[현장맞춤형 이공계 인재양성지원사업 – 성균관대학교]

실전문제해결형 S-HERO 공학인재 양성사업 4차년도 결과보고서

연구과제명	Autonomous Vehicle Separating and Coupling			
연구기간	**2020년 3월 1일 ~ 2020년 12월 31일 (4차년도)**			
구분	**성명**	**소속**	**직위**	**연락처**
지도교수	전재욱(인)	전자전기공학부	교수	031-290-7129
산업체멘토	박성천(인)	한국전자통신연구원	실장	010-8866-1999
구분	**성명**	**학교**	**학부(과)**	**학번**
연구팀장	권용현(인)	성균관대학교	전자전기컴퓨터공학	2020710623
연구팀원	박준호(인)	성균관대학교	전자전기공학부	2017310073
	박진석(인)	성균관대학교	전자전기공학부	2016310047
	홍영욱(인)	성균관대학교	기계공학부	2016312182
	이주헌(인)	성균관대학교	기계공학부	2017315484
	임진우(인)	성균관대학교	기계공학부	2014310788

본 연구팀은 「실전문제해결형 S-HERO 공학인재 양성사업단」의 관리 운영 지침에 의거, 4차년도 결과보고서를 다음과 같이 제출합니다. 아울러, 보고서에는 사실과 다른 내용이 포함되지 아니하였으며 만약 허위 사실이나 중대한 오류가 발견될 경우 그에 상응하는 불이익을 감수할 것을 서약합니다.

2020 년 11 월 26 일

연구팀장 : 권용현(인)

지도교수 : 전재욱 (인)

성균관대학교 실전문제해결형 S-HERO 공학인재양성사업단장 귀하

* 연구팀원 : 팀원이 4명 이상인 경우 칸을 추가하여 작성 요망

연구 결과 요약문

연구의 목적 및 내용	변화하는 글로벌 차량공유 시스템에 맞추어 사용할 수 있는 모듈형 공유 차량 모델을 제작하여 소비자의 요구에 맞추어 상황에 따라 결합과 분리를 자유롭게 진행할 수 있는 새로운 종류의 이동수단을 개발하고자 한다. 차량의 분리 결합의 과정은 차량 이동, 차량 정렬, 차량 개조 및 결합으로 크게 3단계로 나눌 수 있다. 이번 연구에서는 차량의 분리 결합 과정 중 하나인 앞, 뒤 차량을 일렬로 정렬하는 시스템을 영상처리 기술을 이용하여 구현해보고자 한다.
연구결과	초기 목표는 차량 분리 결합부를 설계하고 차량 간 이동 및 정렬을 실현하여 분리 결합 과정을 시연하는 것이었지만 차량에 적합한 분리 결합부 고안의 한계, 분리 결합 후의 통신 문제 등으로 인해 영상인식을 이용하여 차량 간 이동 및 정렬만을 시연하기로 하였다. 우리는 Python에서 OpenCV를 사용하여 카메라로 체스판의 corner를 인식한 뒤 왜곡의 정도를 계산하고 이를 좌회전 우회전 직진 신호로 나타내었다. 그리고 이 신호를 Python에서 아두이노로 Wifi 통신을 이용해 전달하고 아두이노에서는 전달받은 신호를 바탕으로 스테핑 모터에 회전을 주어 바퀴를 제어하기로 하였다. 그러기 위해서 먼저 체스판이 부착된 차량과 카메라가 부착된 차량이 필요해 아크릴을 사용해 제작하였다. 테스트 도중 카메라가 부착된 차량이 움직일 경우 왜곡의 변화량이 너무 커져 체스판을 잘 인식하지 못하는 문제가 발생하였다. 따라서, 대안으로 체스판이 부착된 차량이 움직이면서 카메라 차량에 접근하는 방식을 채택하였다. 따라서 체스판이 부착된 차량에 4개의 스테핑 모터를 부착하고 아두이노를 연결하였다. 카메라가 캘리브레이션을 통해 왜곡 계수 rvecs를 계산하면 차량이 움직여야 할 방향이 결정되는데 이러한 방향을 직진 좌회전 우회전 3개로 분류하고 모드 변수에 각각 저장하였다. 왜곡 계수 rvecs는 왜곡이 클수록 1에 가깝고 작을수록 0에 가깝다. 또한, 좌우 회전에 따라 부호가 결정된다. 이러한 특성을 이용하여 모드 변수 M1, M2, M3를 밑의 그림과 같이 결정하였다.  아두이노에서 Python으로 모드를 받아오면 해당 모드에 맞는 알고리즘이 작동하여 차량이 움직이도록 하였고 차량이 일직선이 될 때까지 이러한 일련의 과정을 반복하였다. 실험 결과 차량의 방향이 엇갈려 있는 경우에도 성공적으로 정렬을 실현할 수 있었다.

연구결과의 활용계획	본 연구의 활용계획은 다음과 같다. - 좁은 땅에 비해 비약적으로 커지는 도시 때문에 야기되는 거주자들의 이동 효율성 저하 해소 - 모터엔진을 기반으로 한 Electrical Vehicle의 형태로 배출가스를 내뿜지 않아 도심 공해 해소 - 여러 대의 결합을 통해서 자율 군집 운행으로 교통과 물류 산업에 혁신적인 변화 - 결합 후 차량 간 배터리 공유 기능으로 배터리 방전으로 인한 전기차 안전성 향상 - 목적에 따라 차량을 생산 가능하기 때문에 배터리 충전용 모듈 자율차를 생산해서 차량들을 따라다니며 무선으로 전력을 공급하는 기능 구현 - 탑승자의 취향과 목적에 따라 분리 결합을 통해서 맞춤형 이동수단 및 공간 제공 가능 - 목적에 의해서 분리 결합을 통해 다양한 형태와 기능 활용 - 이동하는 대중교통 및 모빌리티 환승 거점 허브로 사용 - 다양한 활용성을 통해 현재의 모빌리티 구조가 아닌 새로운 유기적인 모빌리티 생태계를 구축 - 새로운 모빌리티 생태계로 인해 도로가 변화함에 따라 녹지 환경도 새롭게 탈피 - 새로운 기술을 바탕으로 사람과 환경이 어우러진 스마트하고 지속 가능한 도시 유지

