명지대학교
MYONGJI UNIVERSITY

캡스톤 디자인1 최종 보고서

팀명: AUTOPILOT

60191924 서동주

60191944 이승준

목차

1.프로젝트 동기 및 공헌

프로젝트 동기

휠체어 사용자들의 안전과 편의를 고려한 자율주행 휠체어의 개발은 현재 매우 중요한 사회적 과제입니다. 많은 사람들이 신체적 제약으로 인해 휠체어를 필요로 하고 있으며, 이들이 더 독립적이고 안전하게 이동할 수 있는 기회가 필요합니다. 더불어, 몸이 불편한 분들이 보호자의 도움 없이도 원하는 장소를 자유롭게 방문할 수 있도록 하는 것이 저희의 목표입니다. 이러한 동기로, 우리는 자율주행 기술을 활용하여 휠체어 사용자들의 이동 시 불편을 개선하고, 그들의 삶의 질을 향상시키고자 합니다.

프로젝트 공헌

저희는 새로운 기술과 알고리즘을 적용하여 자율주행 휠체어의 성능을 향상시킬 것입니다. 카메라 센서와 Mobilenet SSD 모델을 활용하여 장애물을 정확하게 탐지하고 피하는 시스템을 개발함으로써 사용자의 안전을 보장할 것입니다. 또한, 저희는 블루투스 모듈을 통해 사용자가 필요할 때 언제든지 휠체어를 직접 조종할 수 있는 기능을 구현하여 긴급 상황에 대비하고, 사용자의 안전을 최우선으로 고려할 것입니다.

이러한 과제로 인해 많은 휠체어 사용자들이 일상적인 이동에 어려움을 겪고 있는데, 자율주행 휠체어는 이러한 어려움을 극복하고 더 많은 자유로운 활동을 할 수 있게 도와줍니다. 또한, 이 기술은 보호자나 도우미의 필요성을 줄여주어 휠체어 사용자들에게 더 많은 자립성을 부여합니다.

뿐만 아니라, 자율주행 기술을 휠체어에 적용함으로써 휠체어 사용자들 뿐만 아니라 사회 전반에 기술 발전과 혁신을 촉진하여 이로운 영향을 미칠 수 있습니다. 특히, 사람들의 혼잡도에 따라 경로를 다르게 설정하여 출발지에서 목적지까지 최적의 경로로 이동할 수 있게 하는 기능을 추가하여 사용자 경험을 더욱 개선할 것입니다.

2.서문

현대 사회에서 휠체어는 신체적 제약으로 인해 이동에 어려움을 겪는 많은 사람들에게 필수적인 도구입니다. 그러나 휠체어를 사용하는 사람들은 종종 독립적인 이동에 어려움을 겪으며, 이는 그들의 삶의 질을 저하시킬 수 있습니다. 특히, 예기치 못한 장애물이나 복잡한 경로에서의 이동은 사용자의 안전과 편의를 위협할 수 있습니다.

이 프로젝트의 주된 목적은 자율주행 기술을 활용하여 휠체어 사용자들이 더욱 독립적이고 안전하게 이동할 수 있도록 지원하는 것입니다. 이를 위해 우리는 최신 기술과 알고리즘을 적용하여 장애물을 인식하고 회피하며, 사용자가 원하는 경로를 따라 안전하게 이동할 수 있는 시스템을 구현하고자 하였습니다. 자율주행 휠체어의 개발은 단순한 기술 혁신을 넘어서, 이동의 자유를 제공함으로써 사용자의 삶의 질을 크게 향상시킬 것으로 기대됩니다. 또한, 이 기술은 보호자나 도우미의 필요성을 줄여주어 휠체어 사용자들에게 더 많은 자립성을 부여합니다.

보고서는 다음과 같은 구조로 구성되어 있습니다. 먼저 프로젝트의 동기 및 공헌을 설명한 후, 과제의 요약과 관련된 사전 기술 및 연구를 살펴봅니다. 이어서 시스템 모델과 제안하는 기술 및 이론을 설명하고, 프로덕트 실험 설계 내용과 실험 결과를 제시합니다. 마지막으로 결론을 통해 프로젝트의 성과와 앞으로의 과제를 정리합니다.

3. 과제요약

신체적 제약을 가진 사용자들의 이동 편의성과 안전성을 개선하기 위해 자율주행 휠체어를 개발한 프로젝트를 다룹니다. 프로젝트의 주요 목표는 자율주행 기술을 적용하여 휠체어 사용자가 더욱 독립적이고 안전하게 이동할 수 있도록 돕는 것입니다. 이를 위해 다음과 같은 주요 기술과 방법론을 사용했습니다:

1. **개발 배경 및 환경**:
    - 자율주행 휠체어의 필요성 및 중요성을 기반으로 프로젝트가 시작되었습니다.
    - 개발 환경은 라즈베리파이, Mobilenet SSD 모델, 블루투스 모듈 등을 포함한 최신 하드웨어 및 소프트웨어를 사용하였습니다.
2. **시스템 모델**:
    - 카메라 센서를 사용하여 주행 경로를 실시간으로 인식하고, Mobilenet SSD 모델을 통해 장애물을 탐지하는 시스템을 설계했습니다.
    - 다익스트라 알고리즘을 활용하여 최단 경로를 계산하고, 블루투스 모듈을 통해 사용자가 스마트폰으로 휠체어를 제어할 수 있도록 구현하였습니다.
3. **실험 설계 및 검증**:
    - 주행 경로 인식, 장애물 탐지 및 회피, 경로 설정 알고리즘 검증, 블루투스 제어 기능 검증 등 다양한 실험을 통해 시스템의 성능을 평가했습니다.
    - 실험 결과, 휠체어가 설정된 경로를 따라 정확하게 이동하고, 장애물을 효과적으로 회피하며, 사용자에게 안전하고 편리한 제어 기능을 제공함을 확인했습니다.
4. **한계점 및 발전 가능성**:
    - 현재 시스템의 한계점으로는 복잡한 환경에서의 장애물 인식 정확도 및 반응 속도의 개선이 필요함을 지적하였습니다.
    - 향후 발전 가능성으로는 더욱 정교한 알고리즘 개발, 다양한 환경에서의 테스트 및 사용자 피드백을 통한 시스템 개선 등이 제시되었습니다.
5. **역할 분담 및 개발일정**
    - 프로젝트 팀의 역할 분담과 각 단계별 개발 일정을 주차별 회의록에 상세히 기록하여 체계적인 프로젝트 진행을 하였습니다.

4. 관련된 사전기술 및 연구

자율주행 휠체어는 다양한 최신 기술과 연구 성과를 종합하여 구현됩니다. 본 프로젝트에 적용된 주요 기술과 연구 현황은 다음과 같습니다:

자율주행 기술

자율주행 기술은 주로 자동차 산업에서 발전해 왔지만, 휠체어와 같은 개인 이동수단에도 응용되고 있습니다. 주요 기술 요소로는 센서 데이터 처리, 경로 계획, 실시간 장애물 회피 등이 있습니다.

