

위치 센서를 이용한 로봇 팔 원격제어

결과 보고서

작성 및 검토 확인란

구분	학번	성명	서명
작성자	20184199	신통화	
	20184124	박준호	
	20205062	최찬희	
	20214897	이행승	

개정 이력

개정일자	버전	개정내용	작성자	확인자
2024.12.23		결과보고서 작성	신통화	

목 차

최찬희 결과 보고서	3
박준호 결과 보고서	7
이행승 결과 보고서	14
신통화 결과 보고서	27

* 제가 맡은 역할은 1) 라즈베리파이와 ESP32 간 통신 모듈 개발, 2) 로봇 손가락 제어를 위한
압력 센서 데이터 수집 및 모듈 개발입니다.

* 진도보고서1 기간-----

Esp32 관련 자료조사 후 Esp32와 라즈베리파이 블루투스 연결 관련 코드를 만들었습니다. 아두이노 스케치로 Esp32 블루투스 연결, 센서값 송수신 코드를 먼저 업로드를 해주었고, 라즈베리파이에서 다른 제어 코드를 실행시켜야 하기 때문에 항상 아두이노 코드, 라즈베리파이 코드 각각 1개씩 필요했습니다. 코드 테스트 사이트에서 두 코드를 실행 하여보았는데 관련 모듈을 사지 않은 상태여서 오류가 발생했습니다.



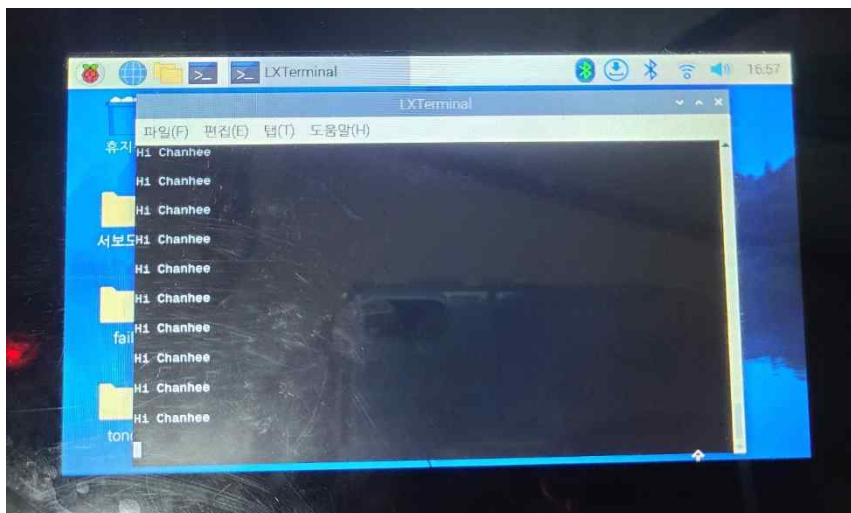
JINDO1_esp32.ino



JINDO1_raspberry
pi.py

* 진도보고서 2 기간-----

Esp32와 라즈베리파이 간의 블루투스 연결을 설정하고, 두 장치 간에 데이터가 안정적으로 송수신될 수 있도록 테스트하였습니다. 블루투스 연결후 Esp32 에서 "Hi Chanhee" 데이터를 라즈베리파이로 송신하면 라즈베리파이 터미널 창에서 **sudo cat /dev/rfcomm0** 명령어로 수신 데이터("Hi Chanhee")를 받을 수 있었습니다.

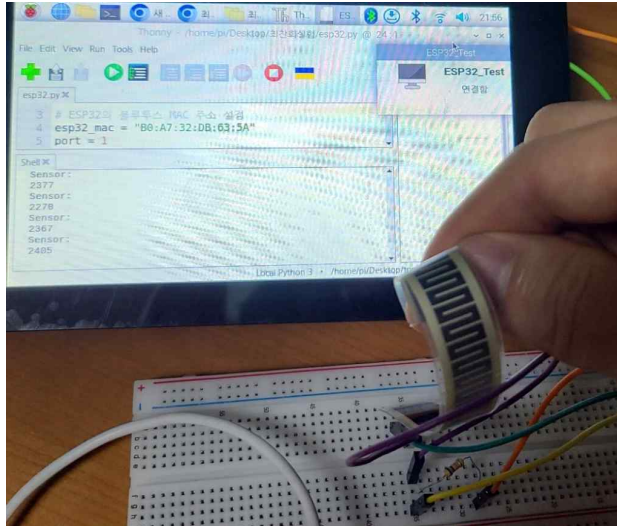


JINDO2_esp32.ino

[블루투스 연결후 esp32
에서 송신한 'Hi
Chanhee'를 라즈베리파
이 터미널 창에서 수신]

* 진도보고서 3 기간-----

압력 센서값을 Esp32로 받고 라즈베리파이에 송신하는 아두이노 코드를 Esp32에 업로드하고, 라즈베리파이에 Esp32 블루투스 설정, 압력 센서값을 화면에 출력하는 코드를 실행하였습니다. 그리고 압력 센서를 구부려서 센서값이 라즈베리파이 화면에 일관되게 증가하는 것을 확인했습니다.



JINDO3_esp32.ino



JINDO3_raspberry
pi.py

[압력 센서를 굽히니 센서값이 2377->2405로 증가하는 모습]

* 진도보고서 4 기간-----

박준호 조원과 협업하여 압력 센서 값에 따라 서보 모터가 일관되게 동작하도록 구현하는 테스트를 하였습니다. 테스트 완료 후 서보 모터를 로봇 집게 손에 장착해서 물건을 집는 테스트도 완료 하였습니다. 아두이노 코드는 저번 주 코드 그대로 변경하지 않고 Esp32 메모리에 저장해놨습니다. 라즈베리파이 코드는 'Esp32 압력 센서+서보 모터 제어'를 개발해 실행하였습니다.



JINDO4_esp32.ino



JINDO4_raspberry
pi.py

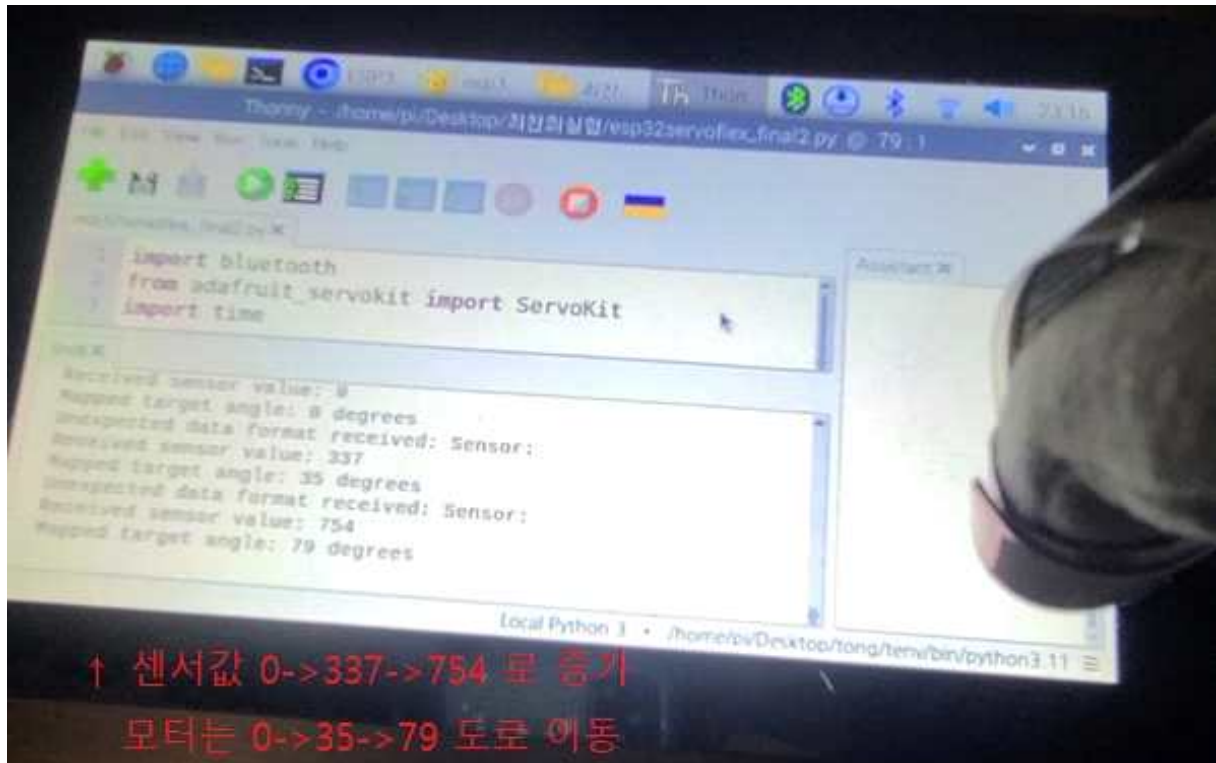
* 진도보고서 1~4 이후 최종시연까지의 기간-----

압력 센서 초기값이 0이 아닌 4095인 문제를 해결하기 위해 기존의 Esp32 아두이노 코드에서

```
int invertedValue = 4095 - sensorValue;
```

```
Serial.println(invertedValue);
```

SerialBT.println(invertedValue); 코드들을 추가하여 반전된 값을 블루투스로 전송시켜 주었습니다.



그렇게 압력 센서 초기값을 0으로 만들고 센서를 굽힐수록 센서값이 올라가는 것을 확인할 수 있었습니다.



Final_esp32.ino



Final_raspberry pi.py

그리고 압력 센서를 장갑에 부착했을 때 0~4095까지의 센서값을 보낼 수 있을 정도로 손가락을 굽히기가 어려웠습니다. 그래서 기존의 라즈베리파이 코드에서 센서값을 0~1700으로 제한하는 코드를 추가시켰습니다.

```
sensor_value = max(0, min(sensor_value, 1700))
```

```
# 센서값을 0~1700으로 제한
```

```
target_angle = max(0, min(int(map_value(sensor_value, 0, 1700, 0, 180)), 180))
```

```
# 센서값을 0~1700에서 0~180도로 매핑
```

따라서 압력 센서값을 0~1700으로 제한했을 때 안정적으로 물건을 집고 놓을 수 있었습니다.