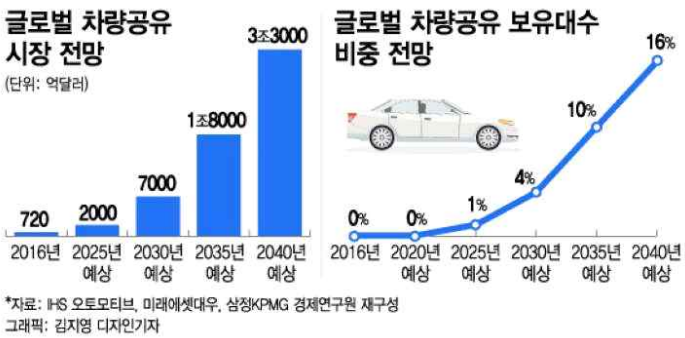
* 최종 결과 발표회 제출 시 작성한 요약문과 동일해도 무관합니다. 더 진행된 연구 내용이 있다면 반영하여 수정 작성하여도 됩니다.

연구내용 및 결과 보고서

1. 연구과제의 개요

1.1 연구 배경 및 목적

자동차 기술이 나날이 발전하고 있는 가운데 자율주행 시대가 머지않아 도래할 것으로 예측된다. 세계의 대부분의 자동차 회사들은 자율주행차 상용화를 목표로 자율주행 연구에 뛰어들고 있다. 사실 현재 시장에 판매되고 있는 자동차에도 자율주행 기술을 살펴볼 수 있는데 전방충돌방지 보조나 차로이탈방지 보조 등이 그 예이다. 따라서 우리는 자율주행 기술이 대중화되는 시대를 대비해야 할 필요가 있다. 자율주행차가 상용화된다면 발생할 수 있는 현상들은 여러 가지가 있지만 그 중에서도 자동차의 공공화 현상이 있다. 자율주행 시대가 오면 운전자가 필요 없어지기 때문에 사유물이던 자동차가 공유물로 바뀌는 현상이 발생할 것이다. 즉, 자율주행 기술과 차량 공유라는 트렌드가 합쳐져 새로운 시대를 열 것으로 전망된다. 실제로 현재 자동차 제조 회사들은 공유 자동차 사업에 뛰어들고 있고 이에 따라 공유자동차 시장은 더욱 더, 커질 것이다. [그림 1-1]은 이러한 사실을 잘 보여준다. 공유 자율주행차 시대가 온다면 더욱더 편리한 삶을 누릴 수 있다는 것은 분명한 사실이지만 그럼에도 불구하고, 발생 가능한 문제점이 두 가지 있다.



[그림 1-1] 글로벌 차량공유 전망

첫 번째, 도로에 공유 자율주행차가 많아진다면 차량의 크기와 모양은 단편 일률적으로 변화할 것이며 이에 따라 차량은 고객의 이동수요를 충족시키기 어려울 것이다. 예컨대 탑승 인원이 많아 승합차가 필요한 상황에서 거리에 승용차만 있다면 고객은 불편함을 겪을 수 있고 차량 운영 측면에서도 비효율적이다. 또한, 업무상 승합차만을 끌어야 하는 사람, 승합차보다는 승용차만을 원하는 사람이 분명 존재할 것이고 공유자율주행차는 이러한 사람들의 요구를 반영하지 못할 것이다.

두 번째, 미래의 과제이자 현재의 과제이기도 한 주차문제이다. 현재에는 개인당 자동차를 소유하고 있기 때문에 차량이 많고 그만큼 주차공간이 부족한 실정이다. 공유자동차가 많아진다면 도심 외곽에도 차량을 주차할 수 있고 주차해 있는 차량이 줄어들 것이기 때문에 주차문제가 다소 해결 될 수 있을지도 모른다. 하지만 대부분의 자동차회사들은 접근성을 높여 더욱더 많은 이윤을 창출하기 위해 공유자율주행차를 도심과 최대한 가까운 곳에 주차하려고 할 것이다. 이러한 과정에서 자동차 회사들 간의 주차 경쟁이 발생할 것이며 같은 공간에 더 많은 차를 주차 할 수 있다면 경쟁력을 갖출 수 있을 것이다.

차량 공유에 관한 문제점들과 더불어 전기를 동력으로 하는 자율주행차가 늘어나면서 전력 수요가 많아지는 현상이 발생할 것이다. 기존에 있던 충전소로 이러한 수요를 충족시키기엔 턱없이 부족

할 것이고 새로운 종류의 전기 충전 방법이 필요하다. 앞서 언급한 문제점들을 해소하고 더 나아가 미래 모빌리티 기술의 지평을 열기 위해 우리는 새로운 종류의 모빌리티를 고안하였다,

1.2 연구 대상 및 목표

본 연구에서는 새로운 모빌리티인 모듈형 자율주행차를 고안해본다. 모듈형 자율주행차란 차량 간 분리 결합이 가능한 자율주행차로서 칸을 이어 붙일 수 있는 열차와 유사한 개념이다. 이동소요가 많을 시 두 자율주행차가 결합하여 승합차의 형태를 갖추고 이동소요가 적을 때는 분리되어 개별 승용차로서 역할을 수행한다. 즉, 한 대의 차량이 결합하여 여러 대의 차량이 되기도 하고 그 반대가 되기도 하는 운송수단이다.

차량 간 분리결합을 위해 우선 차량에 적합한 새로운 분리결합부를 설계할 필요가 있다. 이를 위해 현재 화물차와 트레일러를 결합할 때 사용하는 커플러, 열차를 연결할 때 사용하는 연결기를 살펴본다. 또한 결합시 내부공간이 하나로 합쳐 질 수 있는 방안을 모색해본다.

