Wiring	For the connections (QTY in m)	10

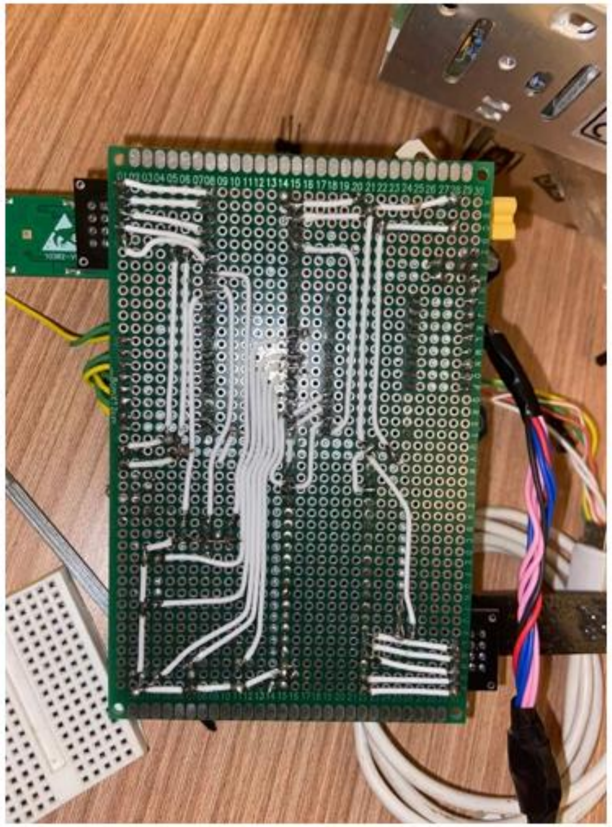
4.2.2 Serial Communication (SPI, I2C, WIRELESS) Map of the Robot Arm Circuit