[초기 상태]


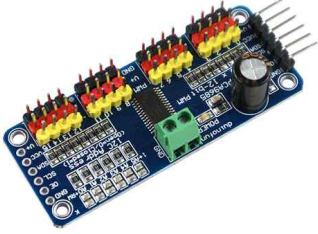



[손가락을 굽혀서 물건(마우스)을 집은 상태]

1. 프로젝트 개요 및 목표

- **목표:** 위치 센서와 서보 모터를 활용한 로봇 팔 제작 및 원격 제어 시스템 구현.

- **기술 사용:**

		
MG996R	PCA9685	ESP32
높은 토크와 합리적인 가격으로 로봇 팔 관절 제어에 적합하다.	다수의 서보 모터를 제어하며, I2C 통신을 통해 Raspberry Pi와 연동한다.	압력 센서로부터 데이터를 수집하여 블루투스를 통해 Raspberry Pi로 전송한다.

로봇 기술의 발전과 사용자 인터페이스의 개선은 인간의 동작을 모방하거나 지원하는 로봇 시스템 개발에 대한 관심을 크게 증대시키고 있다. 특히 인간의 미세한 움직임을 정밀하게 감지하고 이를 로봇 시스템에 실시간으로 반영하는 기술은 의료, 재활, 산업 자동화, 엔터테인먼트 등 다양한 분야에서 응용 가능성이 높다. 본 프로젝트는 기술적 요구에 부응하는 로봇 시스템 개발의 실질적인 가능성을 확인하고, 인간과 로봇의 상호작용성을 강화하는 데 기여하고자 한다.

2. 하드웨어 구성

- 서보 모터 선정

	SG90	MG996R	DS3218
모습			
작동 전압	4.8V ~ 6.0V	4.8V ~ 7.2V	4.8V ~ 6.8V
토크	1.8 kg·cm (4.8V)	9.4 kg·cm (4.8V) 11 kg·cm (6.0V)	17 kg·cm (4.8V) 21 kg·cm (6.8V)
회전 각도	0° ~ 180°	0° ~ 180°	0° ~ 270°
가격	1,700원	3,200원	24,000원

- SG90: 소형 및 저토크, 정밀 제어가 필요할 때 적합하다.
- MG996R: 중간 토크로 로봇 팔 관절 제어에 적합하다.
- DS3218: 고토크로 무거운 로봇 팔 제어에 적합하다.

각 모터들에 장단점과 토크 가격 등을 고려하여 최종적으로 MG996R 모델을 사용하기로 결정하였다.

- PCA9685 연동:

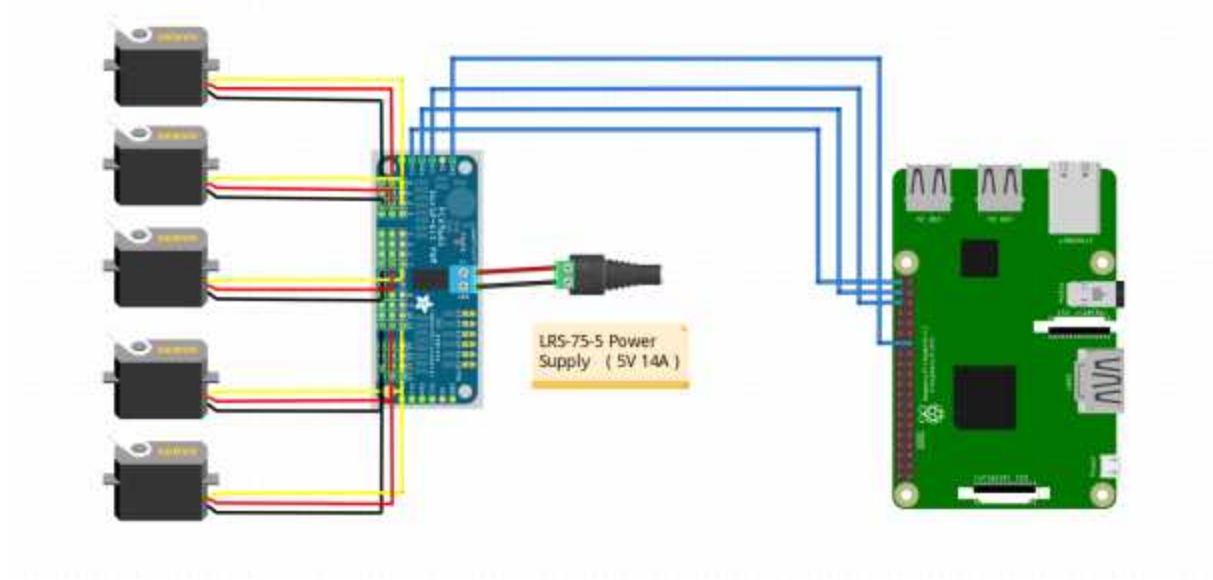
PCA9685는 16채널 PWM 드라이버 칩으로, 주로 서보 모터, DC 모터, LED 등을 제어하는데 사용된다. 또한 I2C 통신을 통해 제어할 수 있어 라즈베리 파이와 쉽게 연결할 수 있으며 12비트 분해능, 주파수 설정 기능, 외부 전원 지원 등의 특징이 있어 여러 장치를 동시에 제어하기 적합하다.

전원 공급은 Mean Well사의 LRS-75-5 모델을 사용한다. 이 모델은 70W 단일 출력 스위칭 전원 공급 장치로서, 5V DC 출력 전압과 최대

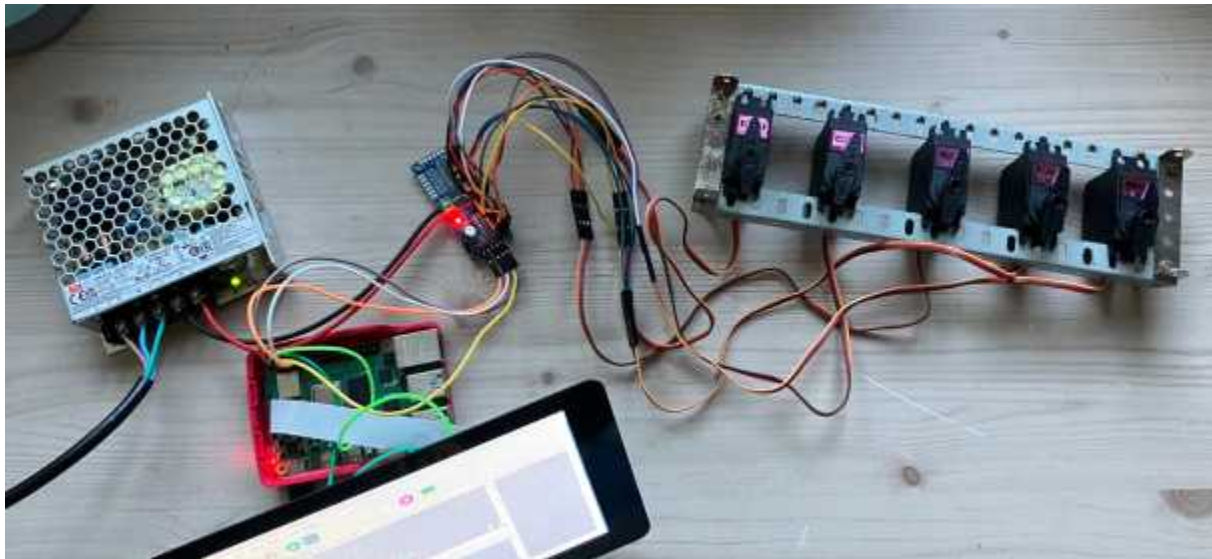
14A의 출력 전류를 제공하는 제품이다.

			
모델명	PCA9685	모델명	LRS-75-5
주파수	40~1000Hz	출력 수	1
채널	16 채널	전압 - 출력	5V
해상도	12 비트	전류 - 출력	14A
전압	DC 5~10V	전력(와트)	70W
가격	6,600원	가격	17,600원

- 회로 설계



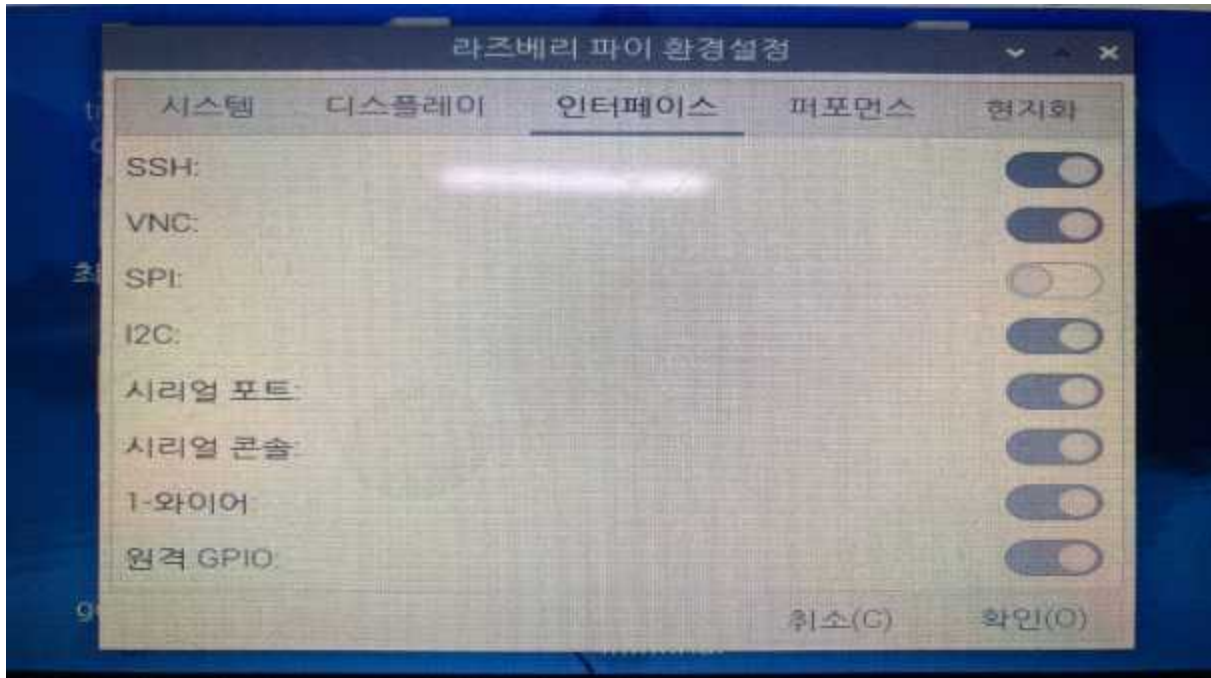
[그림 1] 회로 구성도



[그림 2] 실제 회로 모습

3. 소프트웨어 구성

- Raspberry Pi 설정:



[그림 3] I2C 활성화.