차량 간 분리결합에 앞서 필수적으로 선행되어야할 기술이 있는데 바로 차량 간 정렬기술이다. 차량 간 정렬이 제대로 이루어지지 않으면 어긋남이 발생하여 차량 결합부가 오작동을 일으킬 것이다. 이는 곧 사고로 이어 질 수 있기 때문에 매우 정밀한 기술을 필요로한다. 본 연구에서는 실제 자동차 모형을 제작하고 패턴인식을 통해 정렬을 구현해본다.

1.3 관련 분야 현황

2011년에 한국자동차공학회 추계학술대회 및 전시회에 '결합 분리가 가능한 준하이브리드 자작차량'이라는 논문이 실렸다. 홍익대학교 기계정보공학과 김준수 외 8명이 저자로 참여하였다. 해당 논문에는 결합시 5륜 차량이 되고 분리시 3륜 또는 2륜 차량이 되는 차량의 디자인과 결합 방식, 안전성 평가 등이 나와있다. 자율주행차가 아니기 때문에 결합이 수동으로 이루어지고 2륜이나, 3륜의 BIKE 형태의 차량간의 결합이기 때문에 본 연구에서 제시하는 모듈형 자율차와는 차이가 있어 따로 참고하지 않았다. 밑의 [그림 1-2]가 최종 디자인 사진이다.

[그림 1-2] 결합 분리가 가능한 준하이브리드 자작차량

한편 독일의 DFKI GmbH의 로보틱 이보베이션 센터(Robotic Innovation Centre)에서 Smart Connecting Car라는 이름의 컨셉 카를 제시했는데 논문에 따르면 해당차는 접을 수 있는 샤시를 가지고 있고 같은 종류의 차들과의 결합도 가능하다. 또한, 5자유도를 가진 바퀴를 통해 주차를 편리하게 할 수 있다.

[그림 1-3] The EO Smart Connecting Car

[그림 1-3은] 해당 컨셉카를 보여준다. 마찬가지로 자율주행이 아니고 결합 시 외관이 연결되어 하나의 공간을 형성하지 않는다는 점에서 우리가 고안한 모듈형 차량과는 다르다. 하지만 우리가 고안한 차량과 매우 유사한 컨셉을 가지고 있었기 때문에 참고자료로 활용 하였다.

앞서 언급한 두 가지 논문 외에 차량 분리 결합과 관련한 자료는 찾기 힘들었다. 참고자료가 많지 않아 연구에 어려움이 예상되었지만 그만큼 유의미하고 혁신적인 성과물을 낼 수 있을 것이라 판단하에 연구에 착수하였다.

2. 연구과제수행 내용 및 결과

2.1 분리결합부 종류

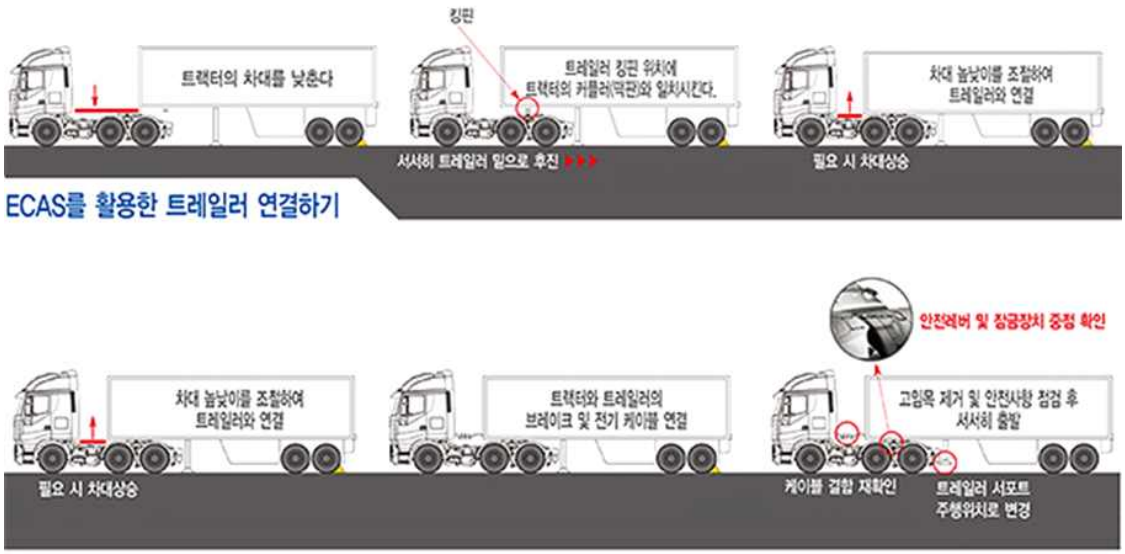
본격적인 설계에 앞서 우리가 고안하고자 하는 분리결합부와 유사한 연결부를 찾아보았다. 알아본 결과 크게 두 가지가 있었는데 하나는 화물차, 즉 트랙터와 트레일러를 연결하는데 쓰이는 커플러와 나머지 하나는 열차 간 연결에 사용되는 연결기였다.

2.1.1 트랙터의 커플러(Coupler)와 결합 시스템

트랙터와 트레일러는 커플러와 킹핀으로 연결되는데 커플러는 트랙터에 부착되어있고 킹핀은 트레일러에 부착되어있다. 결합시에는 트랙터는 차대를 낮춘 상태로 트레일러의 킹핀 위치로 이동한다. 킹핀과 커플러의 핀 구멍이 동심원을 이루면 차대 높낮이를 조절하여 킹핀과 커플러를 연결시키게 된다. 이때 킹핀을 통해 트랙터와 트레일러는 1자유도를 얻게 되며 이에 따라 차량이 코너를 돌 때 유동적으로 움직일 수 있게 된다. 뿐만 아니라 킹핀은 매우 견고하기 때문에 중량이 높은 트레일러를 견인하는데 큰 역할을 한다. 커플러는 저상커플러와 고상커플러로 나뉘며 여기서 고상커플러는 저상커플러와 달리 회전축이 두 개이기 때문에 차량의 전후뿐만 아니라 좌우 흔들림 또한 감쇠 할 수 있다. [그림 2-1]는 트랙터에 장착되어있는 커플러의 사진이다.