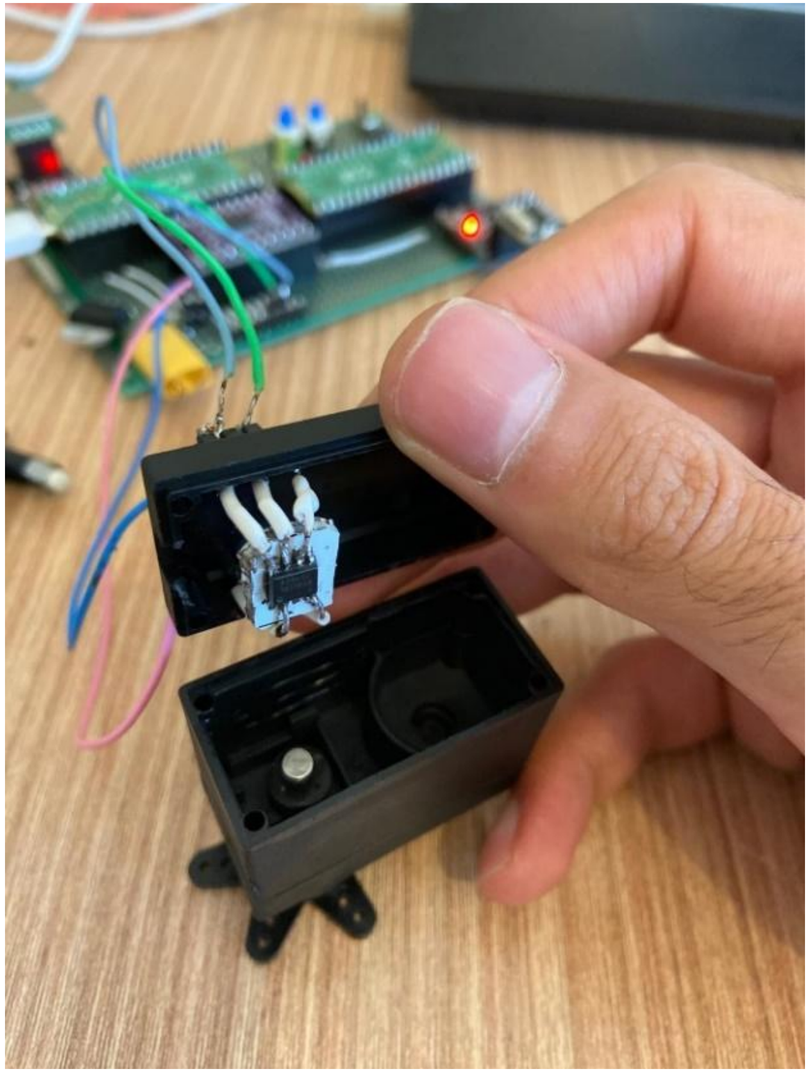
Electronical Design Realization

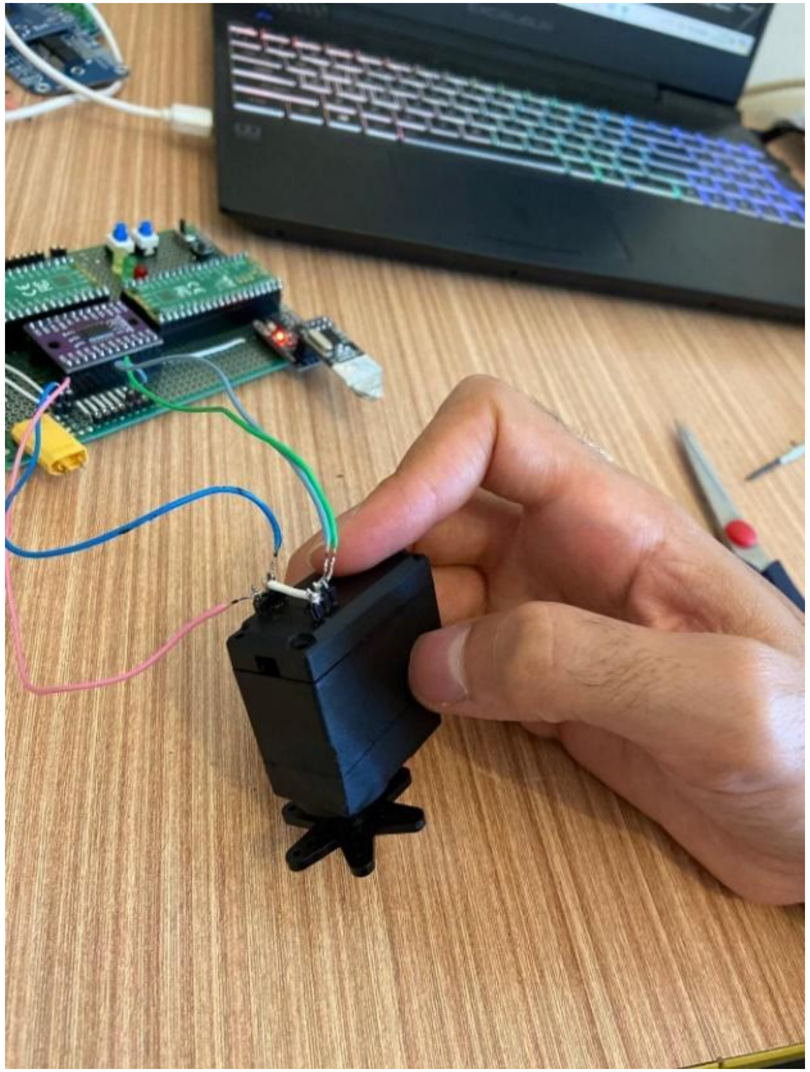




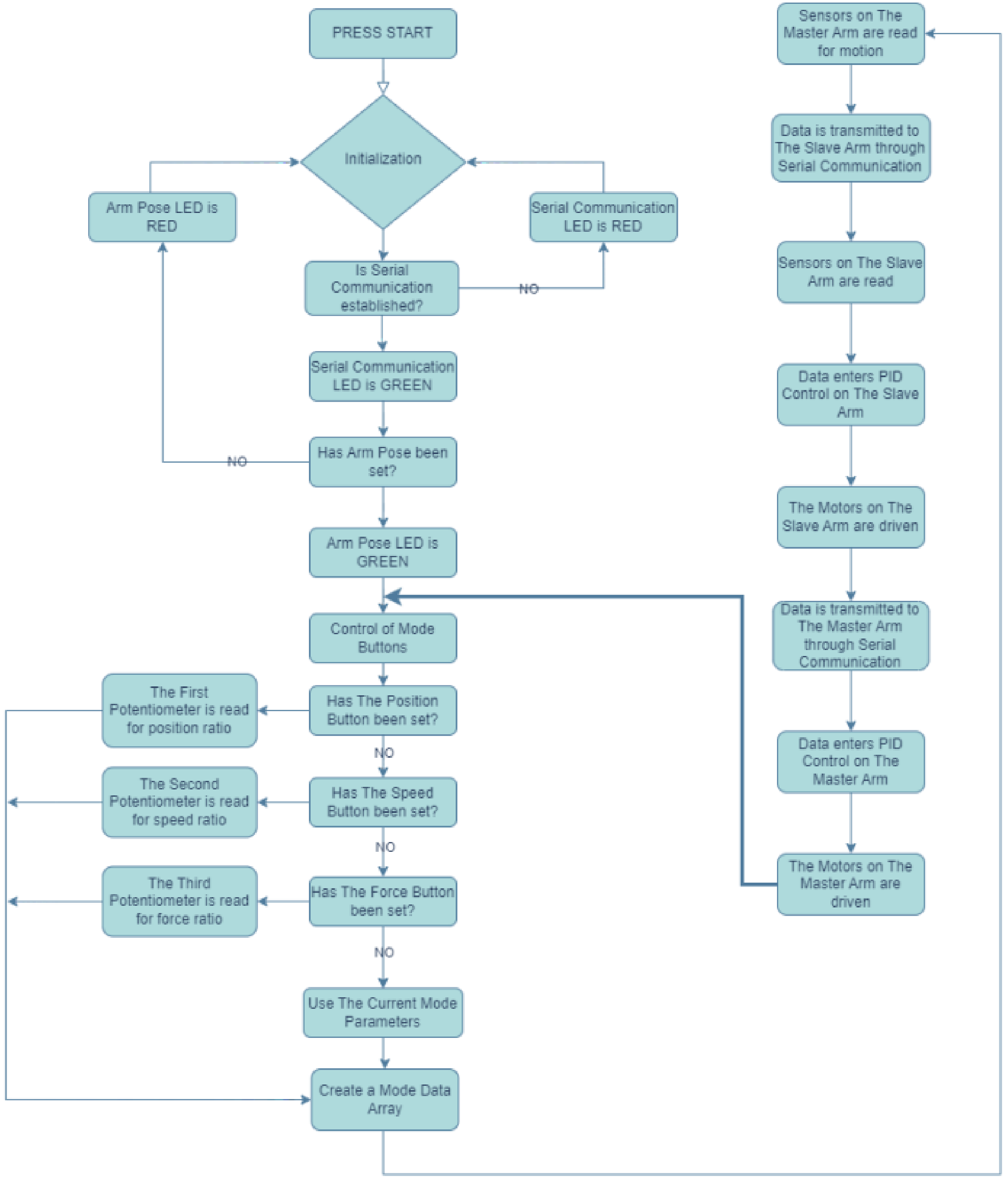


Electronical Design Realization





SOFTWARE & ALGORYTHM



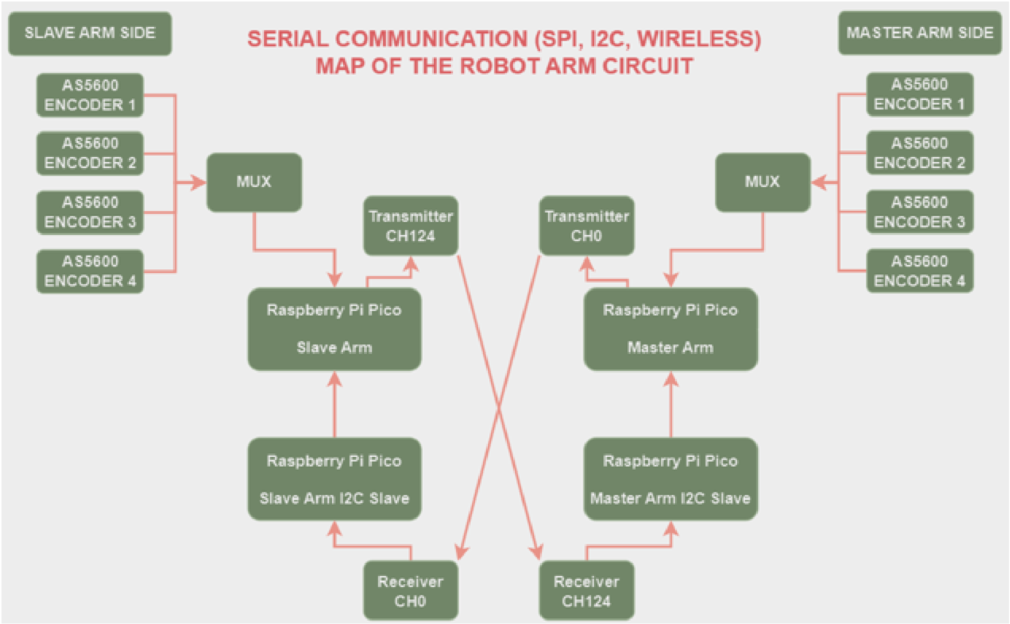
SOFTWARE & ALGORYTHM

- In the electronic design, four separate code bases were created for the four microcontrollers: Master Arm, Master Arm I2C Slave, Slave Arm, and Slave Arm I2C Slave.

- Each code handles communication and control for its respective microcontroller. For example, the Master Pico code communicates with the Master