We are All the same driver

“우리는 모두 다 같은 운전자 입니다.”

Open CV를 이용한 스마트 휠체어

고종락, 정빈원, 정은지

We are all the same driver

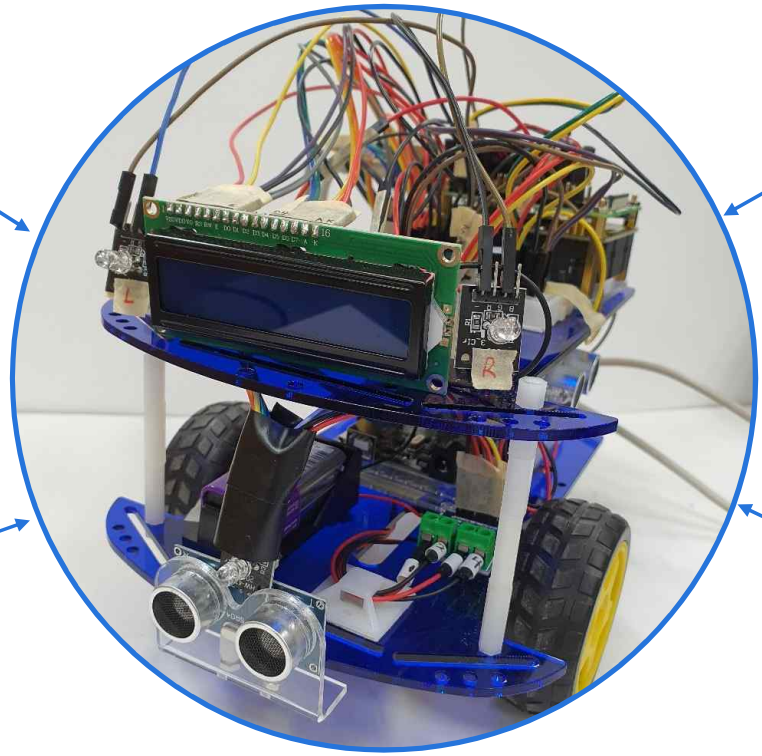
목차

We are all the same driver

01. 제작 개요

Open CV

모터제어



라즈베리파이

아두이노

We are all the same driver

01. 개발 일정

We are all the same driver

02-1. Open CV 이론 - ROI

ROI (Region Of Interest)

- 원본 이미지나 영상에서 원하는 영역을 잘라낼 때 사용
- 잘려진 부분을 관심 영역이라고 한다.
- 처리 할 때 정확성과 처리속도를 높여준다.

- 영상에서 눈과 손만 검출하기 위해 사용하였다.

```
import cv2                                  # Open CV  모듈을 import
import numpy as np                          # np라는 이름으로 numpy 사용

cap = cv2.VideoCapture(0)                   # cap이란 이름으로 실시간 영상 촬영

while True:
    ret, frame = cap.read()                 # cap.read()는 재생되는 비디오의 한 프레임씩 읽음
                                            # 읽은 프레임은 frame

    if ret is False:                        # ret이 없으면, 즉 만약에 영상이 촬영되지 않으면
        break                               # 루프를 나옴

    roi = frame[269: 795, 537: 1416]        # 원본 영상에서 원하는 영역을 설정

    cv2.imshow("Roi", roi)                  # 원하는 영역의 이미지 출력

Cv2.destroyAllWindows()                     # 모든 창 닫음
```

실행 결과

We are all the same driver

02-2. Open CV 이론 - resize()

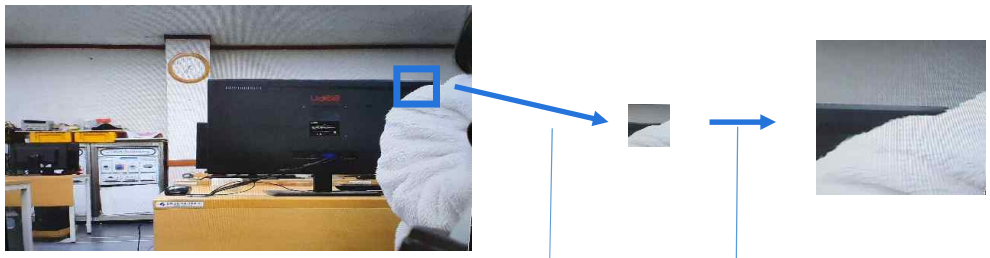
Resize 함수

- 원본 이미지나 영상의 크기를 변환해준다.

- **cv2.resize(roi, (4*width, 4*height), interpolation = cv2.INTER_LINEAR)**
  roi: resizing을 위해 불러올 이미지
  (4*width, 4*height): 원본 이미지에서 4배를 확대
  interpolation = cv2.INTER_LINEAR : 리사이징 시 적용할 interpolation 방법
  Inter_Linear 방식을 사용하였다.

- 영상에서 눈 영역을 확대하기 위해 사용하였다.

실행 결과



ROI로 자름　　ROI로 자른 이미지를 4배 확대

```
import cv2                                  # Open CV  모듈을 import
import numpy as np                          # np라는 이름으로 numpy 사용

cap = cv2.VideoCapture(0)                   # cap이란 이름으로 실시간 영상 촬영

while True:
    ret, frame = cap.read()                 # cap.read()는 재생되는 비디오의 한 프레임씩 읽음
                                            # 읽은 프레임은 frame

    if ret is False:                        # ret이 없으면, 즉 만약에 영상이 촬영되지 않으면
        break                               # 루프를 나옴

    roi = frame[200: 400, 1050: 1200]       # 원본 영상에서 원하는 영역을 설정

    height, width = roi.shape[:2]

    img_result = cv2.resize(roi, (4*width, 4*height), interpolation = cv2.INTER_LINEAR )

    cv2.imshow("x4 INTER_LINEAR", img_result)

    cv2.imshow("Roi", roi)                  # 원하는 영역의 이미지 출력

Cv2.destroyAllWindows()                     # 모든 창 닫음
```

We are all the same driver

02-3. Open CV 이론 - cvtColor(), GaussianBlur()

cvtColor 함수

- 기본 BGR의 색상를 변경해주는 함수
- **cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)**
  img: 처리를 위해 불러올 이미지, 영상
  cv2.COLOR_BGR2GRAY: color 이미지에서 흑백으로 변환

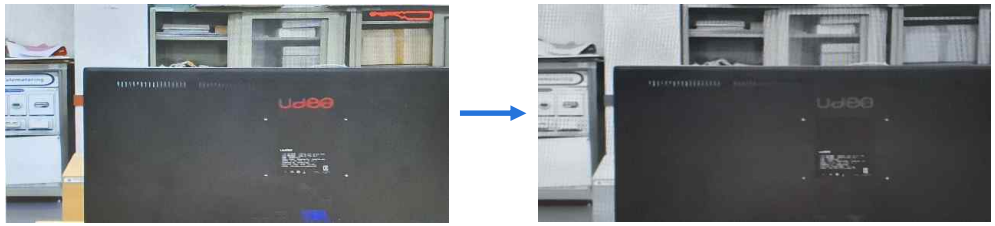
- 눈과 손을 컬러로 볼 필요가 없으며, 흑백 이미지 에서는 처리속도가 더 빨라 지기 때문에 적용해 주었다.