[그림 2-1] 커플러

커플러와 킹핀 외에도 트랙터와 트레일러의 완전한 연결을 위해선 부속장치들간의 결합이 필요하다. 우선 브레이크가 연결되어야 하는데 트랙터 뿐만아니라 트레일러의 제동 또한 잘 이루어져야 하기 때문이다. 마찬가지로 전기케이블도 후미등을 제어를 위해 꼭 필요하다. 아직까진 트랙터와 트레일러의 연결은 자동이 아닌 수동 이루어지며 전자식 제어 에어시스템이 차대 높낮이 조절만 자동으로 해준다.



[그림 2-2] ECAS 시스템

또한 트레일러 결합에서 ECAS(전자제어 에어시스템)는 연결작업 효율성을 높여주는 역할을 수행한다. 운전석의 하단에 비치되어 있어 운전자가 차량 외부에서 작동이 가능하며, 차량의 움직임을 직접 확인하면서 조작 가능한 것이 특징이다. 또한, 차체 및 전륜, 후륜의 높낮이를 조절할 수 있어 차량과 트레일러를 손쉽게 연결할 수 있다. 트렉터의 브레이크 케이블과 전기연결 케이블까지 연결을 해주면 연결된 트랙터에서 트레일러의 모든 기능은 제어할 수 있다. 전기, 공기, 브레이크 작동은 물론 후미등 역시 브레이크 작동 시 트랙터와 동일하게 반응한다.

우리는 여기에 그치지 않고 단순 결합 뿐 아니라 두 차량 간의 공간이 하나로 합쳐져 하나의 공간을 이룰 수 있도록 하기 위한 방법을 모색해보았다.

2.1.2 열차의 연결기

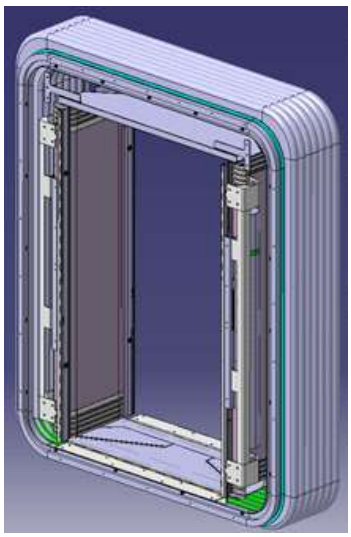
열차간 연결기로는 여러 가지 종류가 있다. 크게 체인 앤 버퍼 연결기 등의 수동연결기와 자동연결기로 나뉘는데 자동연결기는 말 그대로 자동으로 연결되기 때문에 시간을 단축시키고 작업 위험을 줄일 수 있다. 현재는 기계 연결과 전기 회로 접속을 동시에 해주는 자동복합연결기가 많이 사용되고 있다. [그림 2-3]와 같이 해방용 핸들 외에 주공기관, 전기연결기, 제동관이 하나로 구성되어있어 결합 시 한 번에 결합된다.

[그림 2-3] 자동복합연결기 구성

자동연결기를 형태별로 분류하면 너클식 연결기, 밀착식 연결기로 분류할 수 있고 밀착식이 너클식보다 충격을 더 많이 흡수할 수 있기 때문에 현재는 밀차식 연결기의 한 종류인 샤펜베르형 연결기와 시바타형 연결기가 많이 쓰인다.

연결부에 대한 조사를 마친 후 두 차량의 내부 공간이 하나로 합쳐질 수 있는 방안을 모색해보았다. 가장 중요한 문제로 코너링을 할 때에도 두 차량의 내부 공간이 분리되지 않아야 하는 문제가 있었다. 이를 해결하기 위한 여러 가지 아이디어가 제시되었고 우리는 열차의 통로 연결막을 참고하기로 하였다.

2.1.3 열차의 통로 연결막



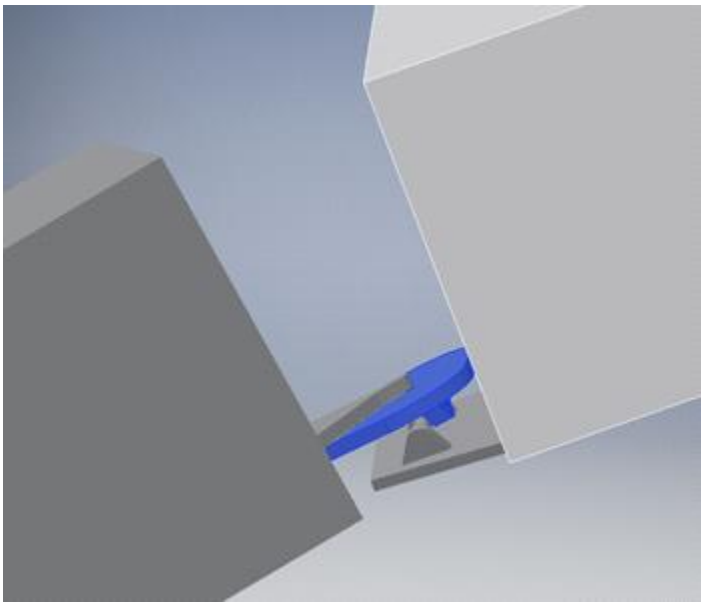
[그림 2-4] 열차 통로 연결막

[그림 2-4]에서 보는 것처럼 통로 연결막은 열차의 객차 사이를 연결해주는 역할을 하며 곡선 구간에서 신장하거나 수축할 수 있게 설계되어있다. 재질은 대부분 고무로 이루어져 있지만 장시간 운행에도 견고하다는 특성을 가지고 있다.

