WE-Meet 프로젝트 결과보고서

기초로봇공학실험 9조
팀원 : 김동현, 김유상, 박준식

1. 프로젝트 개요

1) 프로젝트 소개

자율주행 차량의 대중화를 위한 일반인 대상 주행면허 교육 프로그램을 개발하는 것을 시나리오로 잡아 3가지 요소 기술을 탐색하는 것을 중점으로 한다. 요소 기술로써는 인지(도로상황의 종류, 위치, 상황 확인 등) / 결정(안전하게 주행하기 위한 동작 판단) / 행동(주행면허 시험장의 개별 미션 수행)이 존재한다. 이 시나리오를 토대로 운전면허 장내기능 시험을 상황으로 설정 후, Wego사에서 제공한 맵지를 토대로 하여 Limo를 활용해 프로젝트를 진행한다.

2) 프로젝트 달성 목표

프로젝트의 기본적인 목표는 프로젝트 당일에 공지되는 정해진 미션들을 주행하여 빠르게 완주를 하는 것 이다. 주어지는 미션들은 활용법에 따라 분류 가능하다.

1. 표지판에 부착된 ArUco Marker 인식 후 알맞은 행동 취하기(좌회전, 우회전, 정지, 주차)
2. LiDAR을 활용해 장애물 인식 후 정지하기(돌발상황(보행자), 차단기)
3. 관성 측정 장치(IMU)를 활용해 알맞은 행동 취하기(방지턱)
4. 카메라를 활용하여 차선 인식

3) 추진배경 및 필요성

자율주행차 시대에서는 자동차에 탑재된 A.I.도 운전능력을 검증받아야 한다. 현재 도로교통법 제80조에 의하면 자동차등을 운전하려는 사람은 시·도경찰청장으로부터 운전면허를 받아야 한다. 만약 자율주행차와 일반자동차가 뒤섞인 상황에서는 운전자 / A.I.가 숙지하고 있어야 할 기본적인 부분이 추가될 여지도 고려되고 있다.

그렇기에 프로젝트의 상황을 운전면허 장내기능 시험으로 잡고, 현재 운전면허시험과 유사한 미션을 수행하는 것을 목표로 삼았다. 지금 이 프로젝트는 Limo를 통하여 진행하였지만, 이를 확장할 시 주행면허 교육 프로그램을 개발할 수 있다. 만약 A.I.나 사람이 추가로 운전능력을 검증해야 하거나 추가로 숙지해야 할 점이 추가된다면 이 프로젝트를 확장해 주행면허 교육 프로그램을 이용할 수 있을 것이다.

4) 예상 결과물

이 프로젝트를 성공적으로 완수할 시에 꽤나 완성도 있는 자율주행차(Limo)라는 결과를 얻을 수 있다. 주어진 미션을 모두 달성했다는 가정 하에, ArUco Marker을 인식하여 좌회전, 우회전, 정지, 주차를 할 수 있으며 장애물을 인식하고 정지할 수 있고, 관성 측정 장치를 이용하여 방지턱을 인식 후, 적절한 행동(속도의 감소)을 취할 수 있게 된다. 여기서 ArUco Marker을 이용한 자동 주차 방식은 현재 많은 기업에서 시도하는 방식이기에, 여러

분야에서도 활용될 수 있을 것으로 보인다.

2. 프로젝트 수행과정

전반적인 팀원들의 역할 분배는 김유상 학생은 로봇 구동의 알고리즘 구상, 박준식 학생은 구상한 알고리즘을 사용하기 위한 센서들의 사용방법, 수학적 계산식 등을 찾는 역할, 김동현 학생은 전체적인 총괄 및 구상해준 알고리즘과 계산식들을 코드로 구현하는 역할을 맡아 프로젝트를 진행하였다. 프로젝트 수행은 우선 전체적인 구동 알고리즘을 구상하고 하나씩 해결해나가는 계획을 구상하였고 차선인식, aruco marker 동작, lidar, 부가적인 기능 순으로 해결해나가기로 하였다.