GaussianBlur 함수

- Blur 효과를 주는 함수
- **cv2.GaussianBlur(img, (val, val), 0)**
  img: 처리를 위해 불러올 이미지, 영상
  (val, val): blur 강도 조절량. 강도를 높이면 이미지는 부드러워 지지만 인식률이 떨어짐

- 노이즈 제거를 위해 샤프한 이미지에 blur 효과를 주었다.

실행 결과



```
import cv2                                  # Open CV  모듈을 import
import numpy as np                          # np라는 이름으로 numpy 사용

cap = cv2.VideoCapture(0)                   # cap이란 이름으로 실시간 영상 촬영

while True:
    ret, frame = cap.read()                 # cap.read()는 재생되는 비디오의 한 프레임씩 읽음
                                            # 읽은 프레임은 frame

    if ret is False:                        # ret이 없으면, 즉 만약에 영상이 촬영되지 않으면
        break                               # 루프를 나옴

    roi = frame[250: 500, 900: 1300]        # 원본 영상에서 원하는 영역을 설정

    gray_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)  # roi 를 흑백으로 변환

    gray_roi = cv2.GaussianBlur(gray_roi, (7, 7), 0)  # 흑백 변환된 이미지를 GaussianBlur 적용


    cv2.imshow("Roi", roi)                  # 원하는 영역의 이미지 출력
    cv2.imshow("gray roi", gray_roi)        # 흑백 이미지 출력

Cv2.destroyAllWindows()                     # 모든 창 닫음
```

We are all the same driver

02-4. Open CV 이론 - Threshold()

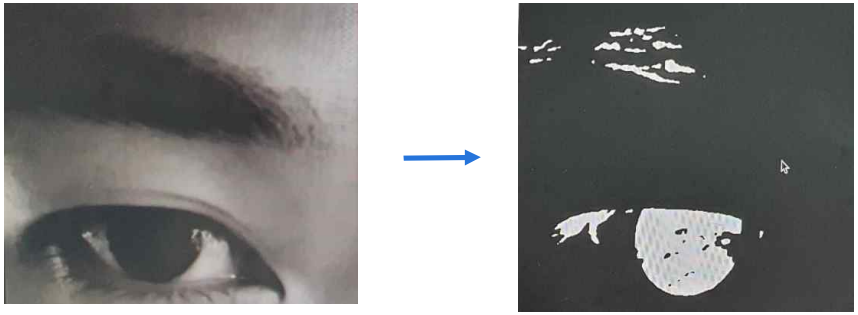
Threshold (문턱)

- Thresholding Value (문턱값) : 이미지 픽셀값이 Thresholding Value 보다 크면 어떤 고정된 값으로 할당하고, 작으면 다른 고정된 값으로 할당
- Thresholding 적용 하려면 grayscale 이미지로 변환하여 적용

- **cv2.threshold(img, threshold_value, value, flag)**
  img: 처리를 위해 불러올 Grayscale 이미지, 영상
  threshold_value: 픽셀 Thresholding Value (문턱값)
  Value: 픽셀 Thresholding Value 보다 클 때 적용되는 최대값
  (적용되는 플래그에 따라 픽셀 문턱값보다 작을 때 적용되는 최대값)
  flag: Thresholding Value 적용 방법 또는 스타일
  cv2.THRESH_BINARY_INV: 픽셀 값이 threshold_value 보다 크면 0, 작으면 value로 할당

- 눈동자, 손 영역에서 흑백만 판별하면 되므로 검은 부분, 흰 부분으로 나누어 이진반전된 이미지를 보여준다.

실행 결과



```
import cv2                                  # Open CV  모듈을 import
import numpy as np                          # np라는 이름으로 numpy 사용

cap = cv2.VideoCapture(0)                   # cap이란 이름으로 실시간 영상 촬영

while True:
    ret, frame = cap.read()                 # cap.read()는 재생되는 비디오의 한 프레임씩 읽음
                                            # 읽은 프레임은 frame

    if ret is False:                        # ret이 없으면, 즉 만약에 영상이 촬영되지 않으면
        break                               # 루프를 나옴

    roi = frame[250: 500, 900: 1300]        # 원본 영상에서 원하는 영역을 설정

    gray_roi = cv2.GaussianBlur(gray_roi, (7, 7), 0)

    gray_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)  # roi 를 흑백으로 변환

    _, threshold = cv2.threshold(gray_roi, 33, 255, cv2.THRESH_BINARY_INV)

    cv2.imshow("Roi", roi)                  # 원하는 영역의 이미지 출력
    cv2.imshow("gray roi", gray_roi)        # 흑백 이미지 출력
    cv2.imshow("Threshold", threshold)      # Thresholding  적용된 이미지 출력


Cv2.destroyAllWindows()                     # 모든 창 닫음
```

We are all the same driver

02-5. Open CV 이론 - Contour

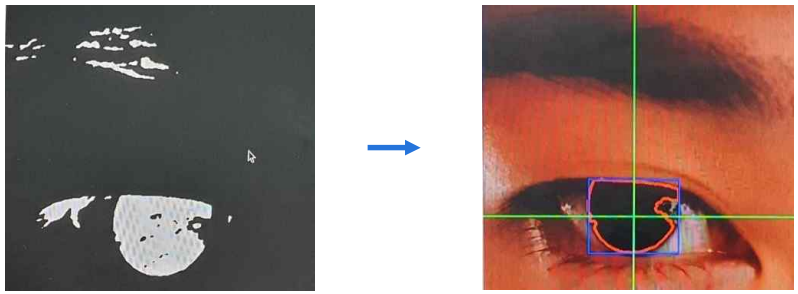
Contour

- 이미지 Contour: 동일한 색 또는 동일한 색상강도를 가진 부분의 가장자리를 연결
- **cv2.findContours(thr, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)**
  thr: 불러올 이미지. Thresholding 통해 변환된 바이너리 이미지 여야 한다.
  cv2.RETR_TREE: 이미지에서 모든 contour를 추출하고 Contour들간의 상관 관계를 추출
  cv2.CHAIN_APPROX_SIMPLE: contour의 수평, 수직, 대각선 방향의 점은 모두 버리고 끝 점만 남겨둠.