- **센서 데이터 처리**: 카메라, 라이다(LiDAR), 레이더 등을 이용한 실시간 환경 인식 기술.
- **경로 계획**: A* 알고리즘, 다익스트라 알고리즘 등을 이용한 최적 경로 탐색.
- **장애물 회피**: 다양한 객체 인식 알고리즘을 통해 실시간으로 장애물을 탐지하고 회피하는 기술.

이미지 및 객체 인식 기술

이미지 및 객체 인식 기술은 자율주행 휠체어의 핵심 요소입니다. Mobilenet SSD 모델과 같은 경량화된 딥러닝 모델이 주로 사용됩니다.

- **Mobilenet SSD**: 효율적인 경량 신경망 모델로, 모바일 및 임베디드 기기에서 실시간 객체 인식을 가능하게 합니다. Mobilenet SSD 는 다양한 환경에서 장애물 및 객체를 신속하고 정확하게 인식합니다.
- **YOLO (You Only Look Once)**: 또 다른 실시간 객체 인식 모델로, 높은 속도와 정확도를 자랑합니다.

경로 계획 알고리즘

경로 계획 알고리즘은 휠체어가 목적지까지 최적의 경로로 이동할 수 있도록 합니다. 대표적인 알고리즘으로 다익스트라 알고리즘이 있습니다.

- **다익스트라 알고리즘**: 그래프 이론에 기반한 최단 경로 탐색 알고리즘으로, 가중치가 있는 그래프에서 최단 경로를 찾는 데 사용됩니다.
- *A 알고리즘**: 휴리스틱을 사용하여 경로 탐색 속도를 높인 알고리즘으로, 실시간 경로 계획에 많이 사용됩니다.

블루투스 및 무선 통신 기술

블루투스 및 무선 통신 기술은 사용자가 스마트폰 등을 통해 휠체어를 원격으로 제어할 수 있도록 합니다.

- **블루투스 모듈**: 간단한 페어링과 저전력 소모로 휠체어와 스마트폰 간의 안정적인 통신을 제공합니다.
- **Wi-Fi 모듈**: 더 넓은 범위의 통신을 필요로 하는 경우, Wi-Fi 를 통해 원격 제어 및 모니터링 기능을 구현할 수 있습니다.

자율주행 휠체어 연구 사례

다양한 연구 기관과 기업에서 자율주행 휠체어에 대한 연구와 개발이 진행되고 있습니다. 일부 주요 연구 사례는 다음과 같습니다:

- **MIT (Massachusetts Institute of Technology)**: MIT 는 자율주행 휠체어 개발을 위해 심층 학습과 강화 학습 알고리즘을 적용하여 복잡한 환경에서도 안전하게 이동할 수 있는 시스템을 연구하고 있습니다.
- **Google X**: Google 의 연구 부문인 X 에서는 자율주행 기술을 휠체어와 같은 소형 이동수단에 적용하기 위한 프로젝트를 진행하고 있습니다.
- **Whill**: 일본의 Whill Inc.는 자율주행 기능을 갖춘 휠체어를 개발하여 상용화 단계에 있습니다. 이 휠체어는 복잡한 실내외 환경에서도 안전하게 이동할 수 있도록 설계되었습니다.

이와 같은 사전기술 및 연구들을 바탕으로 자율주행 휠체어의 기술적 가능성을 탐구하고, 이를 통해 사용자의 이동 편의성을 극대화하고자 하였습니다. 향후 발전 가능성 또한 높으며, 지속적인 연구와 개발을 통해 자율주행 휠체어의 상용화와 보편화가 기대됩니다.

5.시스템 모델

자율주행 휠체어 시스템은 신체적 제약으로 인해 휠체어 조작이 어려운 사용자를 위해 설계되었습니다. 이 시스템은 카메라, 센서, 블루투스 모듈, 라즈베리파이, 아두이노 등의 다양한 하드웨어를 이용하여 휠체어가 자율적으로 경로를 인식하고 장애물을 회피하도록 합니다.

구성 요소

- **라즈베리파이**: 시스템의 중심 제어 장치로서, 이미지 처리, 데이터 관리 및 알고리즘 실행을 담당합니다.
- **아두이노**: 하드웨어 제어에 사용되며, 모터 제어 및 센서 데이터 수집을 담당합니다.
- **카메라 센서**: 주변 환경을 실시간으로 촬영하여 라즈베리파이로 전송하고, 이를 통해 경로 인식 및 장애물 탐지를 수행합니다.
- **블루투스 모듈 (HM-10)**: 사용자가 스마트폰을 통해 휠체어를 직접 조종할 수 있도록 통신을 담당합니다.
- **기타 부품**: 메모리 리더기, SD 메모리, 리튬배터리, 플랫케이블, 광각 카메라, 카메라 연결보드, 카메라케이블, 모터 등.

작동 원리

1. **경로 인식**: 카메라 센서가 촬영한 영상을 라즈베리파이가 실시간으로 분석하여 휠체어가 따라야 할 경로를 인식합니다. 엔비디아의 "End to End Learning for Self-Driving Cars" 논문에서 제안된 CNN 아키텍처를 적용하여 라인을 인식합니다.
2. **장애물 탐지**: Mobilenet SSD 모델을 이용하여 실시간으로 장애물을 탐지하고, 필요한 경우 휠체어를 정지시킵니다.
3. **경로 설정 및 이동**: 다익스트라 알고리즘을 사용하여 최단 경로를 계산하고, 실시간으로 경로를 조정하여 목적지까지의 이동을 보장합니다.
4. **수동 조작**: 블루투스 모듈을 통해 사용자가 필요 시 언제든지 휠체어를 수동으로 조작할 수 있습니다.

인터페이스

- **사용자와 휠체어 간의 인터페이스**: 블루투스 모듈을 통해 스마트폰으로 휠체어를 제어할 수 있습니다.
- **라즈베리파이와 아두이노 간의 인터페이스**: 라즈베리파이에서 처리된 데이터와 명령을 아두이노로 전송하여 모터 및 센서를 제어합니다.
- **카메라와 라즈베리파이 간의 인터페이스**: 카메라가 촬영한 영상을 라즈베리파이로 실시간 전송합니다.

데이터 흐름

1. 카메라가 실시간 영상을 촬영하여 라즈베리파이로 전송.
2. 라즈베리파이가 영상을 분석하여 경로를 인식하고 장애물을 탐지.
3. 라즈베리파이가 아두이노로 명령을 전송.
4. 아두이노가 모터를 제어하여 휠체어의 움직임을 조정.

제약 조건 및 가정

- **환경 제약**: 시스템은 주로 실내 또는 장애물이 적은 평탄한 외부 환경에서 동작하도록 설계되었습니다.
- **전원 제약**: 리튬배터리의 용량에 따라 사용 시간이 제한될 수 있습니다.
- **네트워크 제약**: 블루투스 통신 범위 내에서만 스마트폰으로 조작이 가능합니다.