[그림 4] CircuitPython 기반 ServoKit 라이브러리 설치.

- ESP32 연동:

```

1 import bluetooth
2 from adafruit_servokit import ServoKit
3 import time
4
5 # ESP32 블루투스 설정
6 esp32_mac = "B8:A7:32:DB:63:5A"
7 port = 1
8
9 # 서보 모터 초기화
10 kit = ServoKit(channels=16)
11 servo_channel = 0 # 서보 모터가 연결된 채널
12 MIN_PULSE = 500 # 최소 펄스 (0도)
13 MAX_PULSE = 2500 # 최대 펄스 (180도)
14 kit.servo[servo_channel].set_pulse_width_range(MIN_PULSE, MAX_PULSE)
15
16 # 현재 각도 상태 저장
17 current_angle = 0
18 kit.servo[servo_channel].angle = current_angle # 초기 각도 설정
19 print(f"Servo motor initialized to {current_angle} degrees.")
20
21 # 압력 센서 값을 각도로 매핑하는 함수
22 def map_value(value, from_min, from_max, to_min, to_max):
23     # 선형 매핑 함수: 입력 범위를 출력 범위로 변환
24     return to_min + (value - from_min) * (to_max - to_min) / (from_max - from_min)
25
26 # 모터를 1도 단위로 부드럽게 이동하는 함수
27 def move_servo_smoothly(target_angle):
28     global current_angle
29     if current_angle < target_angle:
30         step = 1 # 증가 방향으로 1도씩 이동
31     elif current_angle > target_angle:
32         step = -1 # 감소 방향으로 1도씩 이동
33     else:
34         return # 이미 목표 각도에 도달한 경우
35
36     for angle in range(current_angle, target_angle + step, step):
37         kit.servo[servo_channel].angle = angle
38         time.sleep(0.01) # 각도 이력 간격 조정 (10ms 대기)
39
40     current_angle = target_angle # 현재 각도를 목표 각도로 갱신
41
42 try:
43     # 블루투스 소켓 연결
44     print("Connecting to ESP32 via Bluetooth...")
45     sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
46     sock.connect((esp32_mac, port))
47     print("Connected to ESP32")
48
49     while True:
50         # 데이터 수신
51         data = sock.recv(1024).decode("utf-8").strip()
52
53         try:
54             # 수신 데이터 처리
55             sensor_value = int(data) # "4095", "3000" 형식으로 수신된 값을 처리
56             print(f"Received sensor value: {sensor_value}")
57
58             # 센서 값을 반대로 매핑 (4095~0 -> 0~180도)
59             target_angle = int(map_value(sensor_value, 4095, 0, 0, 180))
60             print(f"Mapped target angle: {target_angle} degrees")
61
62             # 모터를 부드럽게 이동
63             move_servo_smoothly(target_angle)
64
65         except ValueError:
66             # 숫자가 아닌 데이터 처리
67             print(f"Unexpected data format received: {data}")
68
69         time.sleep(0.1) # 상태 업데이트 간격
70     except KeyboardInterrupt:
71         print("\nExperiment stopped by user.")
72     finally:
73         # 블루투스 소켓 해제 및 서보 모터 비활성화
74         sock.close()
75         kit.servo[servo_channel].angle = None
76         print("Connection closed and servo motor deactivated.")

```

[그림 5] 압력 센서 데이터를 기반으로 서보 모터각도 매핑, 데이터 전송 주기 설정 (100ms).

- 테스트 결과



4. 로봇 팔 제작 및 테스트

- 집게(Gripper) 설계:



[그림 6] 로봇팔 집게

서보 모터는 ESP32와 연결된 압력 센서를 통해 제어되며, 압력 센서값을 모터의 회전 각도로 매핑하여 동작을 구현한다. 센서가 정지 상태일 때 값은 4095이며, 이때 서보 모터의 각도는 0° 로 설정되어 집게가 벌어진 상태가 된다. 압력 센서를 구부리면 센서값이 0까지 감소하며, 모터는 각도를 증가시켜 집게를 점점 오므리는 구조로 설계되었다.

또한, step 함수를 이용해 각도를 1도씩 부드럽게 조정하며, `time.sleep(0.01)`으로 10ms 간격을 설정해 모터가 너무 빠르게 움직이지 않도록 제어한다. 이 방식은 실시간 센서 데이터에 따라 모터 각도를 정밀히 반영해 로봇 팔 제어에 적합하다.



위치센서를 이용한 로봇팔 원격 제어 진도보고서

작성자 : 이형승

조 : 4조(팔로미)

학번 : 20214897

제출일 : 2024.12.03

제 1 장. 설계의 배경 및 목표

1.1 설계 배경

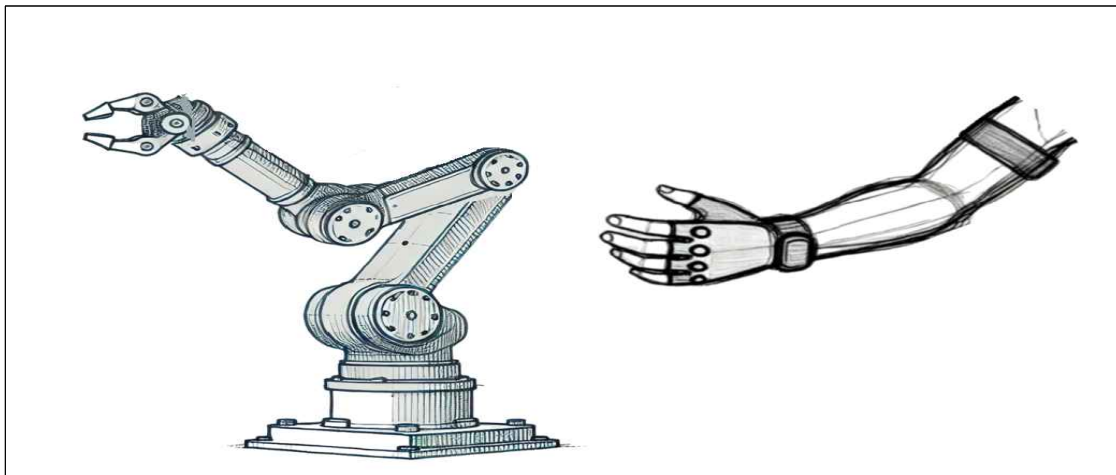
- 최근 로봇 기술은 인간의 정교한 동작을 모방하는 방향으로 발전하고 있다. 특히, 손과 같은 정밀한 동작이 요구되는 로봇팔은 의료, 제조, 서비스 분야에서 중요한 역할을 하고 있다. 본 설계는 이러한 기술 발전에 기여하고자 사람의 손 움직임을 모방하는 로봇팔의 하드웨어 모델링을 목표로 한다.

1.2 설계 목표

- Fusion 360을 사용하여 로봇팔의 주요 관절 구조와 비율을 손 움직임에 맞게 설계한다.
- 설계한 모델을 3D 프린터로 출력한다.
- 출력물을 조립하여 완성한다.

제 2 장. 설계 및 모델링

2.1 초기 스케치 계획

















간단한 구조도

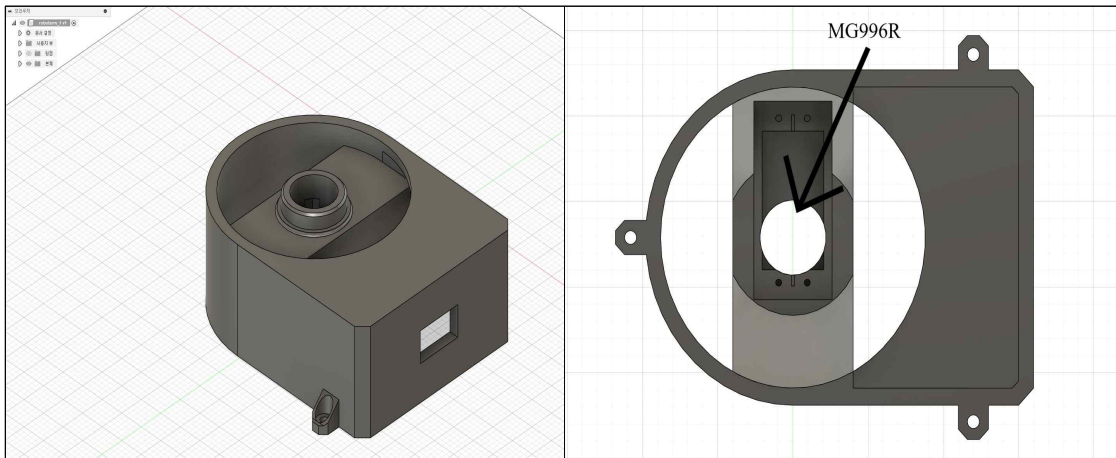
1. 바닥에서 첫 번째 관절까지 18cm
2. 첫 번째 관절에서 두 번째 관절까지 18cm
3. 두 번째 관절에서 세 번째 관절까지 12cm
4. 최종적으로 그리퍼를 부착하여 전체 구조를 완성한다.