2.2 분리결합부 고안

2.2.1 3D 모델링

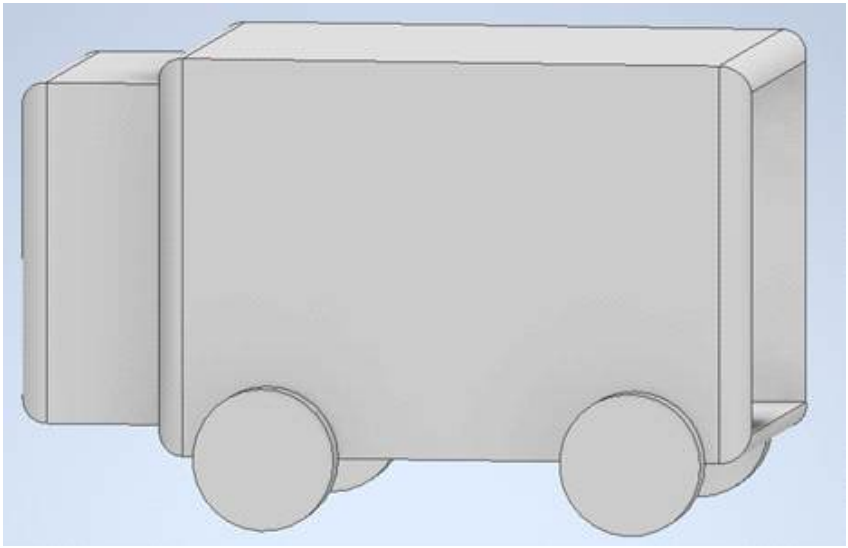
우리는 커플러 중에서도 좀 더 유동적인 고상커플러와 열차의 기계연결과 전기회로 연결을 동시에 해주는 자동복합연결기를 참고하여 새로운 분리결합부를 고안하기로 하였다. 다만, 열차의 경우 레일이 열차를 잡아주기 때문에 따로 정렬을 하지 않고도 안정적으로 결합을 할 수 있다는 점에서 차량의 분리결합부와는 큰 차이가 있었다. 뿐만아니라 열차 주행 시에도 레일을 따라가기 때문에 코너링 시 연결부가 분리 되지 않고도 고속 주행이 가능하였다. 따라서 현재 레일이 없이 달리는 화물차량에 쓰이는 분리 결합부인 고상커플러를 기본 모델로 하여 열차의 자동복합연결기와 같이 물리적인 결합 뿐 아니라 전자 및 제동 장치까지 결합해 주는 수 있는 기능을 추가하기로 하였다. [그림 2-5]은 고상커플러를 인벤터를 통해 3D 모델링 한 결과이다.



[그림 2-5] 고상커플러 모델링

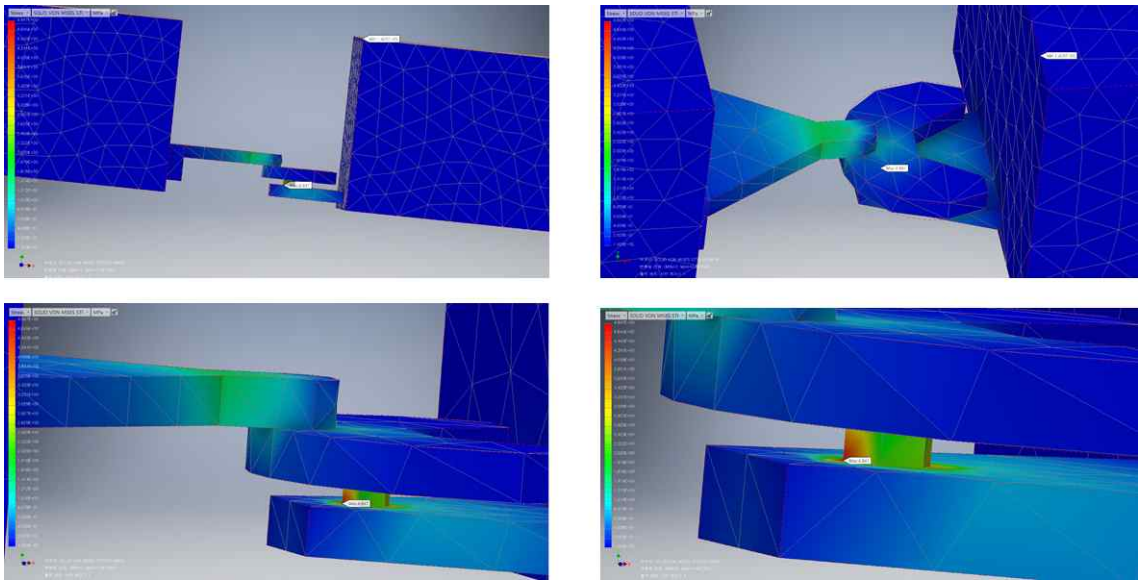
먼저 가장 단순한 형태로 차체를 제작한 후 한 차에는 킹핀을 부착하고 다른 한 차에는 커플러를 부착하였다. 여기서 커플러와 차체는 어셈블리 과정을 통해 조립하였으며 두 개의 핀을 사용하여 고상커플러와 같이 전후좌우로 움직일 수 있도록 하였다. 마지막으로 킹핀과 커플러를 부착하여 모델링을 마무리하였다.

다음으로 외관에 대해서는 열차의 통로연결막을 기본 모델로 하였다. 차량에 통로연결막을 적용하기 위해서는 [그림 2-6]처럼 차량이 버스와 같이 육면체의 형상을 띄어야 했다.