Arm I2C Slave via I2C and receives data from the NRF24L01, which sets the master arm's setpoint based on the slave arm's encoder data.

- Feedback from the master arm's encoders is transmitted to the Slave Pico as setpoints. These setpoints and feedback data are used in PID control to manage the motors.

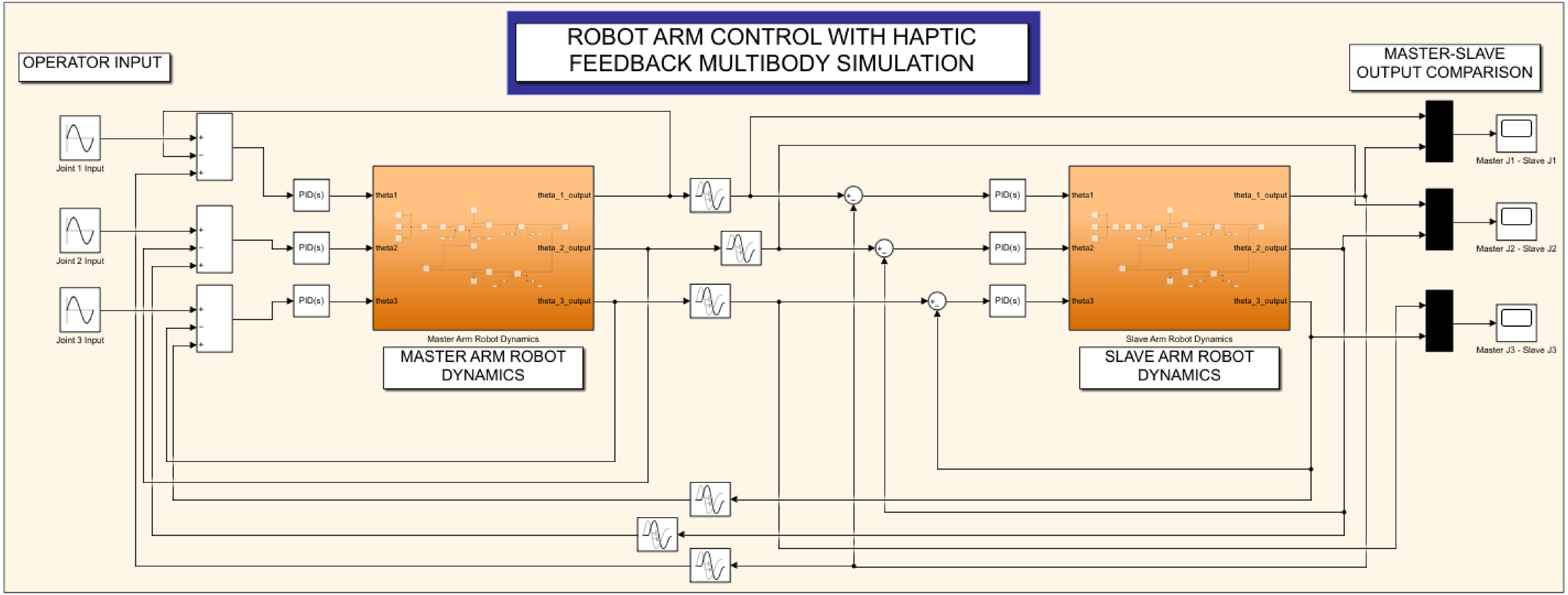
- The code includes necessary libraries, force transmission coefficients, functions, and defines angular motion limits for each joint. Comments and optimizations ensure clarity and efficient operation. The same setup applies to the Slave Arm.