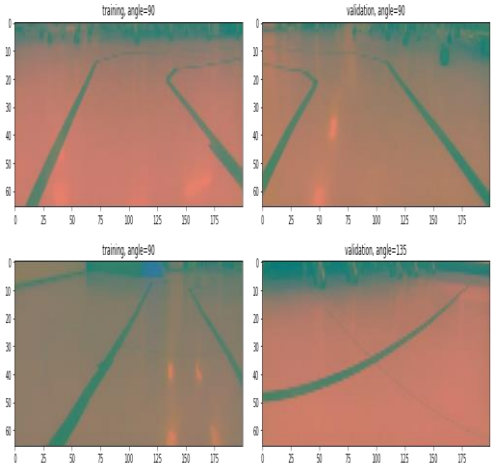
6. 제안하는 기술, 이론

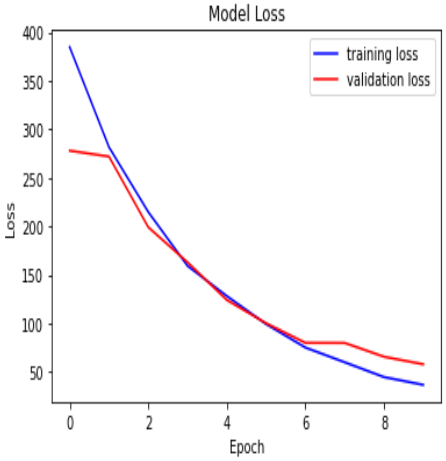
라인 인식을 위해, 엔비디아에서 발표한 "End to End Learning for Self-Driving Cars" 논문에서 제안된 컨볼루션 신경망(CNN) 아키텍처를 적용하여 모델을 구축하였습니다

```
def nvidia_model():
    model = Sequential(name='nvidia_model')
    model.add(Conv2D(24, (5, 5), strides=(2, 2), padding='same', input_shape=(66, 200, 3), activation='elu'))
    model.add(Conv2D(36, (5, 5), strides=(2, 2), padding='same', activation='elu'))
    model.add(Conv2D(48, (5, 5), strides=(2, 2), padding='same', activation='elu'))
    model.add(Conv2D(64, (3, 3), strides=(2, 2), padding='same', activation='elu'))
    model.add(Conv2D(64, (3, 3), padding='same', activation='elu'))

    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1, activation='elu'))

    optimizer = Adam(lr=1e-3)
    model.compile(loss='mse', optimizer=optimizer)
```

이 모델은 66x200x3 크기의 이미지를 입력으로 받습니다. 이는 캡쳐된 이미지의 높이, 너비 및 색상 채널(RGB)을 나타냅니다. 첫 번째 컨볼루션 레이어에서는 24 개의 필터를 사용하고, 필터 크기는 5x5, 스트라이드는 2 로 설정하였으며, 활성화 함수로는 ELU(Exponential Linear Unit)를 사용하여 이미지에서 고수준의 특징을 추출합니다. 이어지는 두 번째와 세 번째 레이어에서는 각각 36 개와 48 개의 필터를 사용하며, 이들 레이어의 구조도 첫 번째 레이어와 유사합니다. 네 번째와 다섯 번째 레이어는 64 개의 필터를 사용하고, 필터 크기를 3x3 으로 설정하여 보다 세밀한 특징을 추출하는 데 초점을 맞췄습니다.

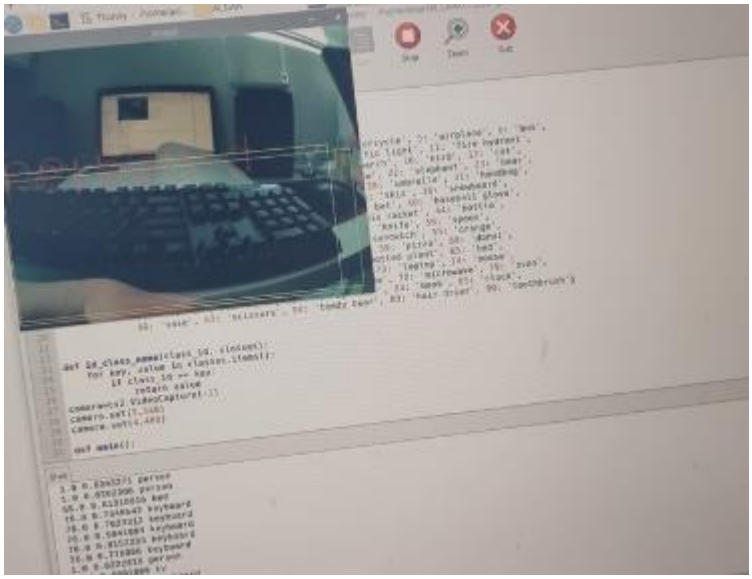
다차원 컨볼루션 출력을 단일 벡터로 평탄화하는 Flatten 레이어를 통해 완전 연결 레이어(Fully Connected Layer, FCL)로의 입력이 가능해집니다. FCL 은 100, 50, 10 개의 뉴런을 갖는 세 개의 레이어로 구성되어 있으며, 이 레이어들은 특징 벡터를 바탕으로 최종 출력을 예측합니다. 과적합을 방지하기 위해 Dropout 레이어를 추가하였으며, 손실 함수로는 MSE(Mean Squared Error)를 사용하여 예측 각도와 실제 각도 사이의 오차를 계산합니다. Adam 최적화 알고리즘을 활용하여 모델을 효율적으로 학습시켰습니다.





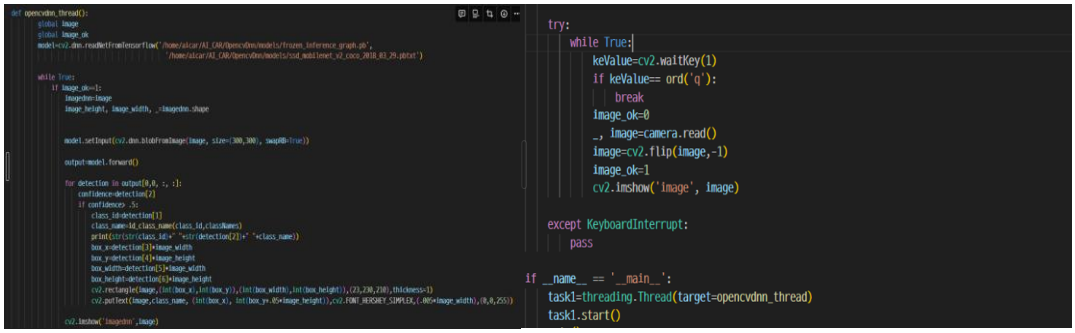
라인 인식을 위한 학습 과정에서는 시뮬레이션 상황을 만들고, 자율주행 휠체어를 조작하여 카메라가 움직이기 시작하면 1초마다 이미지를 캡쳐하였습니다. 총 9,220장의 이미지를 사용하여 이미지 전처리 과정을 거친 후, 학습을 진행하였습니다. 이 학습 과정에서는 10번의 epoch 동안 loss 값이 감소하는 것을 확인하였으며, 실제 주행 테스트에서는 모델이 라인을 따라 잘 주행하는 것을 확인할 수 있었습니다.