1) launch

프로젝트를 시작하기에 앞서 앞으로의 작업을 조금이라도 더 편리하게 하는 것이 가장 중요하다고 생각하여 기존의 launch 파일 3개를 하나하나 매번 실행해야 하는 것을 수정하는 것이 급선무라고 판단하였다. 따라서 우선 새로운 ros 패키지를 생성하여 해당 디렉터리에 작업공간을 구축하였다. 또한 start.launch 파일에 roscore 등이 있는 limo_bringup, 카메라 실행, 주행 노드 실행의 3가지 launch 파일을 하나로 통합하였다. 하지만 카메라 실행 노드와 ArUco Marker 실행 노드가 모두 카메라를 이용하는 노드여서 그런지 실행에서 충돌이 발생하였다. 따라서 우선은 카메라 노드를 실행하고 카메라가 이미지를 받아오기 시작했다면 Bool 형 변수를 True로 전환시키도록 하여 해당 변수가 True인 경우에 ArUco Marker 실행 노드인 maker.launch를 실행하도록 하였다. 이 덕분에 roslaunch 명령 하나만 입력해주면 모든 것이 한 번에 실행되어 후의 작업에서의 편리함을 얻을 수 있었다.

또한 새로 생성한 ros 패키지를 GitHub에 연결하여 Nomachine 없이 github에서 수정하고 로봇에서 git pull 명령 하나로 수정 사항이 반영될 수 있게 하여 작업의 효율성 또한 증가시킬 수 있었다.

2) 차선인식

첫 번째는 차선인식 부분이다. 기존의 코드는 로봇 기준에서 왼쪽 차선의 무게중심만을 계산하여 이로부터 어느정도 떨어진 거리를 유지하며 주행한다. 하지만 전체적인 주행이나 교차로 내부의 주행에서 왼쪽 차선만을 보고 주행하는 것이 다소 불안정함을 확인하였기에 오른쪽 차선도 볼 필요가 있다고 판단하였다. 따라서 기존의 왼쪽 아래부분의 이미지만 잘라 사용하는 코드에서 오른쪽 아래부분의 이미지도 잘라내어 동일하게 흑백처리 후 노란색 부분을 추출한 뒤 무게중심을 계산하는 과정을 거쳤다. 또한 기존의 코드에서는 왼쪽 차선의 무게중심을 기준으로 얼마나 떨어져 주행할 것인지를 계산하는 식에서 REF_X 전역변수를 사용한다. 여기에 오른쪽 차선으로 얼마나 떨어져 주행할 것인지에 관여할 REF_X2 전역변수를 정의하였다. 그리고 해당 무게중심들로부터 얼마나 떨어져 주행할 것인지를 계산한 후 주행 각속도를 결정하는 drive_data.angular.z 계산식에서 계산한 거리들을 더하고 회전할 정도를 정하는 LATERAL_GAIN을 곱하여 결정하였다. 이렇게 하면 양쪽 차선을 인지하며 로봇이 어느 공간에서는 두 차선의 중심에 존재하며 주행한다.

하지만 커브 구간이나 일부 교차로 내부에서 왼쪽 차선이 오른쪽 아래부분의 이미지에 침범

하면 비정상적인 회전을 해버리는 문제가 발생한다. 이는 커브구간에 진입하였을 때 로봇에게 알려주면 해결할 수 있는데, 왼쪽 아래부분 이미지의 가장 오른쪽 한 줄을 적분하였을 때 True이고(픽셀을 적분하는 것이기에 차선을 인식하지 못하면 모두 검은색일텐데 이 경우에는 결과가 0이기에 False를 반환하고, 하얀색 부분이 조금이라도 있으면 적분결과가 0이 아니기에 True를 반환한다), 동시에 오른쪽 아래부분의 이미지의 가장 왼쪽 한 줄을 적분한 결과도 True인 경우는 커브구간임을 의미하기 때문에 이 경우에 True를 퍼블리시하여 해당 구간에서는 오른쪽 아래부분 이미지에 따른 계산을 무시하도록 하였다.