2.2 3D 모델링

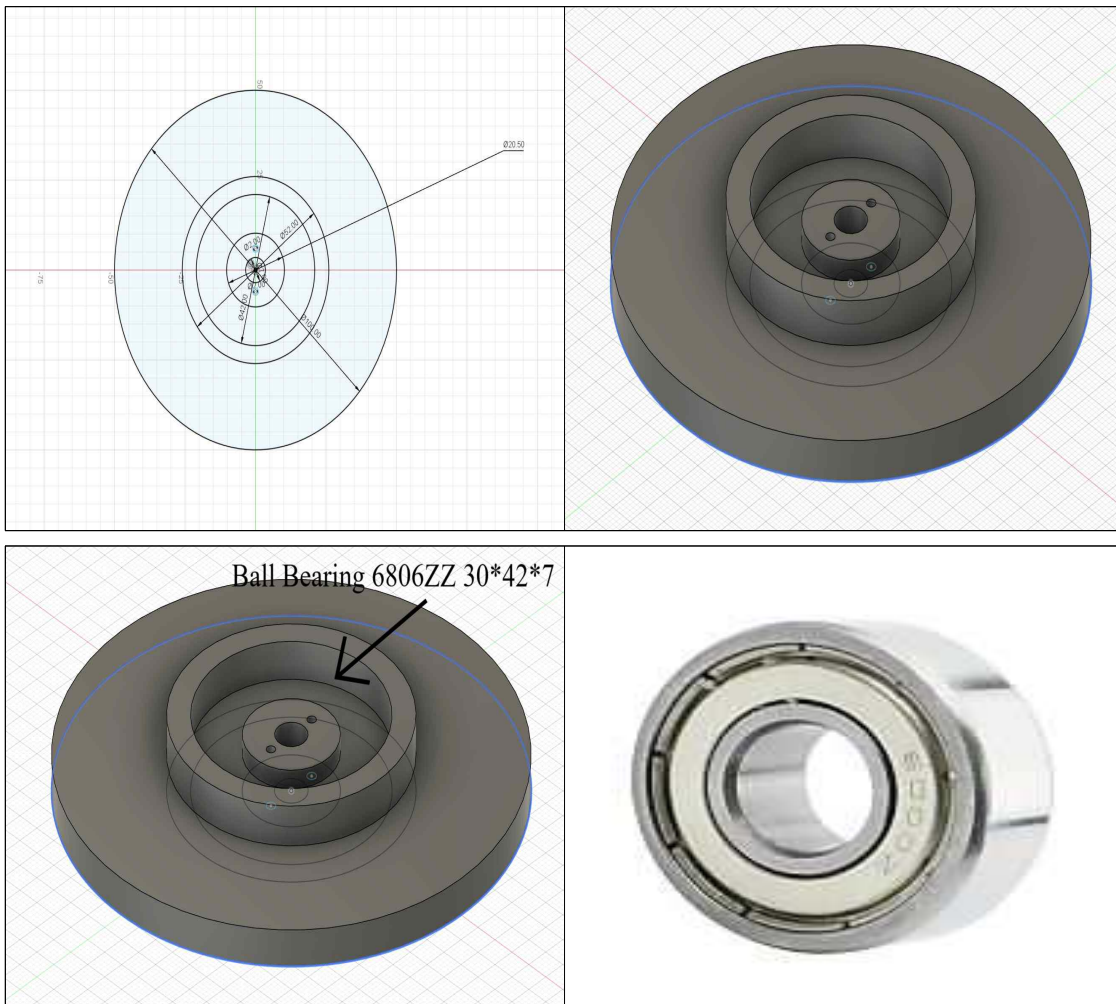
☆ 전체 부품

	gripper_1 2024-11-25 오후 11:32	V2 ▾
	gripper_2 2024-11-25 오후 11:33	V2 ▾
	gripper_3 2024-11-25 오후 11:33	V2 ▾
	gripper_4 2024-11-25 오후 11:33	V2 ▾
	gripper_5 2024-11-25 오후 11:34	V2 ▾
	robotarm_1 2024-11-25 오후 11:24	V1 ▾
	robotarm_2 2024-11-25 오후 8:41	V3 ▾
	robotarm_3 2024-11-25 오후 8:30	V12 ▾
	robotarm_3.5 2024-11-26 오전 2:08	V2 ▾
	robotarm_4 2024-11-25 오후 8:29	V8 ▾
	robotarm_5 2024-11-25 오후 8:28	V6 ▾
	robotarm_6 2024-11-25 오후 10:58	V1 ▾
	robotarm_7 2024-11-25 오후 9:26	V2 ▾
	robotarm_final 2024-11-26 오전 12:46	V4 ▾

☆ robotarm_1

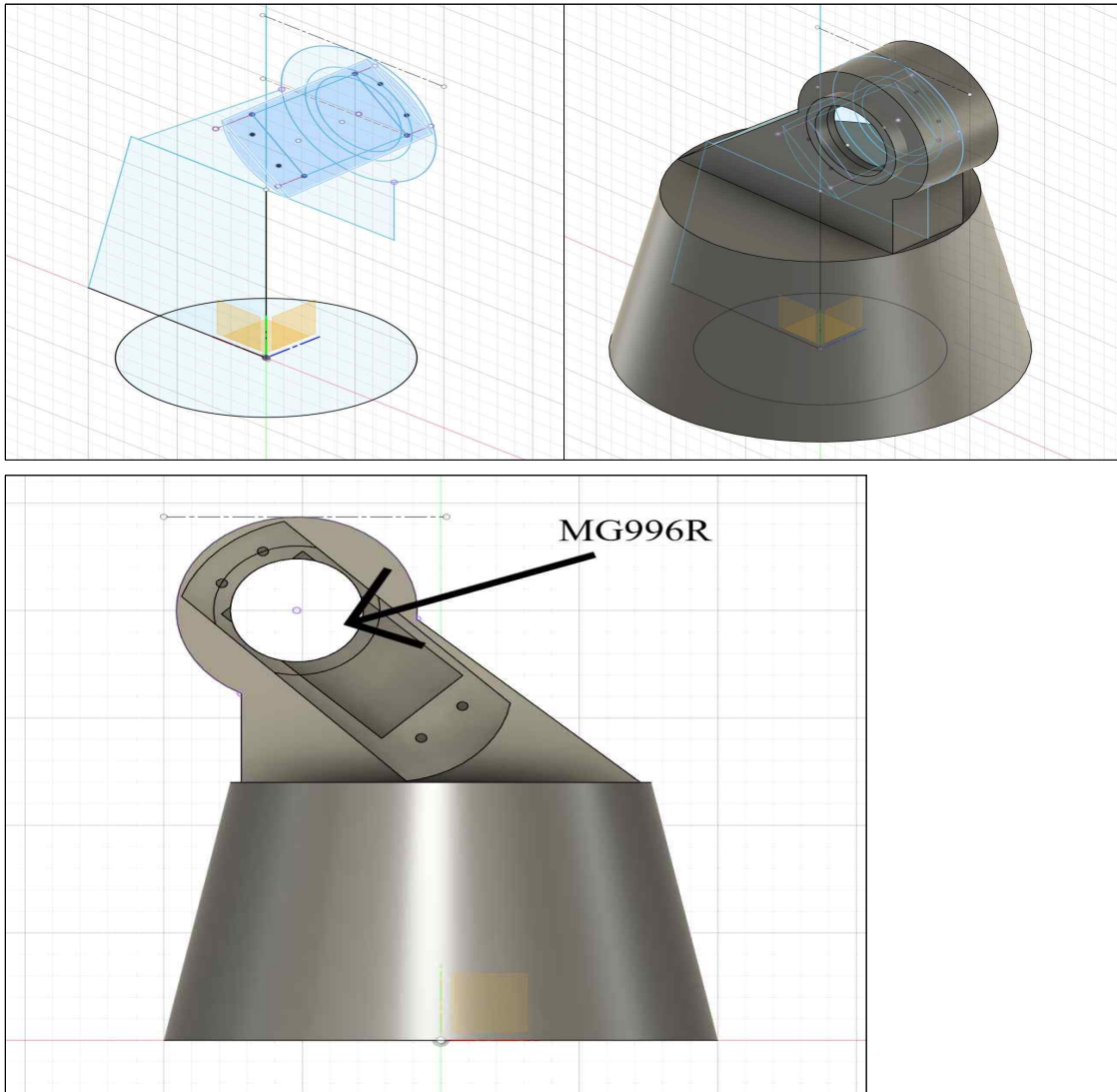


☆ robotarm_2



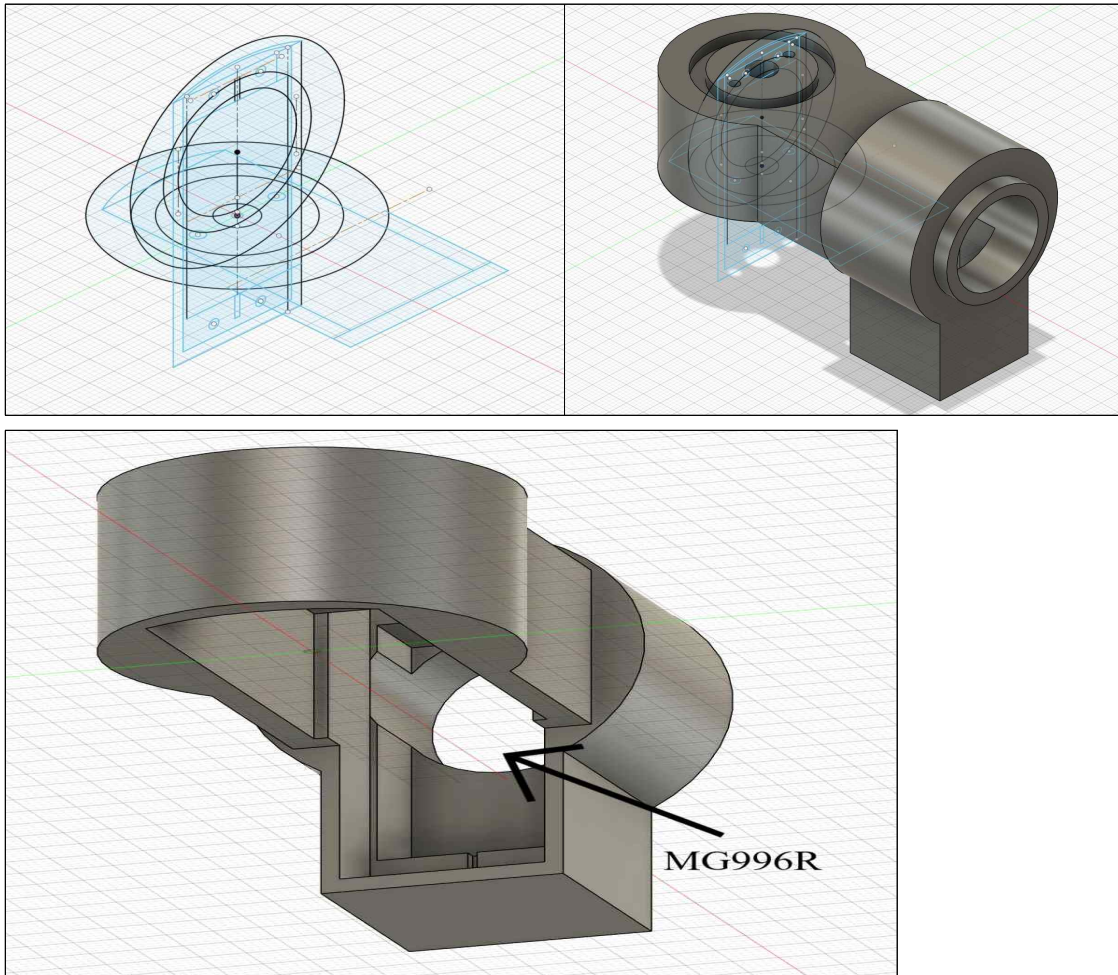
- robotarm_2에는 베어링을 부착한다.