객체탐지 부분은 코코 데이터셋으로 학습된 MobileNet-SSD 모델을 사용하였습니다



OpenCV DNN+MobileNet-SSD 를 사용하여 객체중 일치하는 것이 있으면 bounding box 와 class_name 을 그려주고 확률을 나타내 이미지를 출력합니다.

1~2 초에 한번씩 감지가 되는 것을 볼 수 있습니다. 속도를 빠르게 하기위해 이미지 사이즈를 줄일 수 있지만 인식률에 영향을 줄 수 있어 객체 인식부분을 쓰레드로 분리하였습니다.



```
try:
    while True:
        keValue=cv2.waitKey(1)
        if keValue== ord('q'):
            break
        image_ok=0
        _, image=camera.read()
        image=cv2.flip(image,-1)
        image_ok=1
        cv2.imshow('image', image)

except KeyboardInterrupt:
    pass

if __name__ == '__main__':
    task1=threading.Thread(target=opencvdnn_thread)
    task1.start()
```

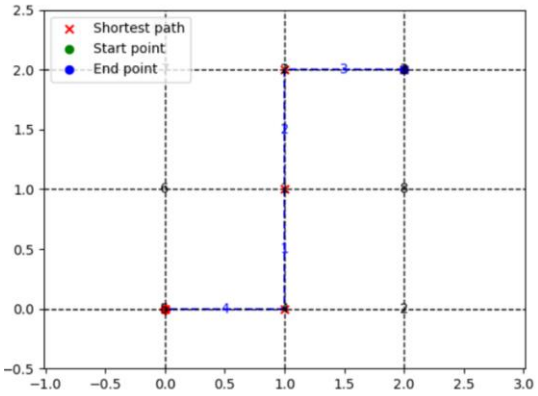
이렇게 하면 main 함수에서 동작하는 [image]는 빠르게 출력하고, opencvdnn_thread 함수에서 동작하는 [imagednn]은 객체가 감지될 때마다 출력되므로 1~2 초에 한번씩 출력됩니다.

객체 인식부분에서는 라즈베리 파이에서 사람, 버스, 정지 신호, 자동차 객체를 탐지하는 시스템을 구현했습니다. 탐지된 객체의 신뢰도가 50% 이상일 경우 차량이 정지하고, 객체가 사라지면 다시 움직일 수 있도록 설정했습니다.

라즈베리 파이에서 영상 프레임 인식이 지연되는 문제를 해결하기 위해, 카메라가 1 초에 30 프레임을 처리하는 것을 확인하고, frame_skip 값을 15 로 지정하여 1 초에 2 프레임을 처리하도록 조정했습니다. 이를 통해 객체 인식 속도가 다소 빨라진 것을 확인할 수 있었습니다.

최단경로 설정 부분에서는 마커를 통해 좌표와 가중치를 주고 다익스트라 알고리즘을 이용합니다.

아래와 같이 최단경로가 나타나게 됩니다.



나타난 최단경로대로 움직이게 하는 부분을 전체적인 흐름에 있어 딜레이되지 않게 하기 위해 쓰레딩으로 분리하여 주었습니다.

7. 프로덕트 실험 실험 설계 내용

휠체어 사용자가 인공지능이 미처 인식하지 못한 장애물로 인해 위급 상황에 빠질 가능성을 고려하여, 사용자에게 직접 조종할 수 있는 기능을 제공하기 위해 블루투스 모듈을 활용한 휠체어 제어 기능을 구현하였습니다. 이를 통해 사용자는 필요 시 스마트폰을 통해 휠체어를 안전하게 조작할 수 있습니다.

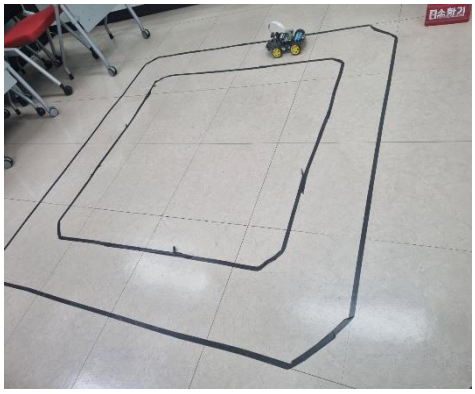
```
def serial_thread():  #시리얼 통신 스레드
        global gData  #serial_tread라는 함수안에서 gData를 사용하기위해 선언해준다.
        while True:
            data = BlSerial.readline() #한줄씩 값을 받는다.
            data = data.decode() #decode로 시리얼통신의 bytes 타입을 문자열 타입으로 변경한다.
            gData = data  #받은 데이터를 gData에 대입해준다.

def main():
    global gData  #gData를 사용하기 위해 선언해준다.
    try:
        while True:
            if gData.find("go") >= 0:  #find를 통해서 go라는 값을 찾는다. 찾는다면 조건을 실행
                gData = ""    #gData를 비어둔다. 비어두지 않으면 gData는 계속 go로 항상 참
                print("ok go")   #go가 들어온것을 확인하기 위해서 ok go라는 대사를 출력
                moving_Front(50)
                OUT_LED_GO()

            elif gData.find("back") >= 0:
                gData = ""
                print("ok back")
                moving_Back(50)
                OUT_LED_BACK()
            elif gData.find("left") >= 0:
                gData = ""
                print("ok left")
                moving_Left(50)
                OUT_LED_LEFT()
            elif gData.find("right") >= 0:
                gData = ""
                print("ok right")
                moving_Right(50)
                OUT_LED_RIGHT()
            elif gData.find("stop") >= 0:
                gData = ""
                print("ok stop")
                moving_Stop()
                OUT_LED_STOP()
```

블루투스 모듈에서 데이터를 읽어와서 gData 변수에 저장합니다.

gData 의 내용을 확인하여 "go", "back", "left", "right", "stop" 명령에 따라 휠체어를 제어합니다.

비상멈춤 버튼을 눌렀을 때 휠체어를 멈춥니다.

시리얼 통신 스레드를 실행하고, 메인 함수에서 휠체어 제어 로직을 실행합니다.