[그림 2-6] 통로연결막이 설치된 차량 모델링

2.2.2 모델링 해석



[그림 2-7] 분리 결합부 모델링 해석

트레일러의 고상 커플러 구조를 사용하기 위해서 응력 해석을 진행해보았다[그림 2-7]. 분리 결합 모듈형 자동차의 특성 상 두 차량간의 수직 및 수평 진동의 주파수가 다르고, 조향시 수평적인 응력이 집중되는 특징또한 가지고 있기 때문에 안정성 평가를 위해 응력해석은 필수적이라 생각하였다.
[그림 2-7]과 같이 두 차량의 주행중에 고상 커플러의 핀 모서리 부분에서 응력 집중이 발생하는 것을 확인할 수 있었다. 핀을 사용하는 커플러의 특성 상 모서리 부분에 응력이 집중될 수 밖에 없었고, 주행중인 상황뿐만 아니라 급격한 충격이 더해지는 차량 사고 상황까지 고려한다면 커플러의 사용은 차량에서 가장 중요한 문제인 안정성 부분에서 크게 떨어지는 모습을 보이는 한계점이 있었다.

2.2.3 한계점

많은 아이디어와 고상커플러, 자동복합연결기와 같은 참고 모델에도 불구하고 차량에 적합한 분리결합부를 고안하는데 다음과 같은 3가지 문제점들이 있었다.

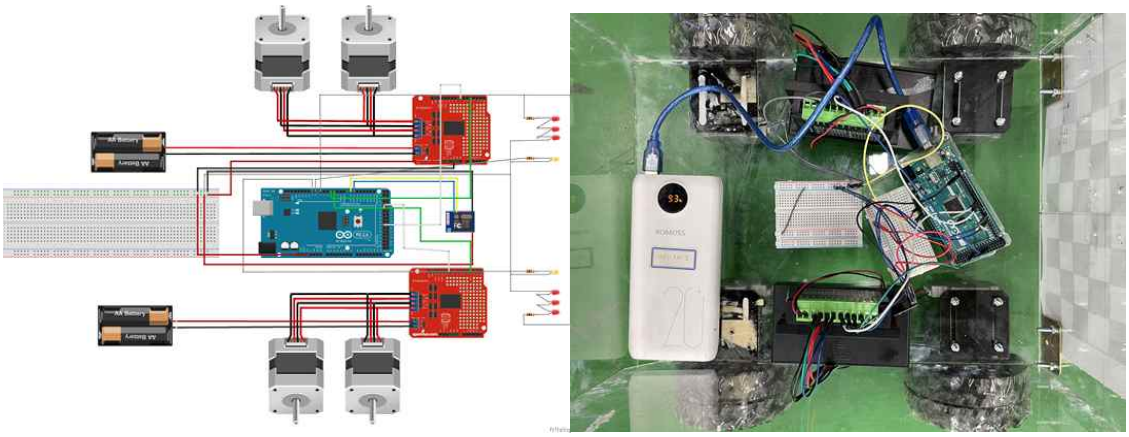
첫 번째, 기계 연결과 전기 회로 접속을 자동으로 구현하는데 어려움이 있었다.

두 번째, 분리결합부를 고안하더라도 안전성을 보장할 수 없었다.

세 번째, 프로토타입을 구현하기 위한 재료 및 부품 선택이 쉽지 않았다.

결론적으로 분리결합부를 설계보다는 분리 결합을 차량 간 정렬 구현에 더 초점을 두고 연구를 계속 진행하기로 하였다.

3. 차량 디자인



(a) 차량 내부 구조 도식화 (b) 실제 차량 내부 구조

[그림 3-1] 차량 내부 구조

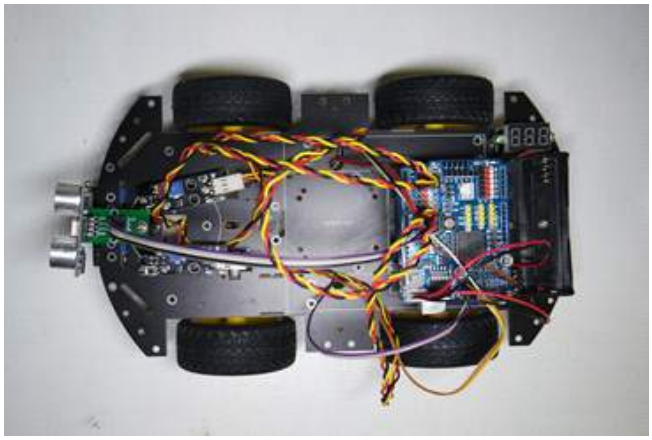
[그림 3-1(a)]은 차량 내부 구조를 도식화한 그림이다. 모터드라이버를 두 개를 설치하여 모터드라이버 한 개당 모터를 앞뒤로 두 개씩 제어하였고, ESP8266을 이용하여 무선통신을 하였다. 또한 9~15V를 필요로 하는 모터에 전압을 공급하기 위하여 1.5V AA건전지를 8개 연결하여 12V를 각각의 모터 드라이버에 공급하였다.

[그림 3-1(b)]는 [그림 3-1(a)]를 실제로 구현한 것이다. 이때, 아두이노에 공급되는 전압은 보조배터리를 활용하였다.

4. 시행착오

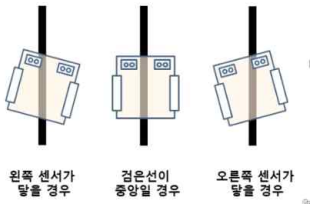
4.1 라인트레이서

4.1.1 라인트레이서를 활용한 차량 제어



[그림 4-1] 라인트레이서를 이용한 차량

분리 결합이라는 컨셉에 초점을 두기 위하여 초기에 차량은 자율주행이 아닌 라인트레이서를 이용하여 차량의 주행부분을 극히 제한하였다. 초기의 차량은 [그림 4-1]과 같이 4개의 적외선 센서와 1개의 초음파 센서를 이용한 라인트레이서를 사용하였다.