To view the codes, please refer to the code section in the Appendix;

- *Master Arm Code*
- *Master Arm I2C Slave Code*
- *Slave Arm Code*
- *Slave Arm I2C Slave Code*

CONTROL


ROBOT ARM CONTROL WITH HAPTIC FEEDBACK MULTIBODY SIMULATION
OPERATOR INPUT
MASTER-SLAVE OUTPUT COMPARISON
Joint 1 Input
Joint 2 Input
Joint 3 Input
PID(s)
theta1
theta2
theta3
theta_1_output
theta_2_output
theta_3_output
Master Arm Robot Dynamics
MASTER ARM ROBOT DYNAMICS
Slave Arm Robot Dynamics
SLAVE ARM ROBOT DYNAMICS
Master J1 - Slave J1
Master J2 - Slave J2
Master J3 - Slave J3

CONTROL

The PID coefficients were optimized through iterations, utilizing trial and error, to achieve the desired response from the system without inducing rapid oscillations. The system was designed to deliver an underdamped response to facilitate a quick reaction. As a result, the following PID coefficients were obtained.

```
int deriv=error-preverror;
preverror=error;
int pwm =8*error+0.1*integral+120*deriv;

if (error<=1 && error>=-1){integral=0;pwm=0; }
if (pwm>=1023){pwm=1023;}
if (pwm<=-1023){pwm=-1023;}
```

Figure: PID Coefficients Found by Optimization

CONTROL

Below is the output graph obtained using the PID coefficients as seen in the code. The System response is underdamped response as it's seen.

CONTROL

Below graph, we observe the output graph of the slave arm, which tracks the periodic motion of the master arm.