- **cv2.drawContours(roi, [cnt], -1, (0, 0, 255), 3)**
  roi: contour를 나타낼 이미지
  [cnt]: 이미지에 그릴 contour
  -1: 인덱스 parameter. 이 값이 음수이면 모든 contour를 그림
  (0, 0, 255): contour 선의 BGR 색상값
  3: contour 선의 두께

- 눈동자와 손의 중앙부를 판별해 화면에 보여주기 위해 적용해 주었다.

실행 결과



```
#contours 찾기
_, contours, _ = cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

#contours를 내림차순으로 정렬
contours = sorted(contours, key=lambda x: cv2.contourArea(x), reverse=True)

# contours를 한번만 출력(가장 큰 부분(=눈동자)만 출력) 후 for문을 나옴
for cnt in contours:
    (x, y, w, h) = cv2.boundingRect(cnt)

    cv2.drawContours(roi, [cnt], -1, (0, 0, 255), 3)          # contour 외곽선 그림

    cv2.rectangle(roi, (x, y), (x + w, y + h), (255, 0, 0), 2)          # contour 주변으로 사각형을 그림

    #눈동자 중앙에 십자 선을 그림
    cv2.line(roi, (x + int(w/2), 0), (x + int(w/2), rows), (0, 255, 0), 2)
    cv2.line(roi, (0, y + int(h/2)), (cols, y + int(h/2)), (0, 255, 0), 2)
    break
```

We are all the same driver

03. 사용 부품 (메인부)

아두이노 메가

- Microcontroller: ATmega2560
- 작동 전압: 5V
- 입력 전압(권장): 7-12V
- 입력 전압(한계): 6-20V
- 디지털 입출력 핀: 54개 (15개 PWM 출력)
- 아날로그 입력 핀: 16개
- 입출력 핀 당 DC 전류: 20 mA
- 3.3V 핀 DC전류: 50 mA
- 플래시 메모리: 256 KB (이 중 8 KB는 부트로더가 사용)
- SRAM: 8 KB
- EEPROM: 4 KB
- Clock Speed: 16 MHz
- LED_BUILTIN: 13
- 길이: 101.52 mm, 넓이: 53.3 mm
- 중량: 37 g

라즈베리파이 3

- Broadcom BCM2837B0, 1.4GHz Cortex-A53 64비트 SoC
- 이중 대역 802.11ac 무선 LAN
- Bluetooth 4.2
- 메모리: 1GB LPDDR2 SDRAM
- 전원 공급 장치: 5V/2.5A DC 전원 입력(마이크로 USB)
- 확장 40핀 GPIO(Genere Purpose Input Output) 헤더
- 풀 사이즈 HDMI 비디오 출력
- 4극 스테레오 오디오 출력 및 컴포지트 비디오 포트
- Raspberry Pi 카메라 연결용 CSI(카메라 직렬 인터페이스) 카메라 포트
- Raspberry Pi 터치스크린 디스플레이 연결용 DSI(디스플레이 직렬 인터페이스) 디스플레이 포트
- 운영 체제 로드 및 데이터 저장용 마이크로 SD 포트
- 작동 온도 범위: 0~50°C
- 크기: 120mm x 75mm x 34mm
- 중량: 75g

We are all the same driver

03. 사용 부품 (센서부)

초음파 거리 센서 (HC-SR04)

- 동작 전압: Digital 5V
- 입력 전압: 5V DC
- 대기 상태에서의 전류: <2mA
- 유효 측정 각도: <15도
- 유효 측정 거리: 20 ~ 5000 mm
- 측정 해상도: 3mm
- 무게: 15g
- 크기: 2.0 x 4.3 x 1.5cm

- 동작원리: 모듈에 DC5V 전원을 인가한 후, Trig 핀을 통해 10us 의 펄스를 인가하면 초음파 센서는 8개의 40kHz 펄스를 발생시키고, 측정된 거리에 따라 150us ~ 25ms의 펄스를 Echo 핀을 통해 출력시킨다.
- 초음파 센서 앞에 방해물이 없다면 38ms의 펄스를 Echo 핀을 통해 발생시킨다.

웹캠 (Logitech C270 HD WEBCAM)

- 최대 해상도: 720p/30fps
- 포커스 타입: 고정 포커스
- 렌즈 타입: plastic
- Diagonal field of view (dFoV): 60°

We are all the same driver

03. 사용 부품 (아두이노 출력부)

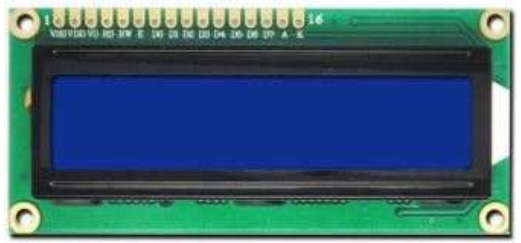
RGB LED module (SZH-EK058)

- 동작 전압: 5V
- LED drive mode: common cathode

Piezo buzzer

- 동작 전압: 5V
- 피에조를 이용해 소리를 내는 작은 스피커
- 특정 방향으로 압력을 가하면 결정체 표면에서 전기가 발생하는 성질을 이용한 것. 얇은 판을 붙여 미세한 떨림으로부터 소리가 나게 된다.

I2C 16x2 문자 LCD

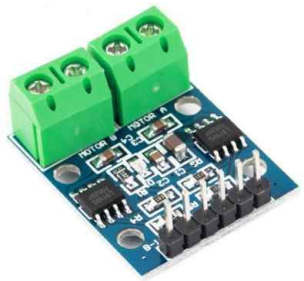


- 정격전압 : 5V
- 출력범위 : 16문자 2열
- 문자 : White
- 백라이트 : Blue
- 인터페이스 : 데이터버스
- 크기 : 80mm×36mm

We are all the same driver

03. 사용 부품 (구동부)

L9110s 모터 드라이버 모듈 (SZH-MDBL-002)



- 공급 전압: 2.5-12V DC
- 2개의 각자 분리된 L9110 motor control chip을 가짐
- 각자 채널은 계속된 800 ma의 출력을 낼 수 있음
- 모터 스피드 조절을 위해서는 PWM 시그널 사용
- 모터 방향을 바꾸기 위해서는 디지털 출력 사용
- PCB Size: 29.2mm x 23mm

기어박스장착모터 (NP01S-220)

(DC모터)