☆ robotarm_3

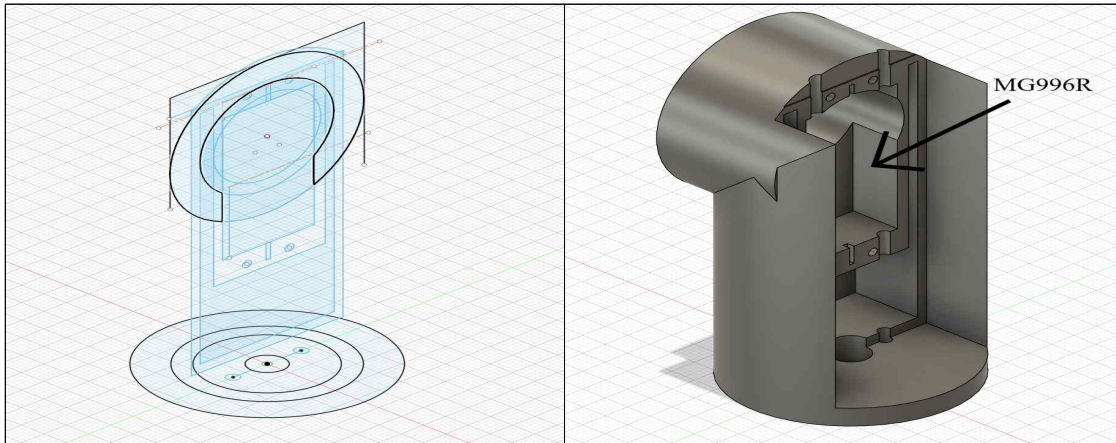


- robotarm_2, robotarm_3은 접착제로 고정한다.