양쪽 차선을 인식하게 되면 직진 구간에서는 빠르게 달리는 것이 어떨까 하는 생각을 하였고 이 또한 적분을 이용하여 왼쪽 아래부분 이미지의 첫 번째 행과 오른쪽 아래부분 이미지의 첫 번째 행의 적분결과가 모두 1보다 큰 경우(두 이미지에서 차선이 검출된 경우), True를 퍼블리시하여 로봇에 직진구간임을 알려주었고 이 경우에는 기존 속도의 1.4배 빠르게 주행하도록 하였다.

3) ArUco Marker

차선인식을 개선시킨 후 이제 본격적으로 ArUco Marker 동작을 시작하였다. 하지만 초기에는 맵지 내에 ArUco Marker가 어디에 배치될지 확정이 되지 않았기 때문에 개인적으로 루트를 구상하여 우선 해당 루트 동작을 완수할 수 있는 코드를 제작하기로 하였다. 가장 처음으로 접근한 방식은 뎁스 카메라를 이용해 ArUco Marker와의 거리를 기준으로 동작을 수행시키는 것이었다. ros의 topic list를 열어 ArUco Marker와 관련한 모든 노드를 열어보았고, /ar_pose_marker 토픽에서 ArUco Marker와 관련한 모든 정보들을 퍼블리시하고 있는 것을 확인하였다. marker.pose.pose.position에 인식한 ArUco Marker와 떨어진 x,y,z축 거리 정보가 있었고 해당 정보를 점과 점 사이 거리 공식을 이용해 로봇과 떨어진 거리를 계산하였다. 계산한 ArUco Marker와의 거리를 이용해 특정 거리 내에 진입하였을 때 여러번의 실험으로 측정한 선속도와 각속도, rospy.get_time()을 이용한 ArUco Marker를 인식한 시점으로부터 지난 시간을 활용하여 모션 제어로 수행하도록 하였다. 하지만 이 방식에는 큰 문제점이 있었다. ArUco Marker가 항상 실험할 때의 그 위치에 배치되지 않으면 원하는대로 동작하지 않는다는 것이었다. 그래서 우리 조는 ArUco Marker가 어느 위치에 배치되어도 원하는 동작을 수행할 수 있는 명확한 기준점이 필요하였다.

두 번째 방법으로 넘어가기 전 ArUco Marker의 처리방식부터 변경해주었다. 우선 ArUco Marker를 인식하면 떨어진 거리값부터 계산한다. 이후 marker의 id에 따라 특정 조건을 만

족하는지를 판단하는 첫 번째 필터를 거쳐 만족하면 collect라는 전역변수에 해당 마커의 정보를 저장함과 동시에 found_sign이라는 메서드에 해당 정보를 전달한다. 그리고 found_sign 메서드에서 두 번째 필터를 거쳐 해당 조건을 만족하면 주행 데이터를 조절하는 메서드로 넘어갈 수 있도록 하였다. 이렇게 복잡하게 구성한 이유는 조금이라도 정확성을 높이기 위함이다. 또한 marker 정보는 가벼운 조건들만 만족하면 전역변수에 저장될 수 있도록 하여 혹여나 카메라의 화각을 벗어나도 가지고 있을 수 있도록 하였고, 이후 복잡한 두 번째 조건을 만족하면 실행될 수 있도록 한 것이다. ArUco Marker 관련 노드는 주기적으로 정지, 우회전, 좌회전, 주차 동작을 수행하는 메서드를 실행시키는데 해당 동작에 맞는 ArUco Marker 정보가 오지 않으면 계속해서 return 당한다. 하지만 그에 해당하는 ArUco Marker 정보가 들어오면 해당 메서드에 맞는 주행 동작을 수행시키는 것이다.