처음에는 검은 우드락 위에 흰색 테이프로 트랙을 만들어 라인을 인식했습니다. 그러나 이 방법은 크기에 제한이 있었고, 크기와 수정의 자유도를 높이기 위해 흰 바닥에 검은 테이프로 라인을 만들었습니다. 초기 시도에서는 곡선 코스에서 자율 주행 중 오류가 발생했습니다. 이를 해결하기 위해 학습 시 곡선 코스에서도 최대한 두 개의 선이 보이도록 주행하였고,

곡선으로만 이루어진 트랙도 설계하였습니다. 이로써 자율 주행 시스템의 성능을 향상시키고자 했습니다.

라인학습을 시킬 때 빛의 상태나 라인의 색상 등 여러 요인으로 인해 달라질 수 있으므로 매주 최대한 비슷한 환경에서 시행하였습니다..


우리가 일반적으로 보는 RGB 영상을 인공지능이 학습하기 좋은 YUV 형식으로 변환해주었습니다..

YUV형식은 RGB표현을 사용할 때보다 더 효율적으로 특징을 찾아 낼 수 있습니다

위에서 만들 실제 트랙을 시계 방향으로 1~2바귀 반시계 방향으로 1~2바퀴를 조종하며 주행하여 데이터를 학습하였습니다.

```
if carState == "go":
    if steering_angle >= 60 and steering_angle <= 110:
        print("Straight")
        motor_go(speedSet)
    elif steering_angle > 110:
        print("right")
        motor_Right(speedSet)
    elif steering_angle < 60:
        print("left")
        motor_Left(speedSet)

elif carState == "stop":
    motor_Stop()
```

그 후 학습데이터를 기반으로 더 나은 주행을 위해 steering angle 값을 조정해주었습니다.

Steering anlge 값이 60~110 일때는 직진 주행, 110 이상일때에는 오른쪽으로 회전 60 이하일 때는 왼쪽으로 회전하게 하였습니다.

여기서 steering angle 값은 자율주행 휠체어가 진행 방향을 결정하기 위해 사용하는 조향 각도입니다.

객체인식부분에서는 OpenCV DNN+MobileNet-SSD 를 사용하여 학습된 결과 파일을 적용한 후, 실시간으로 객체를 인식하여 표시하는 코드로 모델을 불러와 카메라로 받은 이미지를 가져와 image 에 넣어주었습니다.

그 후 모델을 입력 후 학습된 90 개의 객체중 일치하는 것이 있으면 bounding box 와 class_name 을 그려주고 확률을 나타내 이미지를 출력하였습니다.

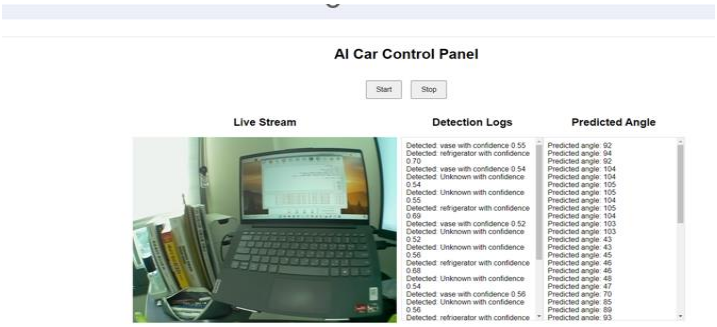
```python
def obdect_detection_thread():
    global image
    global image_ok
    global carState
    global object_detected

    while True:
        if image_ok == 1:
            imagednn = image
            image_height, image_width, _ = imagednn.shape

            input_data = cv2.resize(imagednn, (300, 300))
            input_data = np.expand_dims(input_data, axis=0)
            input_data = input_data.astype(np.uint8)

            interpreter.set_tensor(input_details[0]['index'], input_data)
            interpreter.invoke()

            detection_boxes = interpreter.get_tensor(output_details[0]['index'])
            detection_classes = interpreter.get_tensor(output_details[1]['index'])
            detection_scores = interpreter.get_tensor(output_details[2]['index'])
            num_detections = interpreter.get_tensor(output_details[3]['index'])

            print(f"Num detections: {num_detections}")
            print(f"Detection boxes: {detection_boxes}")
            print(f"Detection classes: {detection_classes}")
            print(f"Detection scores: {detection_scores}")

            detection_class_names = [id_class_name(int(cls), classNames) for cls in detection_classes[0]]
            print(f"Detection class names: {detection_class_names}")

            object_detected = False

            for i in range(int(num_detections[0])):
                class_id = int(detection_classes[0][i])
                score = detection_scores[0][i]
                if score > 0.5:
                    class_name = id_class_name(class_id, classNames)
                    print(f"Detected: {class_name} with confidence {score}")
                    if class_name in ["person","traffic light", "stop sign", "bicycle", "car", "motorcycle", "bus"]:
                        box_x = detection_boxes[0][i][1] * image_width
                        box_y = detection_boxes[0][i][0] * image_height
                        box_width = detection_boxes[0][i][3] * image_width
                        box_height = detection_boxes[0][i][2] * image_height

                        carState = "stop"
                        object_detected = True
                        print("auto stop")
                        motor_Stop()

                        cv2.rectangle(image, (int(box_x), int(box_y)), (int(box_width), int(box_height)), (23, 230, 210), thickness=2)
                        text = f"{class_name}: {score:.2f}"
                        (w, h), _ = cv2.getTextSize(text, cv2.FONT_HERSHEY_SIMPLEX, 0.6, 1)
                        cv2.rectangle(image, (int(box_x), int(box_y) - 20), (int(box_x) + w, int(box_y)), (23, 230, 210), -1)
                        cv2.putText(image, text, (int(box_x), int(box_y) - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 1)

            if not object_detected and carState == "stop":
                carState = "go"
                print("object not detected, resuming movement")

            cv2.imshow('Detection', image)  # 객체 감지 결과를 화면에 표시
            image_ok = 0
```

위의 코드는 객체 감지 스레드를 구현하여, 실시간으로 카메라에서 입력된 이미지를 처리하고 객체를 감지하는 코드입니다. 감지된 객체가 "person", "traffic light", "stop sign", "bicycle", "car", "motorcycle", "bus" 중 하나일 경우 휠체어를 멈추고, 감지된 객체가 없으면 휠체어의 이동을 재개하도록 하였습니다. 감지 결과는 화면에 표시됩니다.