- 동작 전압: 3 – 6V
- 기어 비율: 1:220
  - LOAD 없을 시
    - Speed: 50r/min
    - 전류: 0.25A
  - 최고 효율 시
    - Speed: 34r/min
    - 전류: 0.25A
    - Torque: 0.71Kg.cm
    - 출력: 0.52w
  - Stall
    - 전류: 1.1A
    - Torque: 2.4Kg.cm

We are all the same driver

04-1. 개념설계 - Block diagram

센서부

초음파센서

장애물 감지

센서부

웹캠

화면 입력 (눈, 손)

메인부

아두이노 메가

라즈베리파이 3

입력에 대한 출력부 및 구동부 제어

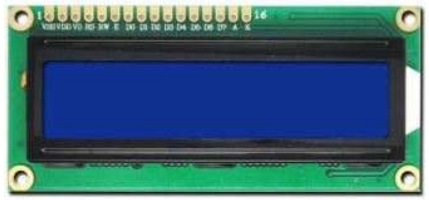
출력부

LED

거리에 따른 LED점등

Buzzer

경고음 발생



LCD

사방 장애물까지의 거리를 표시

모니터

화면 출력

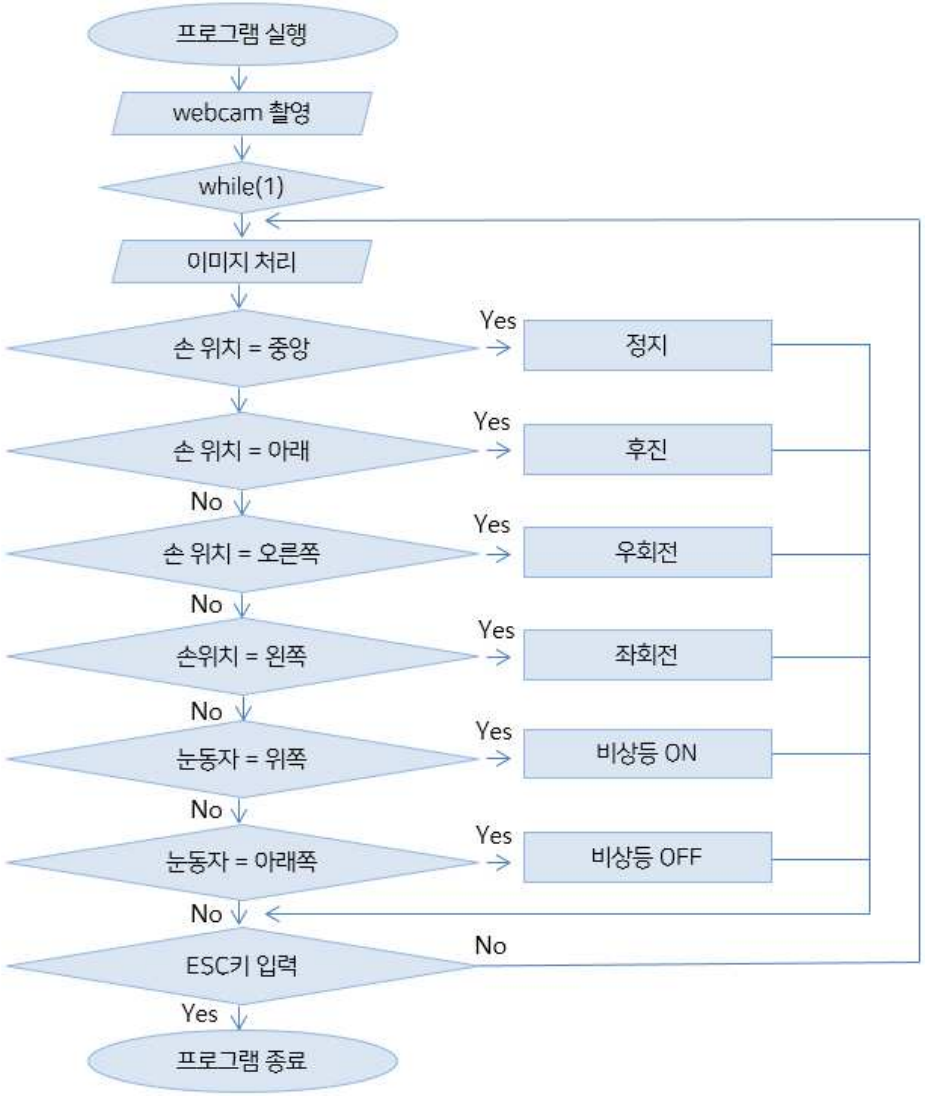
구동부

DC 모터

휠체어 동작 출력 발생

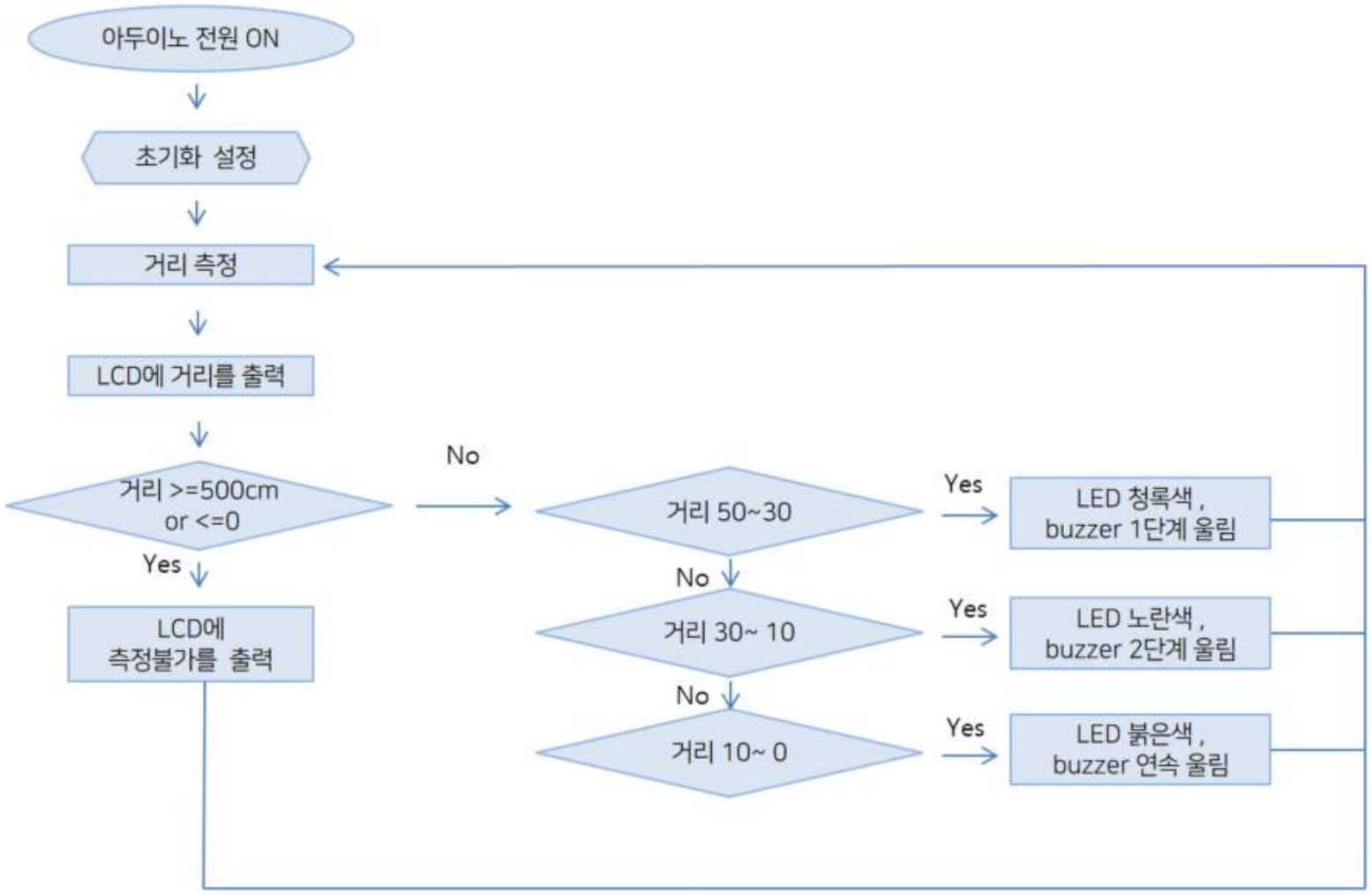
We are all the same driver

04-1. 개념설계 - 알고리즘 (라즈베리파이)



We are all the same driver

04-1. 개념설계 - 알고리즘 (아두이노)

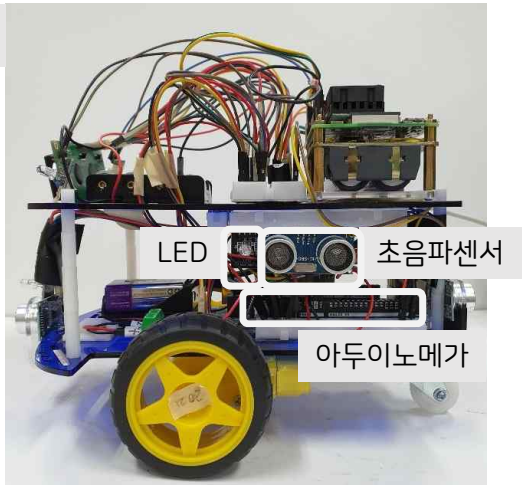


We are all the same driver

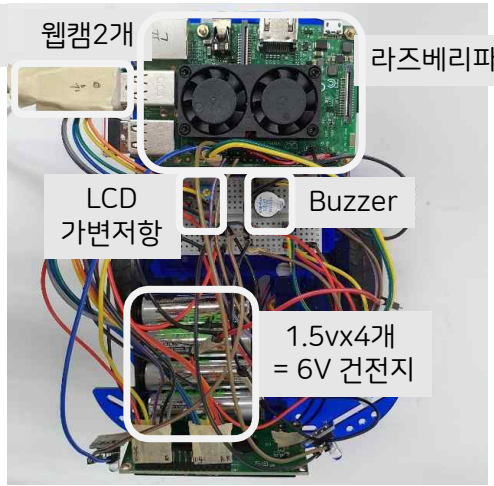
04-2. 상세설계 - 작품사진 (스마트 휠체어)