ArUco Marker 인식 알고리즘을 변경한 후 두 번째로 넘어간 방법이 맵지 내의 횡단보도 표시를 이용하는 것이었다. 교차로 내부에는 없지만 외부 루트에는 우회전이나 좌회전을 시도하는 부분에는 횡단보도 표시가 존재한다. 따라서 차선을 검출하는 방식도 비슷하게 이미지를 흑백처리하고 흰색 부분을 검출 후 에지검출로 이미지 내 선의 개수를 이용해 횡단보도임을 인식하도록 하였고 인식한 경우 True를 퍼블리시하도록 하였다. 이를 이용하니 확실히 교차로 외부 루트에서는 동작이 깔끔해졌음을 확인하였다. 하지만 교차로 내부에서는 횡단보도가 존재하지 않아 당연히 원하는 동작을 수행하지 못하였다.

그렇게 세 번째로 시도한 방법은 차선을 이용한 ArUco Marker 인식이었다. 교차로 내외부 상관없이 우회전을 해야하는 곳은 오른쪽 차선이 뚫려있고, 좌회전을 해야하는 곳은 왼쪽 차선이 뚫려있다. 이는 명확한 사실이기에 이를 로봇이 인지할 수 있으면 좋겠다는 생각에 해당 방법을 시도하게 되었다. 우선 이미지의 중심을 기준으로 x,y의 범위를 지정하고 이를 전체 이미지에 각각 곱한 정보를 matrix_x와 matrix_y에 저장하였다. x축 계산은 단순히 픽셀들을 곱하였고, y축 계산은 전체 이미지의 전치행렬에 가중치를 곱하고 다시 전치행렬을 취하여 값을 얻었다. 그리고 해당 값들이 특정 조건들을 거치고 arctan 값을 구하고 해당 값의 전체를 적분하고, 전체 이미지를 적분한 값과 나누어 차선의 평균 기울기를 계산하였다.

이제 계산한 평균 기울기(gtan라고 하겠다) 값을 활용할 차례이다. 실험을 통해 오른쪽 차선이 없어지는 시점에는 gtan가 -0.5보다 커지고, 왼쪽 차선이 없어지는 시점에는 0.5보다 작아짐을 확인하였다. 이를 이용하여 오른쪽 차선이 없어지는 시점에 우회전 동작을 수행하고, 왼쪽 차선이 없어지는 시점에 좌회전 동작을 수행하도록 하여 방향전환 ArUco Marker는 완벽한 기준점을 잡는데에 성공하였다.

정지마커는 그 자리에서 멈추기만 하면 되기에 문제가 없었다. 이제 남은 것은 주차 마커 동작이다. 주차 ArUco Marker는 배치지점을 결정해주셨기 때문에 모션 제어를 이용하여 수행하기로 하였다. 하지만 ArUco Marker와의 거리값이 회전할 정도가 모든 시도에서 완벽하게 동일한 것은 아니기 때문에 매번 주차 구간으로 들어가는 정도가 달랐다. 따라서 여기서도 gtan를 이용하여 주차 구간에 진입 후, gtan가 0이 되는 시점이 평행하게 존재함을 실험적으로 알게 되었고 진입 후 gtan가 0이 될 때까지 제자리 회전하여 주차구간에 진입 후 정렬할 수 있는 로직을 구성할 수 있었다. 진입 후 좌회전 ArUco Marker를 인식하여 나갈 때도 모션제어이기에 매번 동일하지 않았다. 이 또한 gtan를 이용하여 차선과 평행하게 위치할 때까지 회전하도록 하여 완벽한 정렬기준을 잡을 수 있었다.

4) LiDAR

커브구간을 돌거나 ArUco Marker를 인식하고 난 후 가끔씩 로봇이 벽이나 표지판을 장애물로 인식하여 멈춰버리는 문제가 발생하였다. 실제 시연 상황이라면 그대로 종료겠지만 방향을 꺾어주기만 하면 다시 주행이 가능하다는 점에 주목하여 LiDAR가 장애물을 5초이상 인식하고 있는 경우 오른쪽으로 방향을 크게 틀어버리도록 하여 다시 주행으로 복귀할 수 있도록 하였다.