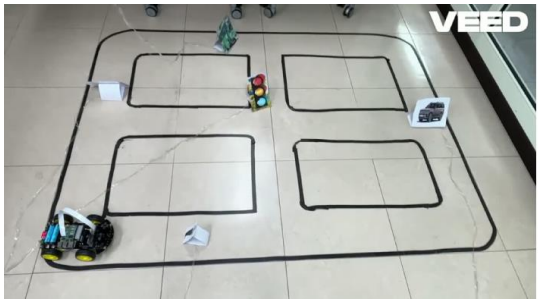
```python
import tensorflow as tf

# TensorFlow 모델 디렉토리 설정
saved_model_dir = './ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/saved_model'

# TFLite 변환기 설정
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)

# 추가적인 최적화 설정 (선택 사항)
# converter.optimizations = [tf.lite.Optimize.DEFAULT]

# 모델 변환
print("Converting model to TensorFlow Lite format...")
tflite_model = converter.convert()
print("TensorFlow Lite conversion complete.")

# 변환된 모델을 .tflite 파일로 저장
tflite_model_path = 'model.tflite'
with open(tflite_model_path, 'wb') as f:
    f.write(tflite_model)
print(f"TensorFlow Lite model saved to {tflite_model_path}")
```

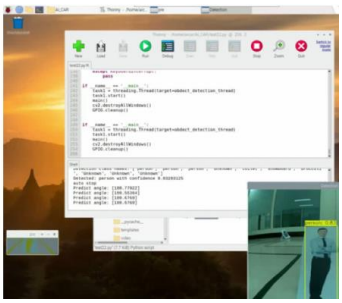
```python
import tensorflow as tf

# TensorFlow 모델 디렉토리 설정
saved_model_dir = './ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/saved_model'

# TensorFlow Lite 변환기 설정
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)

# TensorFlow Lite에서 지원되는 연산으로만 변환
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS, tf.lite.OpsSet.SELECT_TF_OPS]
tflite_model = converter.convert()

# 변환된 모델을 .tflite 파일로 저장합니다.
with open('model_no_flex.tflite', 'wb') as f:
    f.write(tflite_model)
```

라즈베리파이에서 객체 탐지 모델을 TensorFlow 프레임워크로 구현했을 때 영상 처리가 느린 문제가 있어 이를 해결하기 위해 TensorFlow Lite 버전으로 변경하여 성능을 개선했습니다.



또한 객체인식화면에서 화면의 진행이 매끄럽지 않아 Flask 서버를 통해 카메라의 스티어링 각도와 객체 인식 로그를 받아오고, Start 버튼을 눌렀을 때 객체 인식 주행을 시작하려고 했습니다. 그러나 Flask 서버로 데이터를 전송하는 과정에서 라즈베리파이의 CPU 리소스 사용이 많아져 카메라 전송이 지연되는 문제가 발생했습니다. 따라서 이러한 문제를 해결하기 위해 라즈베리파이 내부에서 직접 실행하는 방법으로 변경하였습니다

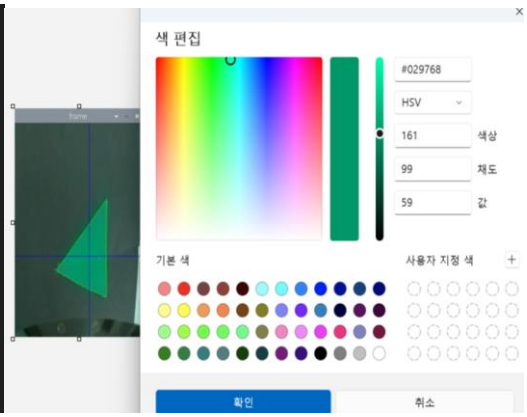




라인 인식 코드와 객체 인식 코드를 결합하여, 실제 주행 중 사람, 자동차, 자전거 등의 장애물이 나타나는 상황을 시뮬레이션 하였습니다. 이를 위해 위에서 제작한 맵에 장애물 사진을 인쇄하여 경로 곳곳에 배치하여 테스트 환경을 조성하였습니다.

최단경로설정부분에서는 각 좌표 값마다 삼각형, 사각형, 오각형을 배치하고, 이 도형들을 RGB 세 가지 색깔로 구분하였습니다. 이러한 설정을 통해 카메라가 도형을 인식하여 색상과 꼭짓점 수를 파악할 수 있도록 하였습니다.

```python
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

lower_color = np.array([
    [0, 76, 76],     # Red (low)
    [170, 76, 76],   # Red (high)
    [78, 200, 100],  # Green (low)
    [98, 140, 140]   # Blue (low)
])

upper_color = np.array([
    [10, 255, 255],  # Red (low)
    [180, 255, 255], # Red (high)
    [88, 255, 148],  # Green (high)
    [108, 180, 163]  # Blue (high)
])
```



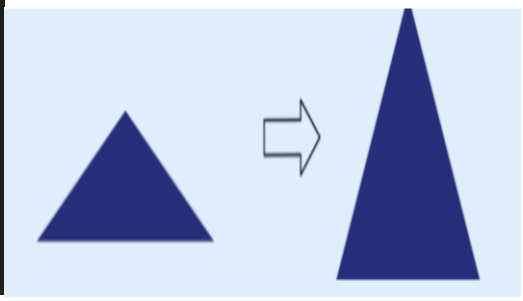
최단경로부분 에서는 각 좌표 값마다 삼각형, 사각형, 오각형을 배치하고, 이 도형들을 RGB 세 가지 색깔로 구분하였습니다. 이러한 설정을 통해 카메라가 도형을 인식하여 색상과 꼭짓점 수를 파악할 수 있도록 하였습니다.

마커의 색상을 인식하기 위해 카메라로 도형을 캡처한 후, 이를 그림판에서 RGB 값으로 나타낼 수 있지만, 이는 주변 조명 등의 영향을 받아 명도와 채도 값이 달라질 수 있는 문제가 있습니다. 이를 해결하기 위해 색상 값을 HSV 형식으로 전환하여 명도와 채도의 변화에 대한 저항성을 높였습니다.

```
    for c in contours:
        epsilon = 0.06 * cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, epsilon, True)
        vertices = len(approx)

        contour_area = cv2.contourArea(c)
        perimeter = cv2.arcLength(c, True)

        if contour_area > MIN_CONTOUR_AREA and contour_area > max_contour_area and perimeter > 10:
            if 3 <= vertices <= 5:
                max_contour_area = contour_area
                max_contour = c
                vernum = vertices
                color_num = i
```

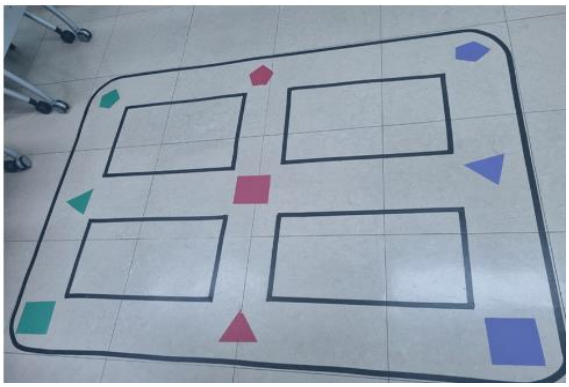
도형을 인식하기 위해, cv2.approxPolyDP 함수를 사용하여 윤곽선의 근사 다각형을 찾았습니다. epsilon 값은 근사 다각형을 찾기 위한 정확도 파라미터로, 전체 둘레의 6%로 설정되어 있습니다. 이 함수를 통해 근사된 다각형의 꼭짓점 수를 vertices 변수에 저장하고, vertices 의 값에 따라 도형의 모양을 판단합니다

```
def trace_thread():
    global image_ok
    global image

    while True:
        if image_ok==1:
            frame = image

            pts1 = np.float32([[220,250],[432,250],[30,440],[620,440]])
            pts2 = np.float32([[0,0],[300,0],[0,450],[300,450]])
            M = cv2.getPerspectiveTransform(pts1,pts2)

            flatframe = cv2.warpPerspective(frame, M, (300,450))
            # Call the function to track the blue object
            track_color_object(flatframe)

            cv2.imshow('frame', flatframe)
            image_ok=0
```



마커를 바닥에 배치하다 보니 카메라가 수직으로 촬영하지 않고 비스듬하게 촬영하는 경우가 많아 기존의 원근 이미지를 평면 이미지로 변환하여 처리하였습니다.