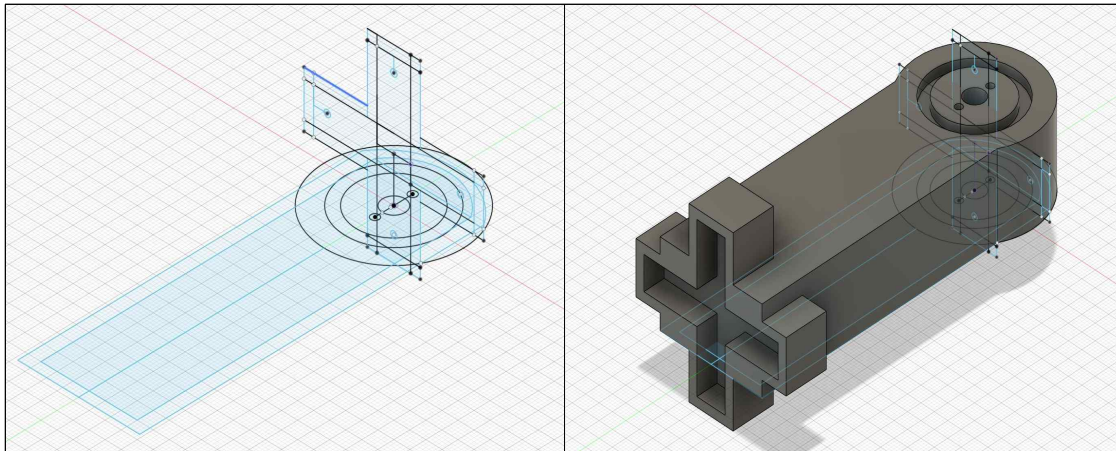
☆ robotarm_4



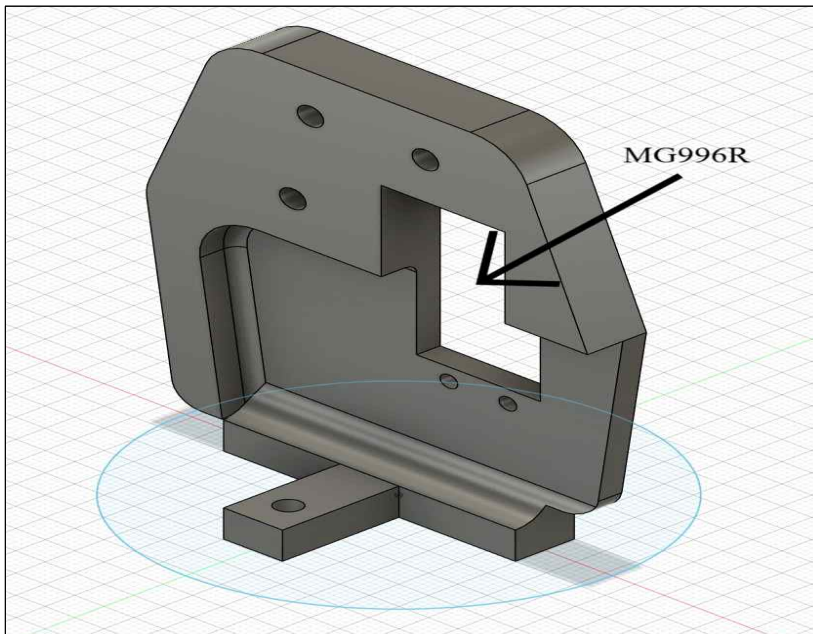
☆ robotarm_5



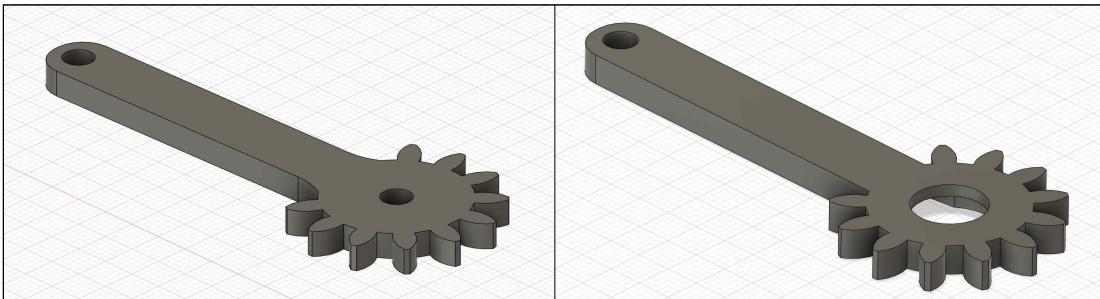
☆ robotarm_6



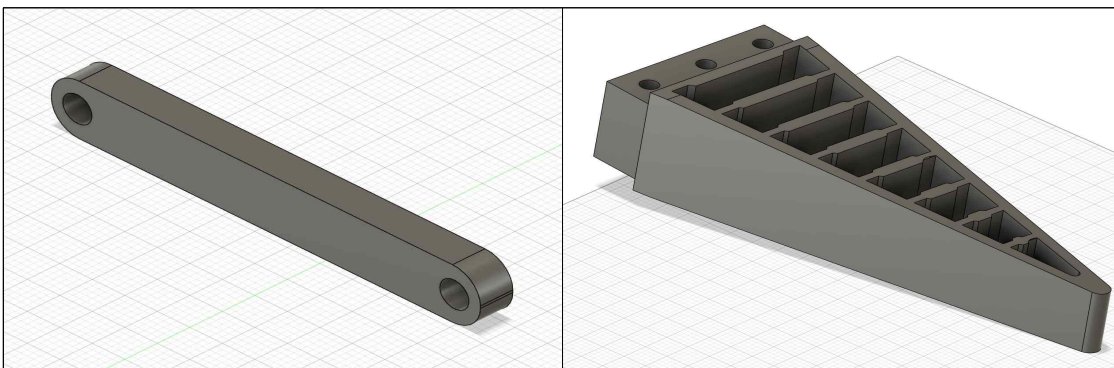
☆ robotarm_7



☆ gripper_1, gripper_2

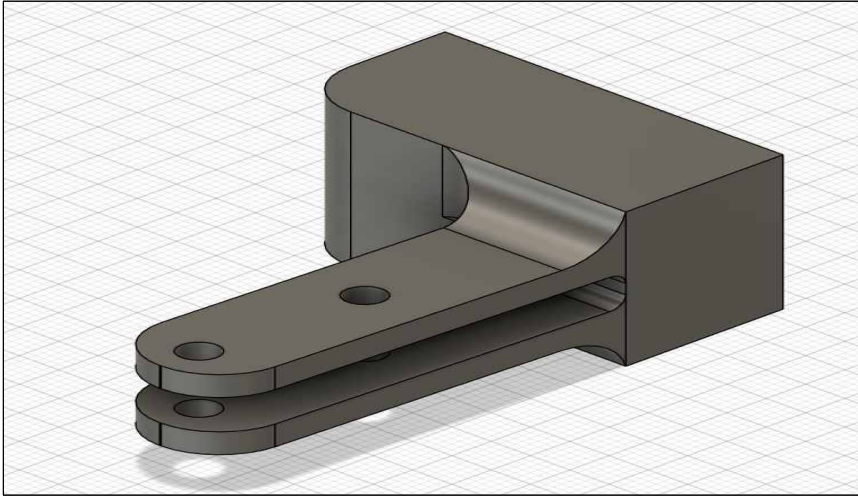


☆ gripper_3, gripper_4



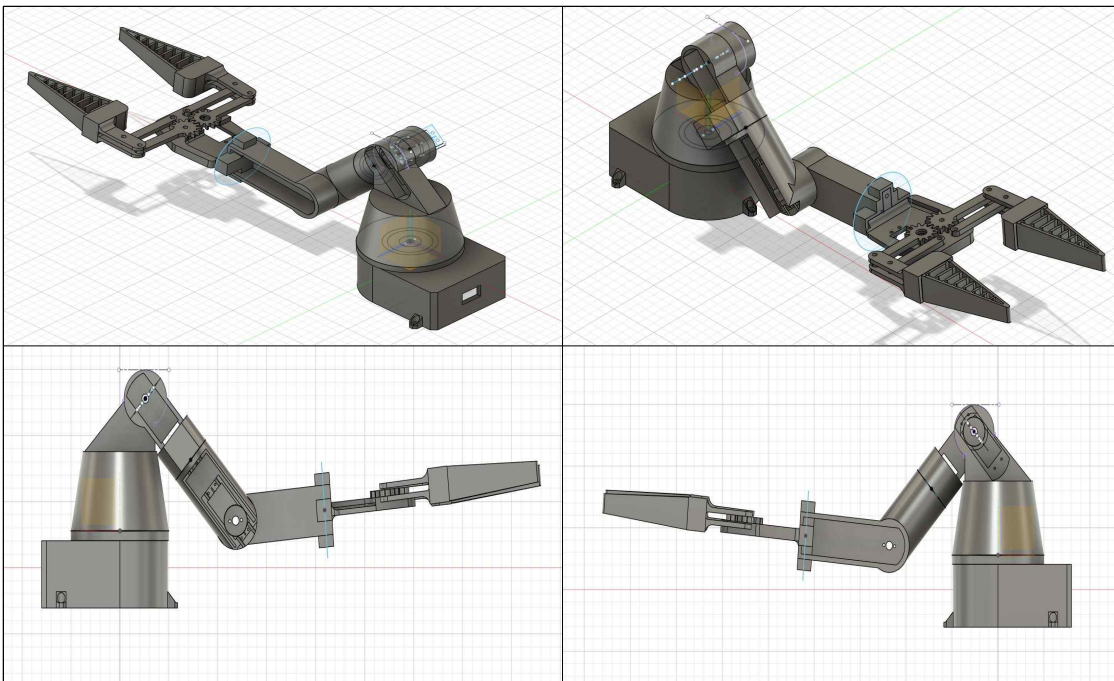
- 2개씩 출력한다.

☆ gripper_5



- 2개씩 출력한다.

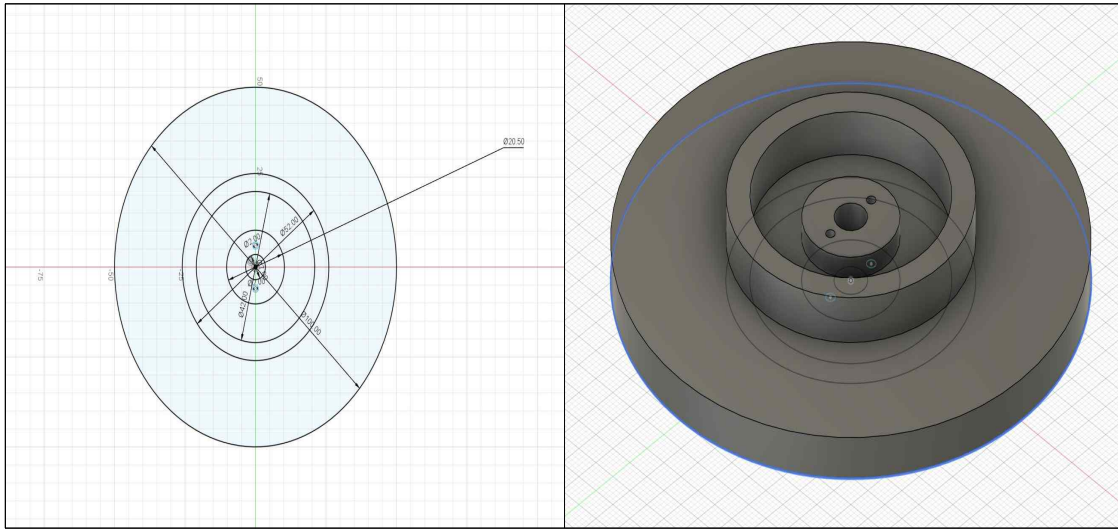
☆ 3D 모델 전체 뷰



- 베이스, 링크, 관절, 그리퍼 등 각 부품을 조립하여 로봇팔의 완성된 구조를 구성하였다. 조립 과정에서 Fusion 360의 조립 기능을 활용하여 각 부품의 연결 상태를 점검하고, 부품 간 간섭이나 충돌이 발생하지 않도록 세심히 배치하였다. 이를 통해 로봇팔이 원활하게 작동할 수 있는 구조를 설계하였다.
- 조립 후, 각 부위의 회전과 움직임을 애니메이션으로 구현하여 로봇팔의 동작을 시각적으로 검증하였다. 애니메이션을 통해 관절의 회전 각도, 링크의 이동 범위, 그리퍼의 개폐 동작 등을 세부적으로 확인하였다.