5) IMU

다음은 방지턱을 넘는 동작을 해결해야 했다. 방지턱을 넘는 순간 카메라도 같이 위로 올라가므로 차선인식이 불안정해짐과 동시에 차선을 잘못 인식하여 오른쪽으로 틀어지는 문제가 있음을 확인하였다. 이는 방지턱 자체를 인식하는 방법도 있겠지만 쉽지 않을 것이라 생각하여 로봇 내에 탑재된 IMU 센서를 이용하기로 하였다. ros의 topic list를 살펴본 결과 IMU라는 토픽에서 IMU 센서가 측정한 정보들을 퍼블리시하고 있는 것을 확인하였다. 하지만 여기에는 x,y,z의 로봇의 위치와 회전 정도 등과 관련한 정보만 담겨있었기에 수학적인 계산이 추가로 필요하였다. 로봇이 실행되는 시점에 rospy.get_time()으로 시간을 재고, imu 센서로부터 서브스크라이브할 때마다 ros시간을 받아와서 두 값을 빼어 dt를 얻었다. 그리고 각속도를 적분하면 각위치라는 점에서 시작하여 y축 각속도를 dt를 이용하여 적분하였다. 하지만 적분상수로 인한 지속적인 오차가 누적되기 때문에 (1-dt)를 곱하여 해당 값을 지속적으로 0으로 수렴시켜 오차를 줄였다. 이렇게 되면 기본 주행을 할 때는 일정한 값이 방지턱을 넘는 순간 증가하게 된다. 이 점을 이용하여 방지턱을 넘고 있을 때 True를 퍼블리시하도록 하여 로봇이 방지턱을 넘고 있다는 것을 인지하도록 하였다. 만약 True 신호를 받았을 경우, 로봇의 주행속도를 절반으로 줄여 방지턱을 실제처럼 안전하게 넘을 수 있도록 하였다. 또한 이 순간에는 차선인식 정보도 받아오지 못하게 하여 직진만 하여 방향이 틀어지는 문제도 해결하였다.

6) Voice Module

IMU 센서를 활용하는 것을 끝으로 기본적인 주행에는 문제가 없는 상태의 LIMO가 되었다. 이제 로봇이 탑재되어 있는 모든 것을 활용해보고 싶다는 생각에 마지막으로 남은 Voice Module에 주목하였다. 로봇의 설정창을 확인해보니 스피커 기능이 있는 것을 확인하였다. 우선 파이썬의 gtts 모듈을 활용하여 ‘stop, right, left, park, start, finish’의 6가지의 mp3파일을 제작하였다. 그리고 pygame 모듈을 활용하여 로봇의 주행이 시작되었을 때, 정지, 우회전, 좌회전, 주차 marker를 인식했을 때, 로봇의 주행을 끝마쳤을 때 해당 mp3 파일이 재생되도록 하였다.

하지만 pygame을 import 하여 각종 내장 함수들을 여럿 사용하니 로봇이 처리량을 견디지 못하여 심한 딜레이가 발생하였다. 위의 차선인식과 ArUco Marker 처리 로직에서 상당히 메모리를 사용하는 것이 원인이라고 판단하였다. 따라서 Bool형 전역변수를 선언하여 마커를 인식하였을 때 True를 퍼블리시하도록 하였고 True인 경우에만 mp3 파일을 1번만 재생하도록 하여 pygame의 내장함수들을 거의 사용하지 않았다. 확실히 딜레이는 크게 줄어들었지만 여전히 딜레이는 존재하였기 때문에 실제 시연 발표 때는 사용하지 않기로 하였다.