전

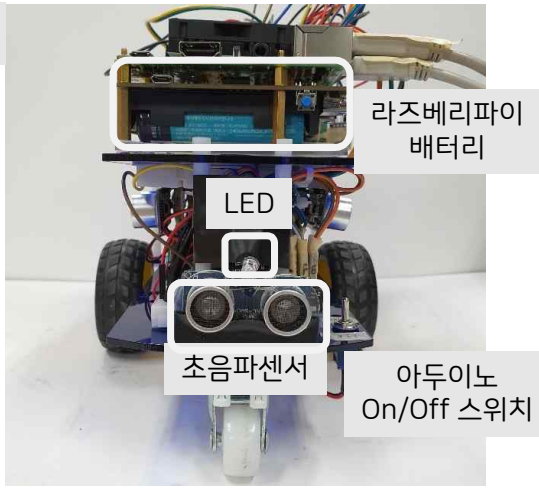
좌



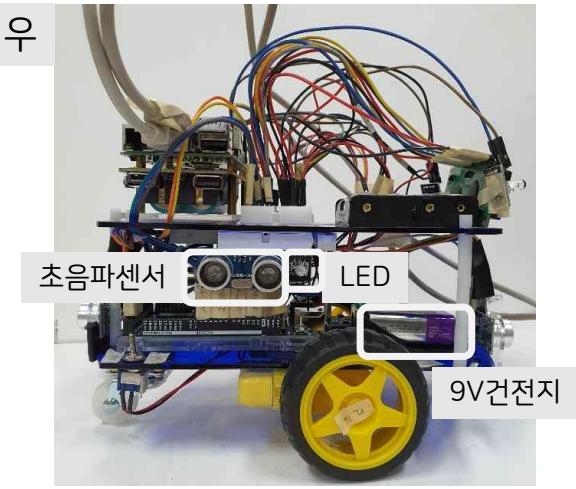
위



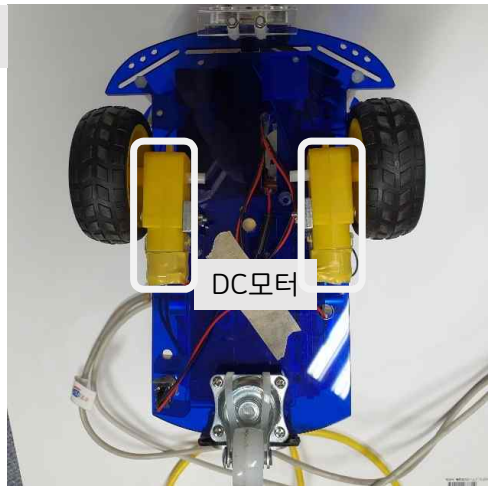
후



우



아래

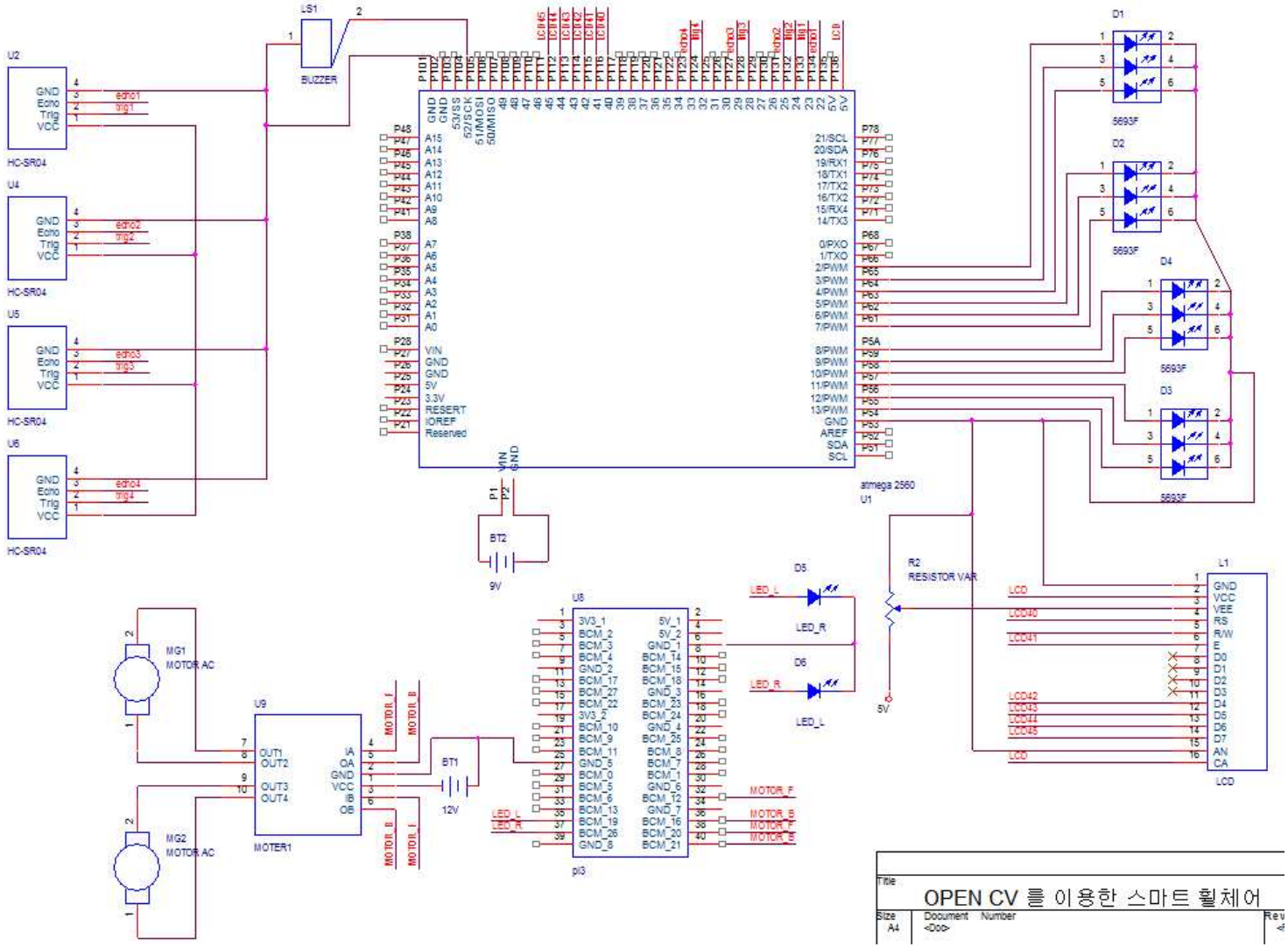


We are all the same driver

04-2. 상세설계 - 작품사진 (모니터, 웹캠)

We are all the same driver

04-2. 상세설계 - Circuit Diagram



We are all the same driver

04-3. 동작 설명

```
if 50 < ax < 650 and 200 <= ay <550:
    print("stop ")
    dc_motorL.stop()
    dc_motorR.stop()

  elif 50 < ax < 650 and 0 <= ay < 200:
    print("FORWARD")
    dc_motorR.forward(speed=0.7)
    dc_motorL.forward(speed=0.7)

  elif 550 <= ay <= 700 and ax >= 0:
    print("BACKWARD")
    dc_motorL.backward(speed=0.7)
    dc_motorR.backward(speed=0.7)

  elif ax <= 50 and ay < 550:
    print("right")
    dc_motorR.forward(speed=1.0)
    dc_motorL.backward(speed=0.1)

  elif 650 <= ax and ay < 550:
    print("left")
    dc_motorR.backward(speed=0.1)
    dc_motorL.forward(speed=1.0)
```

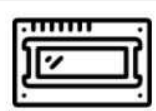
```
if y <= 20: # 위쪽을 볼 때 비상등이 켜짐
    print ("both LED on")
    red_ledR.on()
    red_ledL.on()

  if y >=220 : #아래쪽을 볼 때 led가 꺼짐
    print ("LED off")
    red_ledR.off()
    red_ledL.off()
```

We are all the same driver

04-3. 동작 설명

```
digitalWrite(trigPin1, LOW); //초음파 센서를 초기화
delayMicroseconds(2);
digitalWrite(trigPin1, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin1, LOW);
duration1 = pulseIn(echoPin1, HIGH); // 트리거 핀에서 나온 펄스를 받아서
distance1= duration1*0.034/2; // 거리를 측정
```

```
void sound1() {
 unsigned long currentMillis = millis();
 if(currentMillis - previousMillis >= delayTime){ //0.5초 시간이 흘렀는지 체크
  previousMillis = currentMillis; //0.5초가 지나 참임으로 0.5초 지난 현재시간을 이전시간에 저장
  goState=!goState;   //if문이 참이니깐 0.5초 단위로 ledState 값을 반전시키면 0.5초 단위로 참/거짓
  digitalWrite(go, goState);   //참(5V) or 거짓(0V)
  }
}
void sound2() {
 unsigned long currentMillis = millis();
 if(currentMillis - previousMillis >= delayTime1){
   previousMillis = currentMillis;
   goState=!goState;
   digitalWrite(go, goState);
  }
}
void sound3() {
 analogWrite(go,255);
}
```