2.3 출력물 제작 및 문제 해결 과정

☆ robotarm_2



2.3.1 출력 중 발생한 스파게티 오류

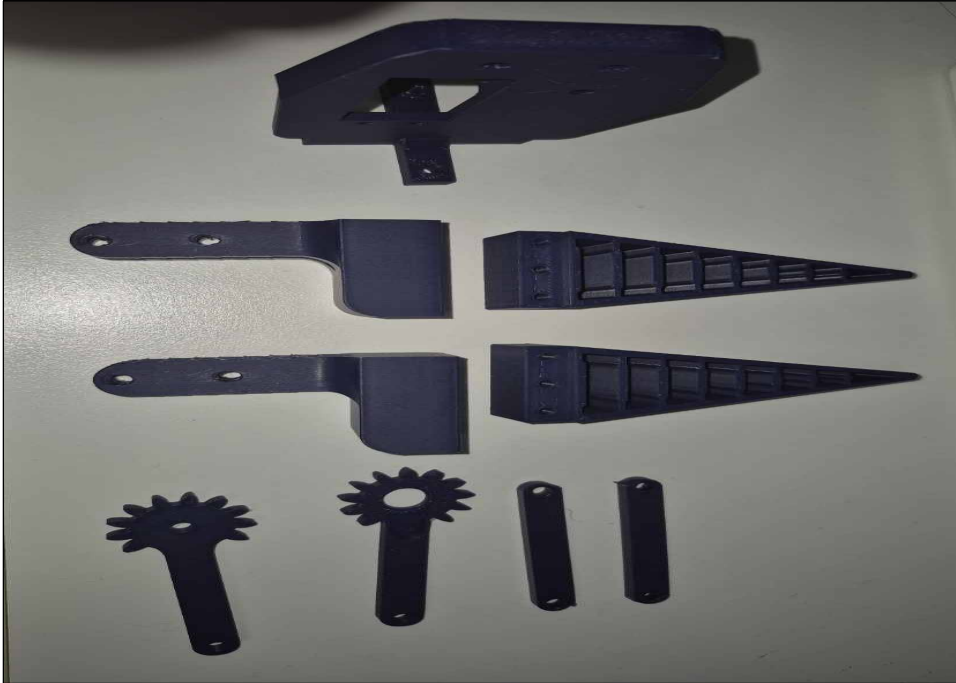


출처: pinshape

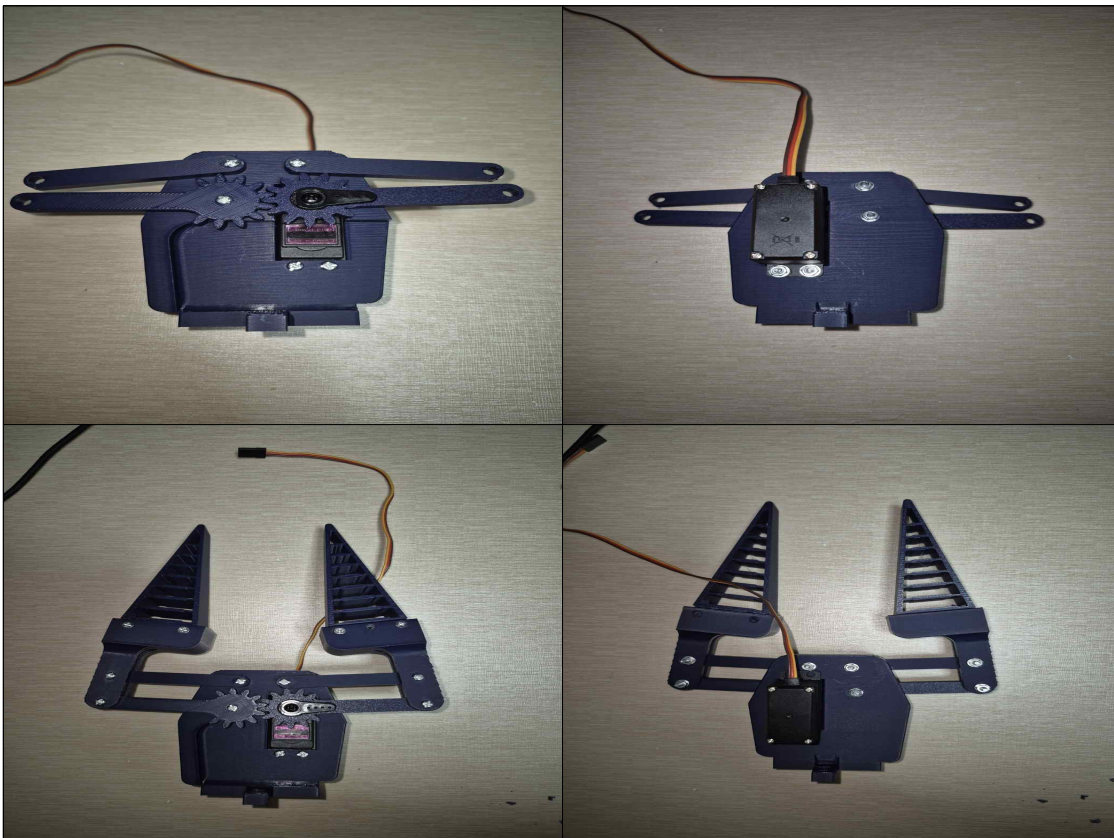
- robotarm_2 부품을 출력하는 중에 스파게티 오류가 발생하였다. 이는 출력물이 제대로 안착되지 않고 엉키는 현상으로, 흔히 안착불량이라고 불린다. 출력 작업 중 자리를 비운 사이 이런 문제가 발생했음을 확인하였다.
- 해당 오류는 주로 프린터 베드와 출력물 간 접착 불량으로 인해 발생하며, 필라멘트가 제대로 고정되지 않아 출력물이 엉키는 현상이 나타난다.
- 나의 경우, 출력실의 온도가 낮아 프린터 베드의 온도 유지가 어려워진 것이 원인으로 추정된다.
- 이를 해결하기 위해 프린터 베드의 최소 온도 설정을 기존 60°C에서 65°C로 올리고, 출력실의 환경을 따뜻하게 유지하는 방식으로 문제를 해결하였다.

2.3.2 그리퍼 문제와 모델링 수정

☆ gripper



☆ gripper 조립

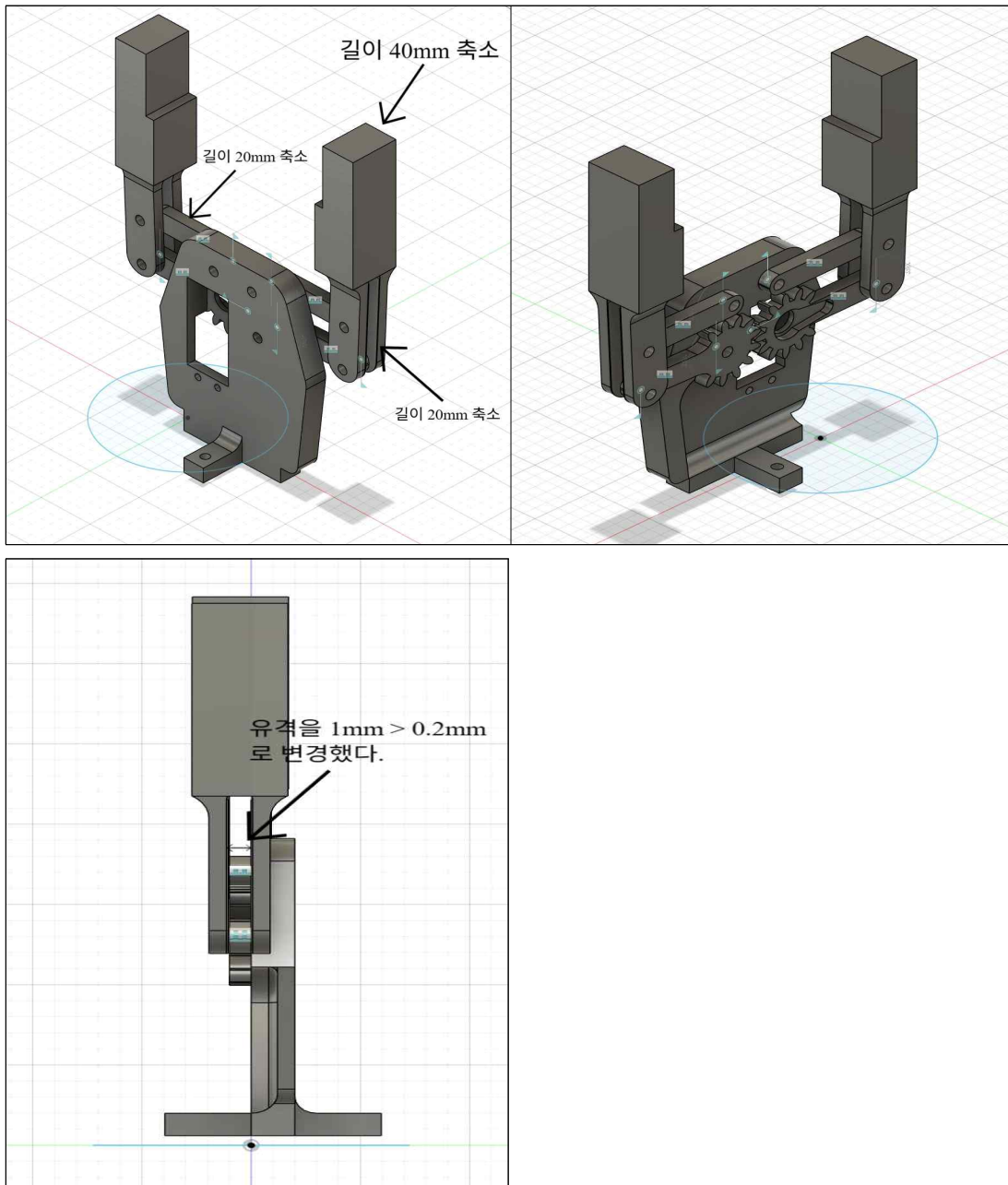


☆ gripper 테스트



- 초기 설계된 그리퍼의 끝부분이 삼각형 모양으로 되어 있어 물체를 안정적으로 잡기 어렵다는 문제가 확인되었다. 특히, 그리퍼 끝의 간격이 불필요하게 커서 설계상 효율적이지 않은 것으로 판단되었다.
- 또한, 설계 과정에서 비대칭성이 발견되었고, 조립 후 일부 부품에서 덜컹거림이 발생하여 구조적 안정성을 개선할 필요가 있었다.

☆ gripper 모델링 수정



1. 그리퍼 끝의 모양을 삼각형에서 직사각형으로 변경하여 물체를 더 안정적으로 잡을 수 있도록 설계했다.
2. 그리퍼의 전체 길이를 축소하여 설계 효율성을 높였다.
3. 그리퍼를 옆에서 바라봤을 때 유격을 줄여 덜킹거림이 발생하지 않도록 수정하였다.
4. 고무 팁이 부착될 수 있는 홈을 추가하여, 물체를 잡을 때 미끄러짐을 방지하고 잡는 능력을 개선하였다.
5. 조립 시 나사를 조일 때, 한쪽만 바로 꽉 조이지 않고 양쪽을 번갈아가며 천천히 조여 안정적인 연결을 확보하였다.

제 3 장. 최종 출력물 및 결과 분석

3.1 최종 출력물



- robotarm_1
- robotarm_2
- robotarm_3
- robotarm_4
- robotarm_5
- robotarm_6
- 수정한 모델의 설계가 안정성과 기능 면에서 개선되었으므로, 이를 기반으로 그리퍼를 출력할 예정이다.

3.2 설계 과정의 평가 및 개선점

☆ 설계 과정 평가

- 모델링 과정을 통해 로봇팔의 주요 구조와 기능을 구체화하였다. 출력 과정에서 발생한 문제를 해결하며 부품의 설계와 출력 환경의 중요성을 이해할 수 있었다.

☆ 개선점

- 출력된 부품의 조립 및 테스트 과정을 통해 설계의 완성도를 추가로 검증할 필요가 있다.
- 출력 환경의 안정성을 확보하기 위해 프린터 설정 및 작업 환경 관리(온도, 습도 등)를 강화해야 한다.
- 부품 간 연결부 설계를 보다 정밀하게 개선하여 조립 후 안정성을 높일 필요가 있다.

10월 15일 진도 보고서

제가 맡은 역할은 라즈베리파이 프로그래밍 환경 조성 및 IMU센서 제어입니다. 해당 역할을 수행하기 위해 우선 물품을 구매하기로 하였고 라즈베리파이는 지난 학기에 사용하던게 있기에 IMU센서에 대해 좀더 알아보았습니다.

해당 센서는 위치 뿐만이 아니라 속도와 방향까지 측정할 수 있습니다.

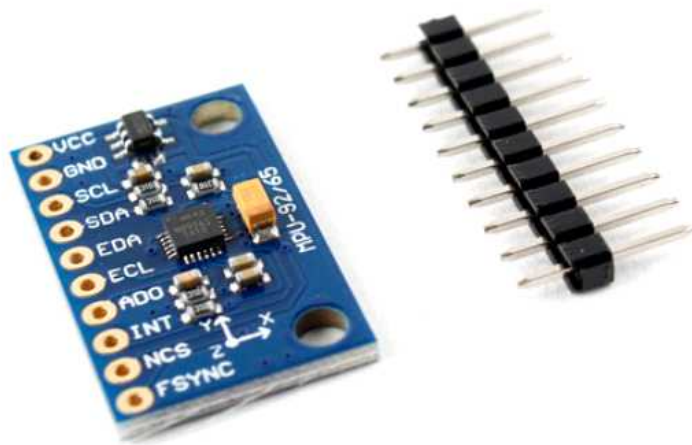
이를 측정할 수 있게 해주는 주요 3가지 구성요소가 있는데 이는 각각 가속도계, 자이로스코프, 자기(력)계이며 각각 3축씩 총 9축으로 이루어져 있습니다.