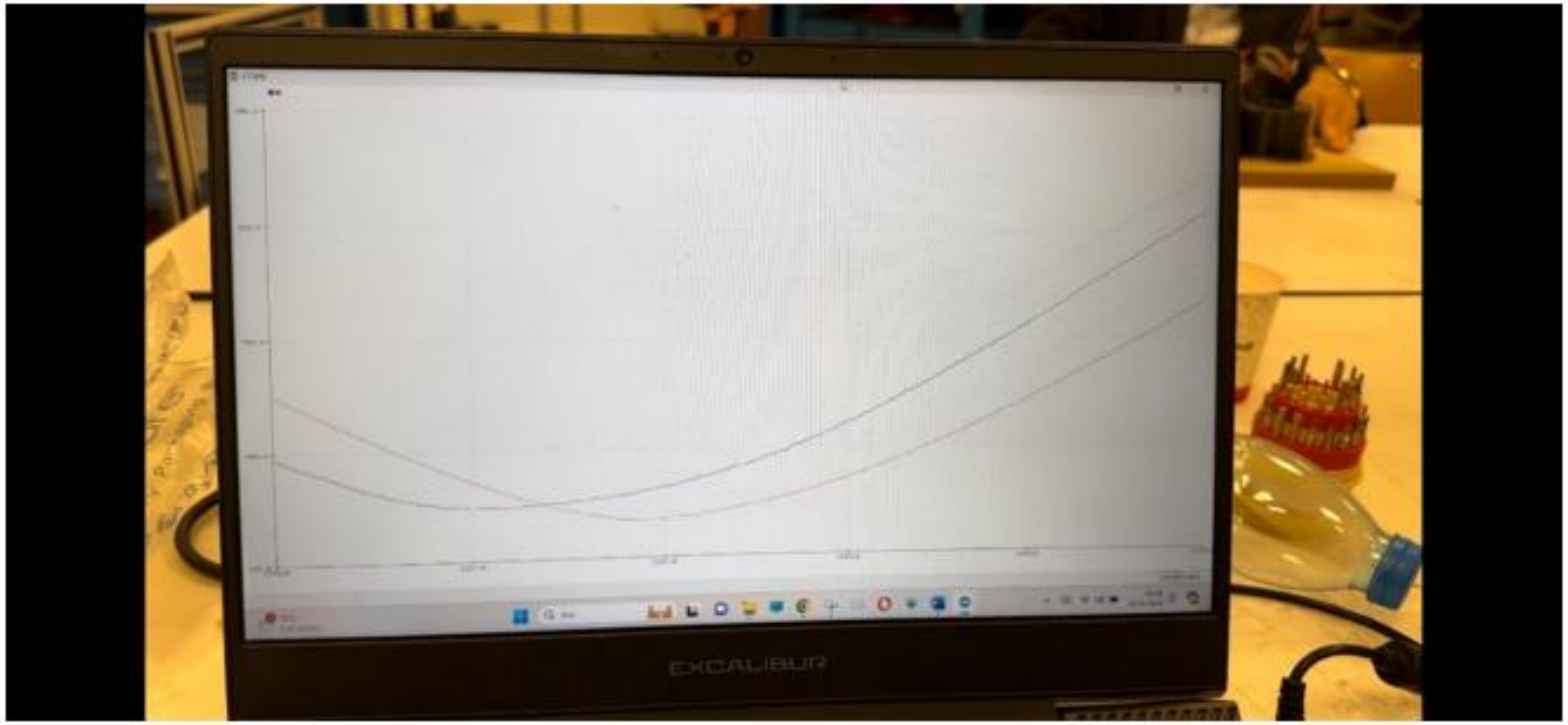
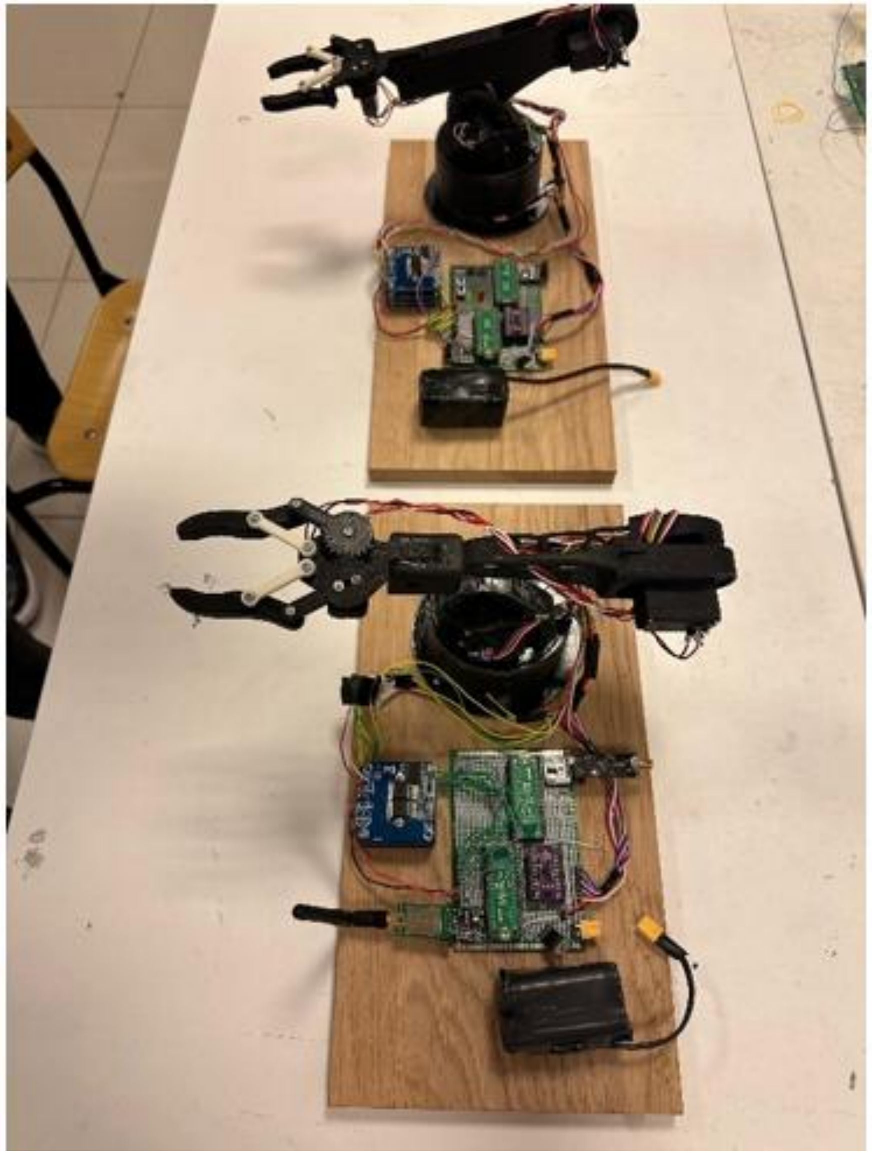
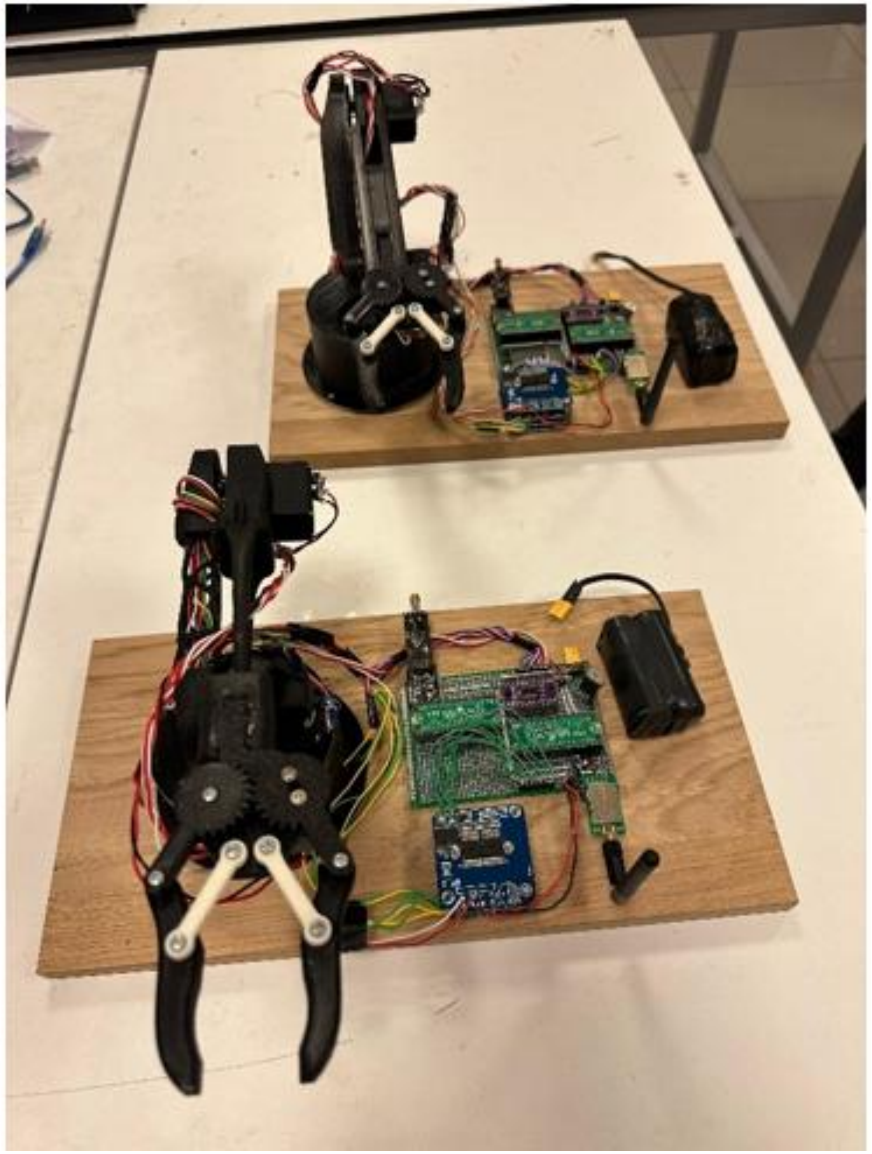


Figure 4.90 Sine Wave Response of the Slave Arm of One of The Joints

OVERALL SYSTEM VIEW

The images depict an overall system view encompassing the power, electronic, communication, and mechanical components of the robot arms. The robot arms are mounted on a wooden base.