주행할 맵은 밑의 사진과 같이 만들었습니다.



```python
def dijkstra(matrix, start, end):
    # (previous node, current node) for path tracking
    prev_nodes = [[None] * len(matrix[0]) for _ in range(len(matrix))]
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    # Initializing tables for shortest path
    distance = [[float('inf')] * len(matrix[0]) for _ in range(len(matrix))]
    distance[start[0]][start[1]] = matrix[start[0]][start[1]]

    min_heap = [(matrix[start[0]][start[1]], start)]

    while min_heap:
        current_dist, current_node = heapq.heappop(min_heap)

        for i, (dr, dc) in enumerate(directions):
            new_row, new_col = current_node[0] + dr, current_node[1] + dc

            if 0 <= new_row < len(matrix) and 0 <= new_col < len(matrix[0]):
                # If direction changes, add 1 to the weight
                weight_change = 1 if i != matrix[current_node[0]][current_node[1]] else 0
                new_dist = current_dist + matrix[new_row][new_col] + weight_change

                if new_dist < distance[new_row][new_col]:
                    distance[new_row][new_col] = new_dist
                    prev_nodes[new_row][new_col] = current_node
                    heapq.heappush(min_heap, (new_dist, (new_row, new_col)))

    path = []
    current = end
    while current is not None:
        path.append(current)
        current = prev_nodes[current[0]][current[1]]

    return distance, path
```

다익스트라 알고리즘을 사용하여 주어진 행렬에서 시작점부터 끝점까지의 최단 경로를 찾습니다. min_heap 을 사용하여 최소 비용을 추적하고, 각 노드의 이전 노드를 저장하여 경로를 재구성합니다. 방향 변화 시 가중치를 추가하여 최단 경로를 계산합니다.

```python
def update_plot():
    global current_slope, current_position, previous_positions
    if previous_positions:
        previous_positions_np = np.array(previous_positions)
        plt.plot(previous_positions_np[:, 0], previous_positions_np[:, 1], 'g-')

    plt.scatter(current_position[0], current_position[1], c='r')

    plt.pause(0.01)
```

previous_positions 에 저장된 경로를 녹색 선으로, 현재 위치를 빨간 점으로 플롯에 실시간으로 업데이트합니다. plt.pause(0.01)을 사용하여 플롯을 잠시 멈추고 갱신합니다.

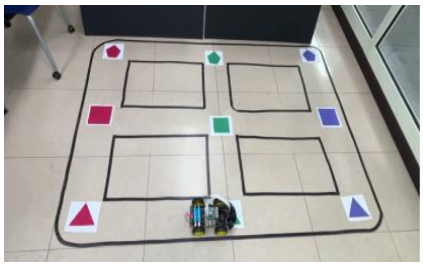
```python
def move_along_path():
    global x_pos, y_pos, steering_angle, steering_ok, map_go
    rrrr, patht = dijkstra(matrix_map, start_coord, end_coord)
    varr=0
    time.sleep(0.5)
    directions = []

    for i in range(len(patht) - 1):
        current_node = patht[i+1]
        next_node = patht[i]
        direction = (next_node[0] - current_node[0], next_node[1] - current_node[1])
        directions.append(direction)
    print('patht : ',patht)
    print('directions : ',directions)
    print('patht.0 0:',patht[(len(directions)-1)][0])
    print('patht.0 1:',patht[(len(directions)-1)][1])

    while True:

        for xy in range(len(directions)-1):
            varr=0
            #print("True False:",(y_pos==patht[xy][0]) and (x_pos==patht[xy][1]))
            #if (abs(y_pos - patht[xy][0]) < 0.5) and (abs(x_pos - patht[xy][1]) < 0.5):
            if (x_pos == int(patht[xy+1][1])  and (y_pos == int(patht[xy+1][0])):
                print('succeed')
                steering_ok=False

                fre_dire=directions[xy+1]
                cur_dire=directions[xy]
                print('fre_dire: ',fre_dire,'cur_dire: ',cur_dire)

                x_dif=cur_dire[0]-fre_dire[0]
                y_dif=cur_dire[1]-fre_dire[1]
                print('x_dif: ',x_dif,'y_dif: ',y_dif)
                map_go=False

                if x_dif<1 and y_dif<1:
                    #steering_ok=True
                    steering_angle=90
                    print('change go')
                    map_go=True
                    time.sleep(7)
                else:
                    time.sleep(4.5)
                map_go=True

                if x_dif>0:
                    steering_angle=45
                    print('change left')
                    time.sleep(3)

                elif y_dif>0:
                    steering_angle=135
                    print('change right')
                    time.sleep(3)
                x_pos=100
                y_pos=100
                varr=1
    if varr>0:
        x_pos=100
        y_pos=100
        varr=0
    steering_ok=True
```