3. 프로젝트 수행결과

우리 조의 로봇의 기능 및 구동 방식을 요약하면 다음과 같다. 양쪽 차선을 인식하여 직진구간, 커브구간 등을 판단하여 주행하고 차선의 평균 기울기를 이용해 ArUco Marker 동작이 수행 가능한 위치를 판단하여 수행하고 이 값을 양쪽 차선과 평행하게 위치하게 하기 위해서도 사용한다. 또한 IMU 센서를 이용해 방지턱을 넘는 순간을 인지하여 속도를 줄이고, Voice Module을 이용해 특정 상황에서 맞는 mp3 파일을 재생시키도록 한다.
아래는 앞서 말한 약 4개월 간의 우리 조의 모든 작업과정들이 담겨있는 깃허브의 주소와 limo 내에 존재하는 거의 모든 코드들을 담아 필요한 코드들을 조원들과 함께 분석한 자료들과 작업 중간중간 찍은 동영상 등 모든 작업과정들이 녹아있는 노션의 링크이다.
github 링크 / Notion 링크
프로젝트를 수행하면서 WEGO Robotics에서 주최하는 대회에 참여신청도 하였었다. 실제로 손실값이 0에 수렴하는 차선을 인식하는 인공지능 모델을 제작하였고 대회준비를 열심히 하였지만, 대회가 평일인 점과 모든 미션들을 수행할 준비가 되지는 않았다는 점을 고려하여 아쉽게도 참여취소를 하게 되었다. 비록 대회에 출전하지는 못하였지만 대회를 준비하는 과정은 인공지능 제작과 SLAM에 대한 기본 지식과 같이 많은 것들을 알아갈 수 있는 시간이었다.
또한 현재 WE-Meet에 업로드하고 있는 보고서 등을 평가하는 대회에 신청을 한 상태이다. 4개월 간 정말 많은 시행착오와 시도들을 차근차근 녹여내린 보고서들인만큼 좋은 성적을 거둘 수 있으면 좋겠다는 마음이다.
하지만 시연 결과는 좋지 않았다. 어째서인지 시연 발표에서 모든 알고리즘이 무너져내려 오작동을 거듭하였고 결국 어떠한 동작도 완벽하게 수행하지 못하였다. 순간은 4개월간의 노력을 로봇에게 부정당한 기분이었지만 실패는 성공의 어머니라는 말이 있듯이 이번의 실패는 앞으로의 성공을 위한 디딤돌이라고 생각하기로 하였다. 실제로 후에 WEGO Robotics의 팀장님과 메일을 통해 함께 원인을 분석하였고 우리 조가 제작한 구동 코드를 로봇의 메모리가 버티지 못하는 것으로 예상하였다. 정말 아쉬운 결과와 원인이었지만 후에도 이런 문제가 발생하지 않게 하기 위해 고성능 파이썬 코드를 제작하기 위해 효율적인 코드 작성과 관련한 공부를 시작할 수 있게 해준 트리거가 되어준 값진 결과였다.

4. 예상효과

이 프로젝트는 솔직하게는 대학생 1학년들에게는 상당히 어려운 개념들을 활용하는 작업이었다. 하지만 그런만큼 1학년이 배울 수 없는 많은 것을 알아갈 수 있었다. 그저 이론만 배우고 아두이노나 라즈베리파이 등을 실습하는 것도 배워가는 것이 많겠지만 젯슨나노 기반으로 앞으로 정말 많이 쓸 ROS를 직접 활용하는 시간을 가지는 것이 처음에는 어짜피 나중에 배울텐데 왜 지금 하는 걸까하는 마음이었지만 지금 생각해보면 지금 배우는 것은 정말 행운이었구나라고 느꼈다. ROS의 기본 및 심화 활용 과정을 누군가에 의해서가 아닌 직접 조사하고 경험해보며 학습하고 바로 실습해볼 수 있는 시간이었기에 상당히 값진 시간이었다. 따라서 이 프로젝트는 후에 다른 1학년 학생들도 꼭 경험하여 과정은 힘들지만 결과는 정말 아름다운 고진감래의 성과를 거둘 수 있는 시간을 가졌으면 하는 바람이 있다.

5. 스터디 자료

이 프로젝트를 진행하기 위해 스스로 많은 시간을 들였다. 깃허브를 사용하기 위해 git에 대

해 공부하고, 인공지능 모델을 제작하기 위해 Tensorflow와 아나콘다에 관한 공부를 하였다. 또한 차선인식을 개선하기 위해 OpenCV와 관련한 공부를 하고, ROS에 대해 알기 위해 팀원들과 ROS2 스터디까지 진행하였다. 내용을 하나하나 적기에는 정말 많기에 아래에 ROS2 스터디 자료를 담은 노션 링크를 업로드한다.
ROS2 스터디