Final

Table 5-1 - Gantt Chart of The Project

	October	November	December	January	February	March	April	May
Literature Review & Source Investigation								
Determination of Scope and Requirements								
Determination of System Topology								
MATLAB Simulations								
Theoretical Calculations								
3D Model Design								
Software Development								
Test Circuit & PID Trial								
3D Model Realization & Integration of Components								
Optimization & Finalization								

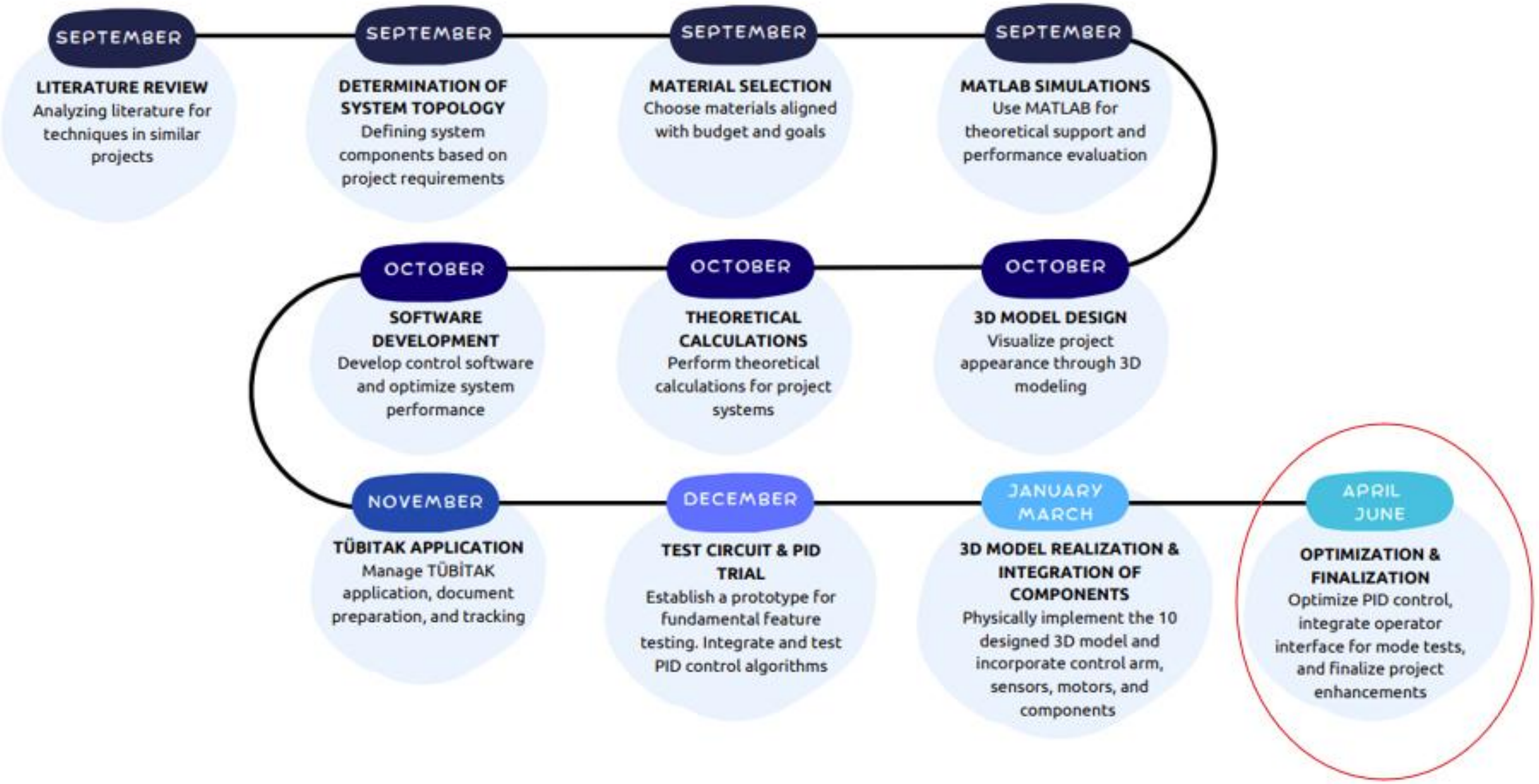


Figure 5.1 Road Map

YILDIZ TEKNİK ÜNİVERSİTESİ 1911

THANK YOU

BACKUP SLIDES

Codes

Master Arm Code

```
/*
All content and software components contained in this document are Copyright ©
[2024] [Robotic Arm Control With Haptic Feedback] Project,
protected by Kadir Kadiroğlu, Özkan Yaşar, Oğuzhan Can.
Unauthorized copying, distribution, or use in any form is prohibited.
All rights reserved.
*/

#include <Wire.h>
#include <AS5600.h>
#include <SPI.h>
#include <RF24.h>

AS5600 as5600;
RF24 radiotransmitter (0,1);

// Radio Address for Master_Transmitter
const byte addr=240;

// Define Slave I2C Address
#define SLAVE_ADDR 9
uint8_t byteArray[8];

//Motor Driver Pins
const int motor1Pin1 = 14;
const int motor1Pin2 = 15;
const int motor2Pin1 = 12;
const int motor2Pin2 = 13;
const int motor3Pin1 = 10;
const int motor3Pin2 = 11;
const int motor4Pin1 = 8;
const int motor4Pin2 = 9;
```

```
//buttons
const int button_5 =22;
const int button_4 =21;
const int button_3 =20;
const int button_2 =19;
const int button_1 =18;

//leds
const int led_yellow =7;
const int led_green =6;
const int led_red =5;

void i2c_mux(uint8_t bus) {
  Wire.beginTransmission(0x70);
  Wire.write(1 << bus);
  Wire.endTransmission();    }

//pid parameters
int feedback_1;
int feedback_2;
int feedback_3;
int feedback_4;
int setpoint[4];
int rf_write[4];
int preverror=0;
int integral=0;
int preverror2=0;
int integral2=0;
int preverror3=0;
int integral3=0;
int preverror4=0;
int integral4=0;

//------------------------

//Sensitivity coefficient
byte force_coeff;
unsigned long previousTime = 0;
byte counter=0;
byte counter2=0;

bool lastButtonState = HIGH;
bool lastButtonState_2 = HIGH;
//---------Setups-------------------

void setup() {
```

```
  // pins setup------------------------
//motor driver pins
pinMode(motor1Pin1, OUTPUT);
pinMode(motor1Pin2, OUTPUT);
pinMode(motor2Pin1, OUTPUT);
pinMode(motor2Pin2, OUTPUT);
pinMode(motor3Pin1, OUTPUT);
pinMode(motor3Pin2, OUTPUT);
pinMode(motor4Pin1, OUTPUT);
pinMode(motor4Pin2, OUTPUT);

//buttons
pinMode(button_1, INPUT_PULLUP);
pinMode(button_2, INPUT_PULLUP);
pinMode(button_3, INPUT_PULLUP);
pinMode(button_4, INPUT_PULLUP);
pinMode(button_5, INPUT_PULLUP);

//leds
pinMode(led_yellow, OUTPUT);
pinMode(led_green, OUTPUT);
pinMode(led_red, OUTPUT);

//pwm setup-------------------------

analogWriteFreq(22000);
analogWriteRange(1023);
analogWriteResolution(10);

//spi setup
SPI.setSCK(2);
SPI.setTX(3);
SPI.setRX(4);

//nrf module setup------------------
radiotransmitter.begin();
radiotransmitter.setAutoAck(false);
radiotransmitter.openWritingPipe(addr);
radiotransmitter.setChannel(0);
radiotransmitter.setPALevel(RF24_PA_MAX);
radiotransmitter.setDataRate(RF24_2MBPS);
radiotransmitter.stopListening();

//i2c setup
Wire.setSDA(16);
Wire.setSCL(17);
Wire.begin();
```

```
Wire.setClock(1000000);

as5600.begin();
as5600.setDirection(AS5600_CLOCK_WISE);

Serial.begin(2000000);

}

void loop() {
bool modes=digitalRead(button_3);
bool currentButtonState = digitalRead(button_1);

if (lastButtonState == HIGH && currentButtonState == LOW) {
 if(modes){if(counter<3){counter++;}}
 if(!modes){if(counter2<3){counter2++;}}
  }

lastButtonState = currentButtonState;

bool currentButtonState_2 = digitalRead(button_2);

if (lastButtonState_2 == HIGH && currentButtonState_2 == LOW) {
 if(modes){if(counter>0){counter--;}}
 if(!modes){if(counter2>0){counter2--;}}
  }

lastButtonState_2 = currentButtonState_2;

//Serial.print("Button pressed! Counter: ");
//Serial.println(counter);
//Serial.print("Button pressed! Counter2: ");
//Serial.println(counter2);

if(modes){
digitalWrite(led_red, counter == 3 ? HIGH : LOW);
digitalWrite(led_green, counter == 2 ? HIGH : LOW);
digitalWrite(led_yellow, counter == 1 ? HIGH : LOW);
}

else{digitalWrite(led_red, counter2 == 3 ? HIGH : LOW);
digitalWrite(led_green, counter2 == 2 ? HIGH : LOW);
digitalWrite(led_yellow, counter2 == 1 ? HIGH : LOW);}

if(counter==0){
  //unilateral mode
  force_coeff=200;
```