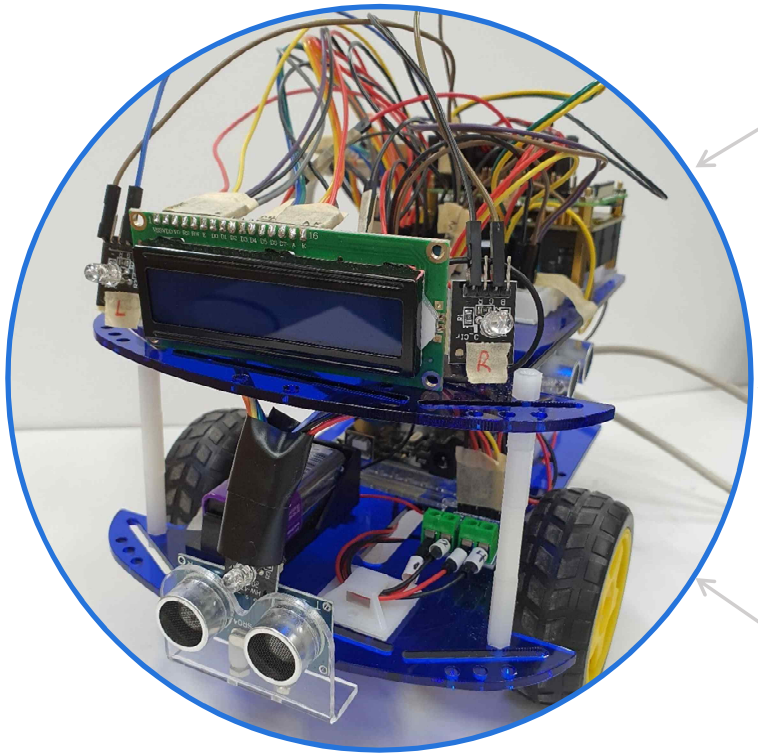


```
void lcdB(){ //후방거리 lcd에출력
 if (distance1 >=100){
  lcd.setCursor(10, 0);
  lcd.print("B:Safe");
 }
 else if ((distance1
<100)&&(distance1>=10) ){
  lcd.setCursor(10, 0);
  lcd.print("B:");
  lcd.setCursor(12, 0);
  lcd.print(distance1);
  lcd.setCursor(14, 0);
  lcd.print("cm");
 }
 else if (distance1<10) {
  lcd.setCursor(10, 0);
  lcd.print("B: ");
  lcd.setCursor(13, 0);
  lcd.print(distance1);
  lcd.setCursor(14, 0);
  lcd.print("cm");
 }
}
```

```
if (distance1 >= 500 || distance1 <= 0){
  Serial.println("1 Back sensor Out of range");
  noTone(go); //부저 소리 끔
  digitalWrite(red1, LOW);
  digitalWrite(grn1, LOW);
  digitalWrite(blu1, LOW);
 }
 else  {
  Serial.print ( "Sensor1 : "); //센서 1에
  Serial.print ( distance1);// 거리 값
  Serial.println("cm"); // cm를 출력
  if ((distance1 <=50)&&(distance1>30) ) {
   sound1();
   digitalWrite(red1, LOW);
   digitalWrite(grn1, HIGH);
   digitalWrite(blu1, HIGH);
  }
  else if ((distance1 <=30)&&(distance1>10)){
   sound2();
   digitalWrite(red1, HIGH);
   digitalWrite(grn1, HIGH);
   digitalWrite(blu1, LOW);
  }
  else if ((distance1 <=10)&&(distance1>0)) {
   sound3();
   digitalWrite(red1, HIGH);
   digitalWrite(grn1, LOW);
   digitalWrite(blu1, LOW);
  }
 }
```

We are all the same driver

05. 오류 개선



1. 영상 끊김 현상

웹캠을 하나만 연결 하였을 때는 영상이 끊김이 없었지만 두개를 연결하니 영상이 끊겼다.
GPU 메모리를 여러 값으로 테스트를 해본결과 GPU값 500에서 영상도 끊기지 않고 라즈베리파이 실행속도도 적당하였다.

2. 모터 제어

테스트를 해본 결과 코드의 수치를 변경하는 것보다 하드웨어적으로 전압을 조절 해서 모터의 출력을 조절하는 것이 더 효율적이라는 판단이 들었다.
1.5V를 4개 병렬로 연결하여 6의 출력을 주니 적당한 출력이 나왔다.

3. 컨셉 변경

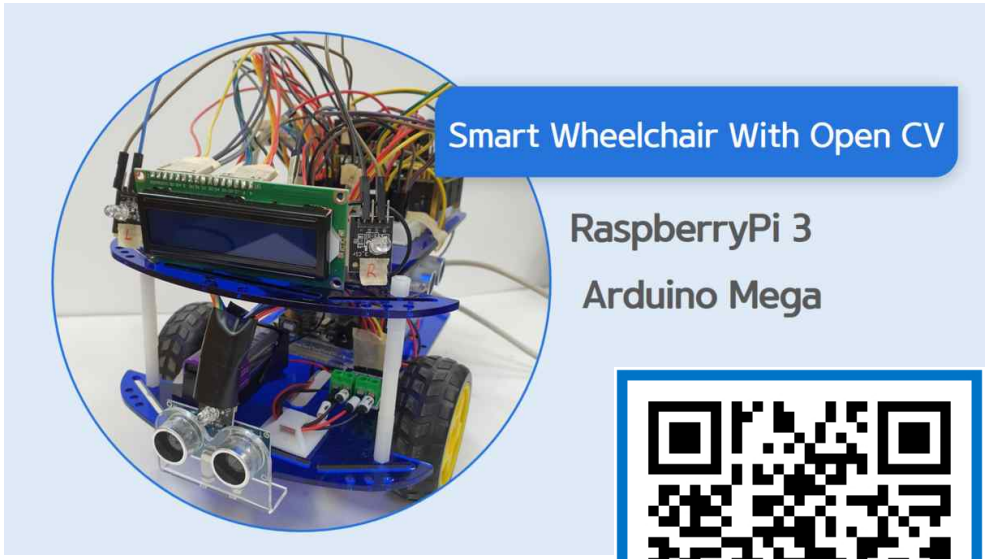
눈으로만 운전을 한다는 것이 테스트와 함께 병행하려니 쉽지가 않았다.
손으로 메인 운전을 하고 눈으로는 운전 보조를 하는 기능을 넣어주는 것으로 변경하였다.