[그림 4-2] 라인트레이서의 동작 원리

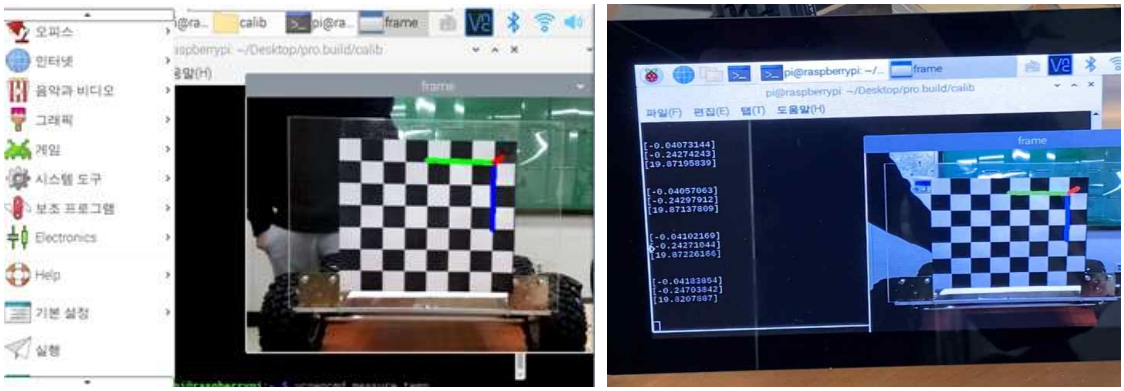
[그림 4-2]와 같이 하단부에 있는 두 개의 적외선 센서가 송신부에서 각각 초음파를 쏘면 바닥에 있는 검은선에 의해 그 적외선이 흡수된다. 따라서 수신부는 적외선을 받지 못하고 이를 통해 검은선만을 따라가는 방식이다. 또한 전면에 있는 적외선 센서와 초음파 센서를 사용하여 차량 앞의 장애물을 감지하였다.

4.1.2 문제점(라인트레이서)

적외선 센서를 사용하는 라인트레이서를 이용하여 구현한 차량은 상황에 따라 센서가 검은색 선을 인식하지 못하여 잘못된 방향으로 주행하는 경우가 많았다. 특히 빛에 의해 결과 값이 민감하게 변하였다. 또한 라인트레이서의 특성상 차량이 주행하는 데에 있어서 차량의 좌우 흔들림이 발생하여 이를 이용하여 정밀한 차량제어를 하기에는 불가능 하였다. 이러한 라인트레이서의 한계에 부딪혀 멘토님과의 상담을 통하여 해결방법을 모색하였다. 이후 영상처리를 이용해보라는 멘토님의 의견을 적극 수용하여 Opencv를 활용한 차량제어를 시작하였다.

4.2 라즈베리파이

4.2.1 라즈베리파이를 활용한 차량 제어



(a) 카메라 자세 추정 (b) Opencv의 결과 값

[그림 4-3] 라즈베리파이를 활용한 차량제어

영상처리를 사용하기로 결정한 이후, 어떠한 방법으로 차량을 제어할 수 있을지 찾아보았다. 도로를 인식하는 방법, 체스판을 인식하는 방법, 그리고 사물을 트래킹하는 방법 등 여러 가지가 있었는데 우리는 그 중에서 물체의 회전각을 계산할 수 있고 상대적으로 주변 환경에 의해 덜 좌우되는 체스판을 인식하여 이미지를 캘리브레이션 하는 방법을 선택하였다.

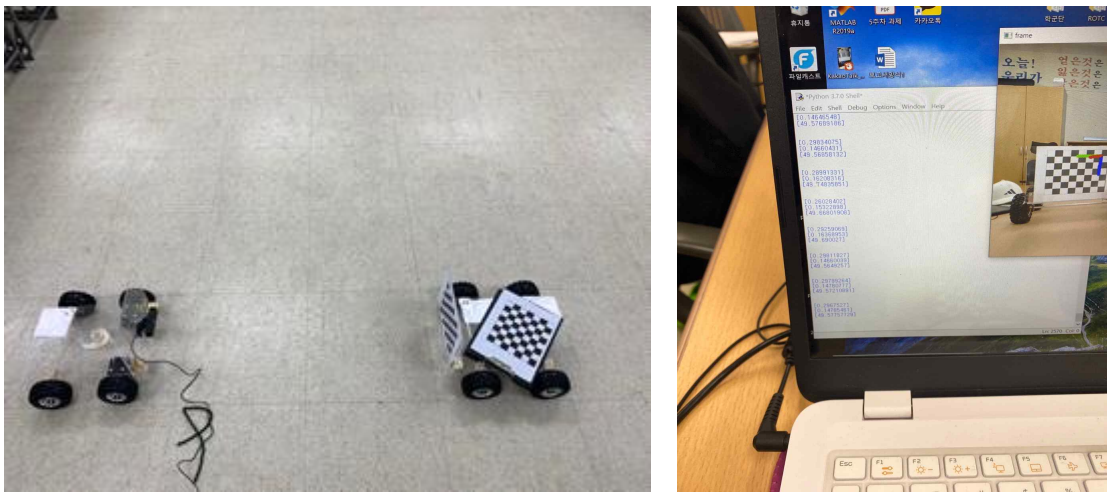
영상처리를 활용하기 위하여 먼저 영상처리를 수행할 수 있는 하드웨어가 필요하였다. 우리는 이를 위해 대표적으로 잘 알려진 싱글 보드 컴퓨터 라즈베리파이를 사용하여 영상처리를 수행하였다. 먼저 라즈베리파이에 Python을 설치하고 Python안에 Opencv 라이브러리를 설치하였다. [그림 4-3]은 차량 제어를 위한 Opencv 코드를 작성하고 이를 라즈베리파이에서 수행해본 결과이다. [그림 4-3(a)]의 파란색 초록색 빨간색선 각각 x축 y축 z축을 의미하며 빨간색 선은 카메라가 있는 방향을 기준으로 결정된다. [그림 4-3(b)]의 숫자들은 차량의 회전이나 이동에 따라 변하는 값들을 출력한 것이다. Opencv에 사용한 코드나 결과 값에 대한 설명은 밑에서 더욱 자세히 다룰 예정이다.