Codes

```
  }
if(counter==1){
  force_coeff=2;
  }
if(counter==2){
  force_coeff=1.6;
  }
if(counter==3){
  force_coeff=1.3;
  }

//AS5600 Readings via TCA9548A
i2c_mux(3);
 feedback_1=as5600.readAngle()/4;
i2c_mux(4);
 feedback_2=as5600.readAngle()/4;
i2c_mux(5);
 feedback_3=as5600.readAngle()/4;
i2c_mux(2);
feedback_4=as5600.readAngle()/4;

Serial.println(feedback_1);
unsigned long currentTime = micros();
unsigned long elapsedTime = currentTime - previousTime;
 // Serial.print("Elapsed Time (in microseconds): ");
//Serial.println(elapsedTime);

rf_write[0]=feedback_1;
rf_write[1]=feedback_2;
rf_write[2]=feedback_3;
rf_write[3]=feedback_4;

// Transmitting Data via Radio Module
radiotransmitter.write(&rf_write,sizeof(rf_write));
// Receiving Data via I2C From Slave Pico hat is connected to Master Receiver
Wire.requestFrom(SLAVE_ADDR,sizeof(byteArray));

if (Wire.available() ==sizeof(byteArray)) {
for(int i=0;i<sizeof(byteArray);i++){

byteArray[i]=Wire.read();

  }
  }

// Converting 8 bytes value Array to 4 integer array
 for (int i = 0; i < 4; i++) {
```

```
 setpoint[i] = (byteArray[i * 2] << 8) | (byteArray[i * 2 + 1]);
  }

// Limits of the robot arms for each joint/link in order not to conflict the
other parts of the body
if(setpoint[0]<400){setpoint[0]=400;}
if(setpoint[0]>900){setpoint[0]=900;}

if(setpoint[1]<250){setpoint[1]=250;}
if(setpoint[1]>820){setpoint[1]=820;}

if(setpoint[2]<120){setpoint[2]=120;}
if(setpoint[2]>900){setpoint[2]=900;}

if(setpoint[3]<382){setpoint[3]=382;}
if(setpoint[3]>620){setpoint[3]=620;}

 Serial.print(',');
 //Serial.print("slvdata1 ");
 Serial.println(setpoint[0]);
 previousTime = currentTime;


//PID 1------------------------------------
 int error=(setpoint[0]-feedback_1);
 //int error=(512-feedback_1); // In order to set the Joint in the middle
angle position

integral= (integral+error);

if (integral>=1023){integral=1023;}
if (integral<=-1023){integral=-1023;}

int deriv=error-preverror;
preverror=error;
int pwm =8*error+0.1*integral+120*deriv;

if (error<=1 && error>=-1){integral=0;pwm=0; }
if (pwm>=1023){pwm=1023;}
if (pwm<=-1023){pwm=-1023;}

// Motor control part------------------
 if (pwm >=0) {
    analogWrite(motor1Pin1, pwm/force_coeff);
    analogWrite(motor1Pin2, 0);
  }
```

```
  if(pwm<0)
  {
    analogWrite(motor1Pin1, 0);
    analogWrite(motor1Pin2, -pwm/force_coeff);
  }
//PWM&MOTOR1 END


//PID 2------------------------------------
 int error2=(setpoint[1]-feedback_2);
//int error2=(512-feedback_2);
integral2= (integral2+error2);

if (integral2>=1023){integral2=1023;}
if (integral2<=-1023){integral2=-1023;}
int deriv2=error2-preverror2;
preverror2=error2;
int pwm2 =8*error2+0.1*integral2+120*deriv2;
if (error2<=1 && error2>=-1){integral2=0;pwm2=0; }
if (pwm2>=1023){pwm2=1023;}
if (pwm2<=-1023){pwm2=-1023;}

// Motor control part------------------
if (pwm2 >=0) {
    analogWrite(motor2Pin2, pwm2/force_coeff);
    analogWrite(motor2Pin1, 0);
  }
   if(pwm2<0)
  {
    analogWrite(motor2Pin2, 0);
    analogWrite(motor2Pin1, -pwm2/force_coeff);
  }
//PWM&MOTOR2 END


//PID 3------------------------------------
 int error3=(setpoint[2]-feedback_3);
 //int error3=(512-feedback_3);
 integral3= (integral3+error3);

if (integral3>=1023){integral3=1023;}
if (integral3<=-1023){integral3=-1023;}

int deriv3=error3-preverror3;
preverror3=error3;
int pwm3 =8*error3+0.1*integral3+120*deriv3;
```

Codes

```
if (error3<=1 && error3>=-1){integral3=0;pwm3=0; }
if (pwm3>=1023){pwm3=1023;}
if (pwm3<=-1023){pwm3=-1023;}


// Motor control part------------------
 if (pwm3 >=0) {
    analogWrite(motor3Pin1, pwm3/force_coeff);
    analogWrite(motor3Pin2, 0);
  }

   if(pwm3<0)
  {
    analogWrite(motor3Pin1, 0);
    analogWrite(motor3Pin2, -pwm3/force_coeff);
  }
//PWM&MOTOR3 END

//PID 4-------------------------------------

int error4=(setpoint[3]-feedback_4);
//int error4=(512-feedback_4);
integral4= (integral4+error4);

if (integral4>=1023){integral4=1023;}
if (integral4<=-1023){integral4=-1023;}
int deriv4=error4-preverror4;
preverror4=error4;
int pwm4 =9*error4+0.10*integral4+120*deriv4;

if (error4<=1 && error4>=-1){integral4=0;pwm4=0; }
if (pwm4>=1023){pwm4=1023;}
if (pwm4<=-1023){pwm4=-1023;}

// Motor control part------------------
 if (pwm4 >=0) {
    analogWrite(motor4Pin1, pwm4);
    analogWrite(motor4Pin2, 0);
  }

   if(pwm4<0)
  {

    analogWrite(motor4Pin1, 0);
    analogWrite(motor4Pin2, -pwm4);

  }
```

```
//PWM&MOTOR4 END


}
```

Master Arm I2C Slave Code

```
/*
All content and software components contained in this document are Copyright ©
[2024] [Robotic Arm Control With Haptic Feedback] Project,
protected by Kadir Kadiroğlu, Özkan Yaşar, Oğuzhan Can.
Unauthorized copying, distribution, or use in any form is prohibited.
All rights reserved.
*/

#include <Wire.h>
#include <SPI.h>
#include <RF24.h>
RF24 radioreceiver (0,1);
const byte addr=20;

// Define Slave I2C Address
#define i2c_slave_addr 9
 uint8_t byteArray[8];

// data array to get data from rf module wirelessly
int rf_read[4];

void setup() {
//spi setup
SPI.setSCK(2);
SPI.setTX(3);
SPI.setRX(4);

//nrf module setup as receiver-------------------
radioreceiver.begin();
radioreceiver.setAutoAck(false);
radioreceiver.openReadingPipe(1, addr);
radioreceiver.setChannel(124);
radioreceiver.setPALevel(RF24_PA_MAX);
radioreceiver.setDataRate(RF24_2MBPS);
radioreceiver.startListening();

//i2c setup
Wire.setSDA(16);
Wire.setSCL(17);
// Initialize I2C communications as Slave
Wire.begin(i2c_slave_addr);
Wire.setClock(1000000);
// Function to run when data requested from master
Wire.onRequest(requestEvent);
}


void requestEvent() {
Wire.write(byteArray, sizeof(byteArray));
}

void loop() {
if (radioreceiver.available()) {

while(radioreceiver.available()){
radioreceiver.read(&rf_read,sizeof(rf_read));
 }

}

//Serial.println(rf_read[0]);
// Convert integer array to byte array

for (int i = 0; i < 4; i++) {
  byteArray[i * 2] = rf_read[i] >> 8;  // Extract high byte (shift right by 8)
  byteArray[i * 2 + 1] = rf_read[i] & 0xFF;  // Extract low byte (mask with
0xFF)
}

}
```