We are all the same driver

06. 결과 및 고찰

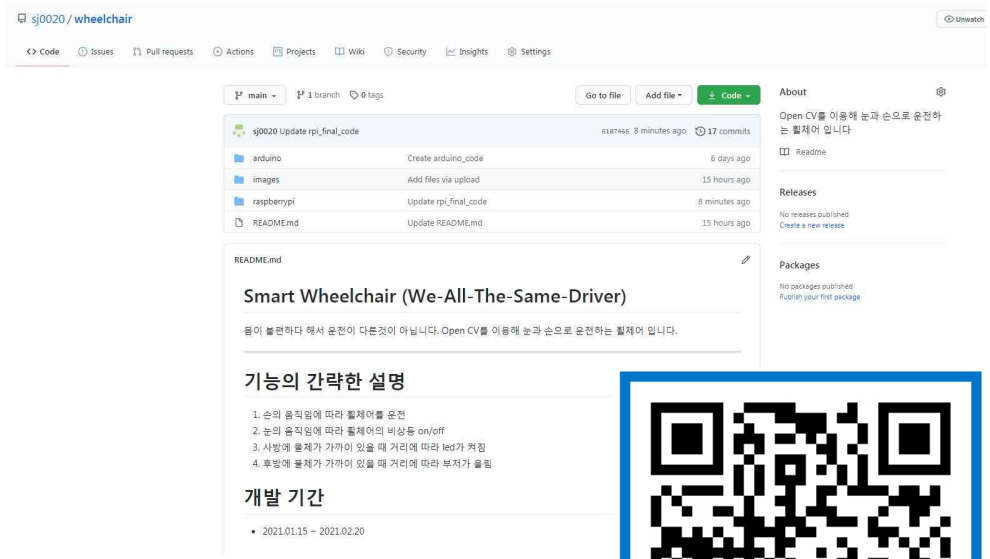
동작 영상

- https://youtu.be/NodGg3xb4tI



Github

- https://github.com/sj0020/wheelchair



06. 논의 및 고찰

영상 처리 속도 문제

- 영상처리 속도가 느린것을 GPU 값을 조절하는 것으로 해결은 했지만, 라즈베리파이는 영상 처리만 하고 아두이노가 모터 제어까지 하는 크로스 플랫폼 방식으로 했다면 영상처리가 조금 더 매끄럽지 않았을까 하는 생각이 든다. 다만 이렇게 했을 때에 아두이노가 초음파센서, LED, LCD, 부저, 모터까지 처리해야 하므로 이 점은 고려해야 할 것이다.

Eye-Tracker

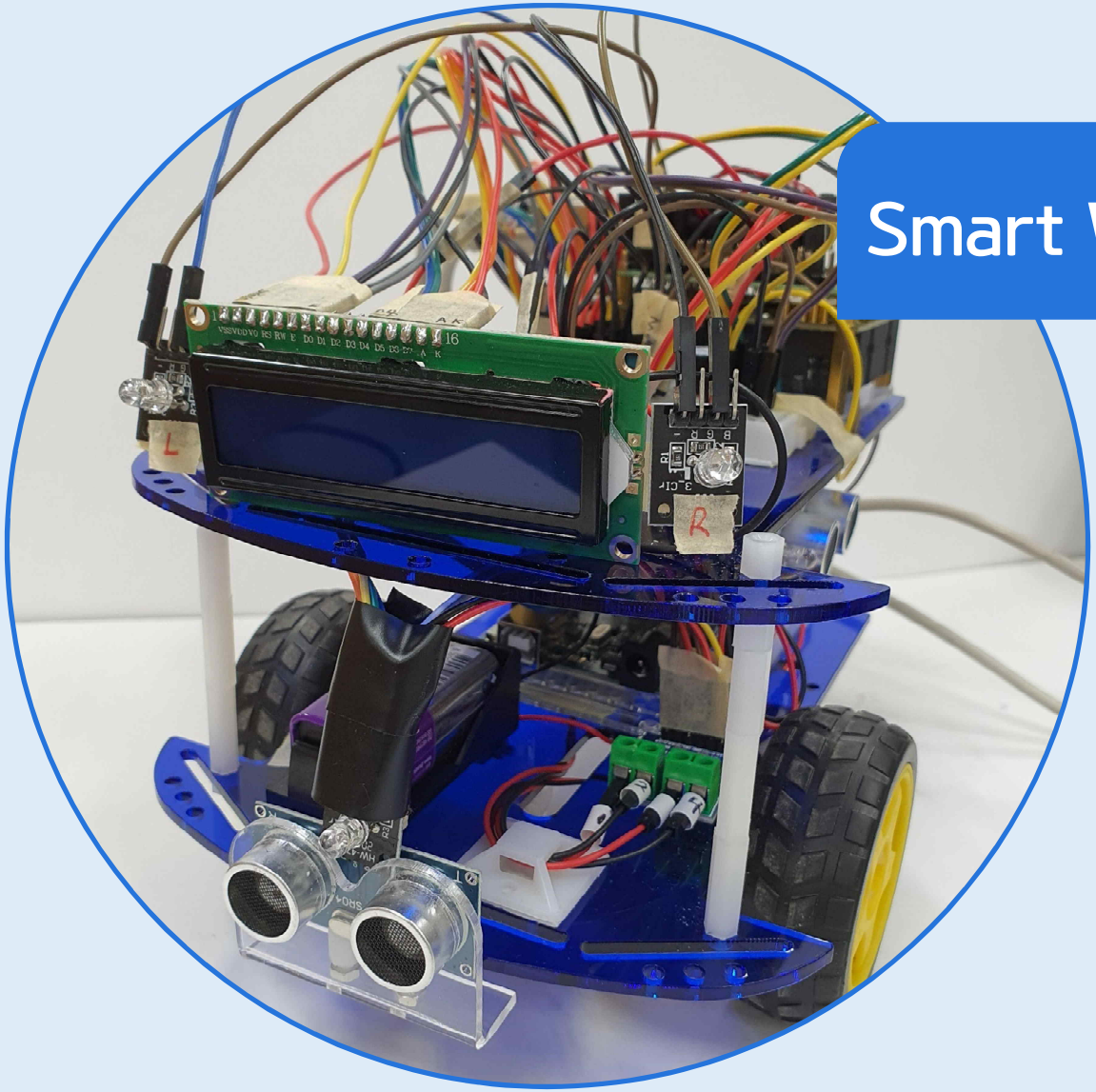
- 처음 프로젝트 아이디어를 계획 할 때는 아이트래커를 사용해 눈 인식을 하는 것이었지만, 비용 문제로 웹캠을 사용하게 되었다. 만약 아이트래커를 사용했다면 좀 더 상세한 운전과 기능을 구현할 수 있었을 것이다.

휠체어 구현의 한계

- 사람이 탈 휠체어를 구현할 수 있는 상황이 아니라서 차를 작게 만들고 웹캠 선을 길게 연결하여 책상에서 운전하는 방식으로 구현을 하였다.

휠체어 무게 문제

- 라즈베리파이, 라즈베리파이 배터리 및 아두이노메가 등 무게가 휠체어 뒷쪽에 쏠려있다. 설계 시 무게를 앞쪽과 뒷쪽에 더 균등하게 분배 하였으면 움직임이 더 유연할 것이다.



Smart Wheelchair With Open CV

RaspberryPi 3

Arduino Mega