가속도계는 내부에 가속도에 반응하는 미세한 질량이 부착되어있는데 가속도가 생기면 해당 질량이 변화하여 이를 통해 가속도를 측정할 수 있게 됩니다

자이로스코프는 물체의 각속도를 측정 하여 물체의 회전 방향과 각도를 결정할 수있습니다.

자기(력)계는 자기장의 강도를 측정하고 그 방향을 알 수 있습니다. 기본적으로는 지구의 자기장을 따라 방향을 정할수 있습니다.

저렴한 IMU센서는 가속도계와 자이로스코프를 사용한 6축 센서가 있었지만 조의 목표는 방향도 중요하였기에 9축센서인 MPU-9250제품을 구입 하였고 현재 배송중에 있습니다

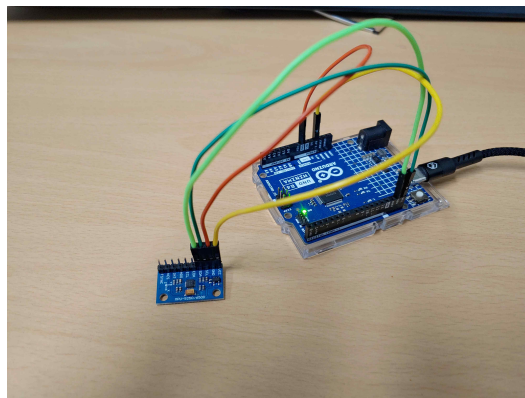


센서는 위와 같이 생겼습니다.

이후 예정은 해당 센서는 초기에 노이즈가 많기 때문에 필터 작업을 거쳐서 3d 영상으로 부드러운 움직임을 구현할 예정입니다.

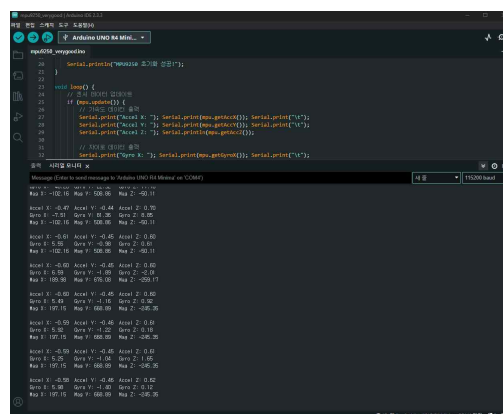
10월 29일 진도 보고서

지난번 IMU센서인 MPU9250을 주문하였고 그게 도착한 뒤로 바로 실험 진행하였습니다 해당 제품은 라즈베리파이가 아니라 ESP32에 장착될 것이기 때문에 기존에 있던 arduino uno r4 minima를 통해 진행하였습니다.



[그림 1] arduino uno r4 minima,
MPU9250

이후 여러 시행착오 끝에 MPU9250의 값을 입력받는데 성공하였고

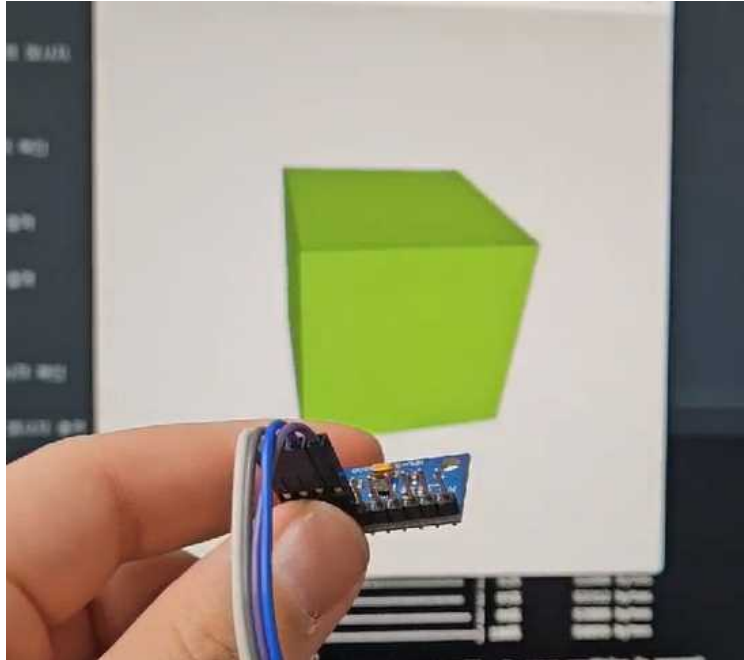


[그림 2]

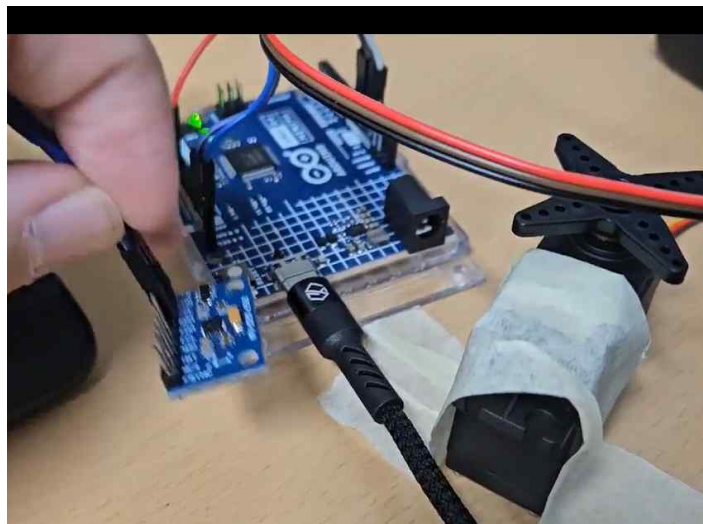
processing 이라는 tool을 이용하여 이를 시각화 하는데 성공하였습니다.

이후에는 필터를 적용하여 수치 안정화를 진행 할 예정입니다.

11월 26일 진도 보고서



imu 센서를 계속 필터링 진행하는 중이고 추가적으로 그 이후에는 모터 제어를 진행하였습니다.



다음과 같이 모터가 동작하는 모습을 보였고 현재는 이를 라즈베리파이로 진행하는 중이지만 다소 어려움이 있습니다

12월 16일 진도 보고서

지난 절차서 및 평가서 이후 코드 통합 작업을 이어 진행 하였습니다.

먼저 ESP32에는 기존 mpu9250 2개의 데이터를 받아오는 코드에 조원의 압력센서의 데이터를 받아오는 코드를 참고하여 추가 하였고 여기에 이중 지수이동평균(EMA)을 적용시켜 센서값을 더 안정적으로 받아오게 하였습니다.

그리고 라즈베리파이는 그대로 해당 값을 서보모터에 적용되게 하였습니다.

최종적으로 ESP32와 라즈베리파이 코드 구성은 다음과 같이 구성하였습니다.

ESP32는 mpu9250의 가속도와 자이로센서를 통해 산정된 roll, pitch, yaw 값을 컴플리멘터리 필터를 통해 수치가 튀는 것을 방지하였고 사전에 값을 -90 ~ 90도로 제한하여 서보모터의 가동 범위내에서 값이 나오도록 한 뒤 이를 라즈베리파이로 전송 하도록 하였고 압력센서는 위와 같이 EMA를 적용시키고 압력센서의 출력값을 1500으로 제한시키고 이를 0 ~ 180도에 적용하고 값을 라즈베리파이로 보내도록 하였습니다.

라즈베리파이는 수신된 값으로 서보모터를 제어할 수 있게 하였는데 우선 첫 번째 mpu9250의 데이터는 모터드라이버의 0, 1, 2채널에 각각 roll, pitch, yaw를 대입하였고 두 번째는 3, 4, 5 압력센서는 6에 대입하여 사용하게 하였습니다

이렇게 만들어진 코드를 실행시켜 로봇팔을 1차적으로 제어를 해본 결과 일부 모터는 동작이 반대로 되었고 초기 각도를 압력센서는 0도 mpu9250은 90도로 맞춰 놓았는데 해당 사항이 원하는 로봇팔의 초기 위치와 맞지 않아서 수정 작업을 진행하였습니다.

로봇팔의 움직임을 몇 차례 확인한 결과 움직임을 반대로 동작하는 것처럼 보이는 이유는 잘못된 초기 각도로부터 발생한 문제로 확인하고 라즈베리파이 코드에 초기 각도 설정을 채널마다 적용되게 바꾼 후 코드를 실행 및 조정을 진행하였습니다.

이후 원하는 초기 각도가 나온 후 다시 로봇 팔 제어를 진행해 보았고 채널 4번의 2번째 mpu9250의 움직임이 잘못된 것을 확인하였고 이를 라즈베리파이의 코드에서 값을 반전시키는 코드를 적용하였습니다.

이후 코드를 적용하였을 때 로봇 팔의 동작은 잘 되었으나 3번 채널에 해당하는 부분의 움직임이 생각했던 동작보다 자연스럽게 움직이지 못하여 이 부분은 8채널로 바꾸어 초기 각도에 고정시켜 움직이지 못하게 하였고 이렇게 까지 진행된 코드로 최종 데모를 진행하였습니다.

최종 진행된 데모는 사람이 직접 들어가기 어렵거나 위험한 장소에서 직관적인 원격 제어를 통해 특정 행동을 수행하는 것이 가능하다는 것을 보여주는 것을 목표로 하였고 이는 다음과 같이 진행하였습니다.

우선 esp32는 전원인가를 위한 선을 제외하고는 라즈베리파이와 외부적으로는 일절 연결되어 있지 않은 원격제어 환경임을 보여주었고 데모 진행자의 팔 위치를 조정 후 esp32를 리셋하여 센서값을 초기화한 후 라즈베리파이 코드를 실행시켜 로봇팔 위치와 진행자의 팔 위치를 비슷하게 한 후 진행자의 팔 위치에 따른 로봇팔의 위치변화를 보여주어 직관적인 제어가 가능하다는 것을 보여주었습니다.

이후 로봇팔을 이리저리 움직여보고 물병이나 작은 과자 봉지 등을 들고 옮기고 놓는 과정을 보여줌으로써 특정 행동을 수행이 가능하다는 점을 보여주며 데모를 마쳤습니다.