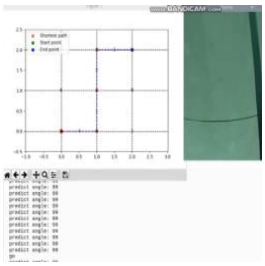
다익스트라 알고리즘을 통해 계산된 경로를 따라 이동하는 로직을 구현합니다. 경로의 각 지점 간 방향을 계산하고, 현재 위치와 목표 지점 간의 방향을 비교하여 조향 각도(steering angle)를 조정합니다. 특정 지점에 도달하면 방향을 변경하고, 이동을 재개합니다. 최종적으로, 실시간으로 위치를 업데이트하며 경로를 따라 이동합니다.

```
def shortest_path_and_mapping():
    global matrix_map, start_coord, end_coord, path
    global current_slope, current_position, previous_positions
    global carState, steering_angle, map_go

    plt.figure()
    plt.axis('equal')
    plt.show(block=False)
    result, path = dijkstra(matrix_map, start_coord, end_coord)
    if result[end_coord[0]][end_coord[1]] != float('inf'):

        path_x, path_y = zip(*path)
        plt.imshow(np.ones_like(matrix_map), cmap='gray', alpha=0)

        for i in range(len(path) - 1):
            x_values = [path[i][1], path[i + 1][1]]
            y_values = [path[i][0], path[i + 1][0]]
            plt.plot(x_values, y_values, color='blue', linestyle='dashed')
            weight_label = f"{matrix_map[path[i][0]][path[i][1]]}"
            plt.text((x_values[0] + x_values[1]) / 2, (y_values[0] + y_values[1]) / 2, weight_label, color='blue',ha='center', va='center')

        plt.gca().invert_yaxis()
        # Add horizontal and vertical lines at 0.5 and 1.5
        plt.axhline(y=0, color='black', linestyle='dashed', linewidth=1)
        plt.axhline(y=1, color='black', linestyle='dashed', linewidth=1)
        plt.axhline(y=2, color='black', linestyle='dashed', linewidth=1)
        plt.axvline(x=0, color='black', linestyle='dashed', linewidth=1)
        plt.axvline(x=1, color='black', linestyle='dashed', linewidth=1)
        plt.axvline(x=2, color='black', linestyle='dashed', linewidth=1)

        plt.scatter(path_y, path_x, color='red', marker='x', label='Shortest path')
        plt.scatter(start_coord[1], start_coord[0], color='green', marker='o', label='Start point')
        plt.scatter(end_coord[1], end_coord[0], color='blue', marker='o', label='End point')
```

```
    for i in range(len(matrix_map)):
        for j in range(len(matrix_map[0])):
            plt.text(j, i, str(matrix_map[i][j]), color='black', ha='center', va='center')

    plt.legend()
while True:

    previous_positions.append(current_position.copy())

    if carState == "go":
        if steering_angle >= 85 and steering_angle <= 95 and map_go==True:
            current_position += 0.2*np.array([np.cos(current_slope), np.sin(current_slope)], dtype=np.float64)

        elif steering_angle > 96:
            current_slope -= np.radians(7)
            #current_position += 0.01*np.array([np.cos(current_slope), np.sin(current_slope)], dtype=np.float64)

        elif steering_angle < 84:
            current_slope += np.radians(7)
            #current_position += 0.01*np.array([np.cos(current_slope), np.sin(current_slope)], dtype=np.float64)


    update_plot()
```

Dijkstra 알고리즘으로 최단 경로를 계산하고 이를 그래픽으로 시각화합니다. 휠체어의 상태가 "go"일 때 steering_angle 에 따라 휠체어의 위치와 방향을 업데이트하며, previous_positions 에 이전 위치를 저장해 실시간으로 위치를 갱신하여 그래프에 표시합니다. update_plot 함수를 사용해 휠체어의 현재 위치를 시각화합니다.





따라서 전체코드를 실행하면 오른쪽 사진과 같이 최적경로가 설정되고 휠체어는 지정된 경로를 따라 라인과 마커를 인식하며 주행하게 됩니다.

8. 실험 결과에 대한 설명

사용자가 휠체어의 자율주행 기능만으로 주행하다가 위험에 빠질 수 있는 상황을 대비하여, 블루투스를 통해 직접 조작할 수 있는 기능을 구현하였습니다.

블루투스 조작: https://youtu.be/2_wiKAO4sD0?si=IcgNYwRb1IOVsJt5

도로를 인식하여 차선(도보)를 유지하며 주행할 수 있는 상황을 시뮬레이션하였습니다. .

라인인식 주행: https://youtu.be/ZKHQAO6BSBQ

예상치 못한 장애물을 감지했을 때 안전하게 정지하고 장애물이 사라진 후 주행을 재개하는 기능을 구현하였습니다.

객체인식 주행:1: https://youtu.be/3f7In2cvNJg?si=U6kk80CyGBEDJzYS

객체인식 주행 2: https://youtu.be/3yJEzsuRwks?si=DYXpgkIoEzPgvX27

출발지와 목적지를 설정한 후, 교통 혼잡도를 반영하여 최적 경로를 설정하고, 해당 경로를 따라 주행하도록 구현하였습니다.

경로지정 주행 1: https://youtu.be/Tl5BwMToeVY

경로지정 주행2: https://youtu.be/rR67AxUV6p0

전체 시연영상 및 설명: https://youtu.be/s794wi4A7Ck?si=tBx-BXjRpyfTALJa

9.결론

AUTOPIOLOT 프로젝트는 블루투스 조종, 라인 인식 자율주행, 객체 인식, 최적 경로 지정 및 주행 기능의 구현을 목표로 하여 자율주행 휠체어 시스템을 개발하였습니다. 각각의 기능을 성공적으로 구현하여, 실험 결과 휠체어가 설정된 경로를 따라 정확하게 이동하고 장애물을 효과적으로 회피하며 사용자에게 안전하고 편리한 제어 기능을 제공함을 확인하였습니다.

프로젝트 진행 중 라즈베리파이의 처리 속도와 맵 크기의 제한과 같은 하드웨어적 제약이 있었으나, 이를 극복하기 위해 각 기능을 개별적으로 구현하고 테스트하였습니다. 이를 통해 전체 시스템의 신뢰성과 실용성을 높이는 데 중점을 두었습니다.

여러 시행을 통해 자율주행 휠체어의 가능성과 잠재력을 확인할 수 있었으며, 향후 더욱 정교한 알고리즘 개발과 다양한 환경에서의 테스트를 통해 시스템의 성능을 지속적으로 개선할 계획입니다. 이러한 발전은 휠체어 사용자의 이동 편의성과 안전성을 크게 향상시킬 수 있을 것으로 기대됩니다.

프로젝트의 성과는 자율주행 휠체어의 실현 가능성을 보여주었으며, 이는 휠체어 사용자의 자립성을 높이고 삶의 질을 개선하는 데 기여할 것입니다. 향후 관련 공모전이나 대회에 참가할 기회가 주어진다면, 본 프로젝트를 기반으로 하여 더욱 발전된 자율주행 휠체어 시스템을 선보일 계획입니다. 이를 통해 전문가들의 피드백을 받고, 실용적인 측면에서의 개선점을 찾아내어 실제 사용자에게 더욱 유용한 솔루션을 제공하고자 합니다. 앞으로도 지속적인 연구와 개선을 통해 더욱 발전된 자율주행 휠체어 시스템을 개발할 수 있기를 기대합니다.

10. 참고문헌

1. Automatic Moving Vehicle using by Raspberry Pi Woo-Ri Cho*, Joo-Hyeon Ahn**, Da-Young Lee**, Yeon-Ju Yu** *Dept. of Computer Science, DongDuk Women's University ** Dept. of Computer Science, DongDuk Women's University

2. Implementation of Autonomous Vehicle using Raspberry Pi Tae-Sun KimO, Yang-Hyuk Jang*, Hyeon-Dong Jeong* O*Dept. of Avionics Engineering, Kyungwoon University

3. 한국통신학회: 자율주행자동차 기술 동향 및 핵심 기술 – 성경복, 민경욱, 최정단

4. AI 인공지능 자율주행 자동차(앤써북) 도서

5. End to End Learning for Self-Driving Cars – NVIDIA