4.2.2 문제점(라즈베리파이)

라즈베리파이를 사용하여 영상처리를 수행한 결과, Opencv에서 작성한 코드에 대한 결과는 성공적으로 출력되었다. 하지만 무거운 영상처리를 수행하는 만큼 라즈베리파이 자체에 명확한 한계가 있었다. 처음 한두번은 정상적으로 작동되는 듯 하였으나 이후 실험값을 얻기 위해 지속적으로 코드를 실행한 결과, 라즈베리파이의 자체적인 온도가 상승하면서 코드 실행에 있어서 짧게는 2초에서 길게는 10초까지 delay가 발생하였다. 따라서, 우리는 라즈베리파이 대신에 노트북과 아두이노의 시리얼 통신을 이용하여 노트북에서 Opencv를 수행하고 이 결과 값을 아두이노에 전송하는 방식을 택하였다.

4.3 주행하는 차량

4.3.1 카메라 차량의 주행

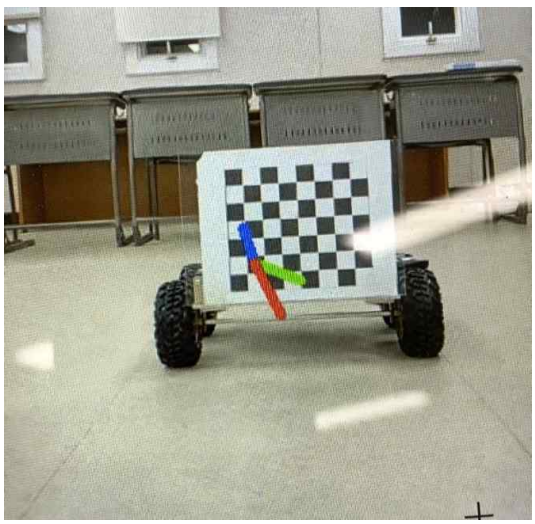


(a) 노트북에 연결된 카메라 (b) 노트북에서 Opencv를 수행한 결과

[그림 4-4] 노트북을 활용한 영상처리

위에서 발생한 라즈베리파이 과열 문제를 해결하지 못하여 결국 노트북을 사용하여 영상처리를 수행하였다. 주행하는 차량에 웹캠을 설치하고 이 웹캠을 USB 연장케이블을 사용하여 노트북과 길게 연결하였다. [그림 4-4(a)]는 카메라를 설치한 차량과 체스판을 설치한 차량의 모습이다. [그림 4-4(b)]는 노트북에서 Opencv를 실행하였을 때, 카메라가 인식하는 체스판의 모습과 그 결과 값을 출력하는 사진이다. 라즈베리파이에 의해 상대적으로 환경이 좋은 노트북을 통해 영상처리를 했기 때문에 앞에서 발생하였던 delay는 완벽히 해결된 모습을 볼 수 있었다.

4.3.2 문제점(카메라 차량의 주행)

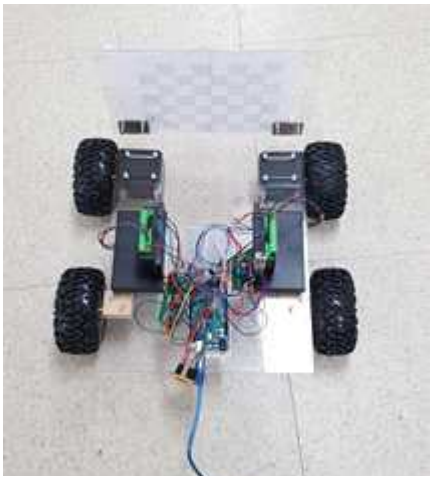


[그림 4-5] 체스판 인식 오류

Delay는 완벽히 해결하였으나 카메라 차량이 움직이면서 카메라가 체스판을 잘 인식하지 못하는 문제점이 발생하였다. [그림 4-5]는 차량이 회전하면서 체스판의 인식이 불안정해지는 모습이다. 이전의 실험에서는 카메라가 달린 차량을 매우 천천히 움직이거나 체스판이 설치된 차량을 움직여서 그

결과를 관찰하였다. 하지만 차량이 주행하기 위해서는 차량이 천천히 움직이는 상황만을 상정하여 설계할 수는 없었다. 이후 이 문제점을 해결하기 위해 여러 가지 방법을 생각해본 결과, 카메라가 움직이면 카메라의 각도에 따라 체스판의 움직임이 급변하지만 체스판이 움직인다면 카메라에서 상대적으로 그 변화가 훨씬 덜 하다는 것을 알았다. 따라서, 우리는 주행하는 차량을 카메라가 달린 차량에서 체스판이 달린 차량으로 변경하였다.

4.3.3 체스판 차량의 주행



[그림 4-6] 체스판 차량의 주행

[그림 4-6]은 카메라 차량의 주행에서 체스판 차량의 주행으로 바꾸어 차량을 재설계한 모습이다. 본격적으로 차량의 주행을 테스트하기 위해 1.5V AA건전지 8개를 연결하여 12V를 만든 배터리케이스 2개를 모터 드라이버 각각에 연결하고 모터 드라이버와 아두이노를 연결하였다. 카메라 차량의 카메라를 노트북에 연결하고 노트북에서 연산한 Opencv 결과 값을 시리얼 통신을 이용하여 아두이노에 보내어 체스판 차량을 주행하였다.

4.3.4 문제점(체스판 차량의 주행)

[그림 4-6]을 보면 모터가 앞바퀴 두 개에만 설치되어있는 것을 알 수 있다. 전력의 소모를 줄이기 위해서 모터를 두 개만 설치하고 뒷바퀴는 앞바퀴를 따라오는 형태로 설계하였으나 차량을 직접 주행시켜본 결과, 차량의 무게 때문에 앞바퀴가 제대로 굴러가지 않았다. 당시 사용하였던 모터의 무게가 약 1.2키로 였고, 모터드라이버의 무게와 배터리의 무게 또한 상당하였다. 따라서, 우리는 차량의 동력을 더 늘리기 위해 뒷바퀴에도 모터를 설치하기로 하였다.

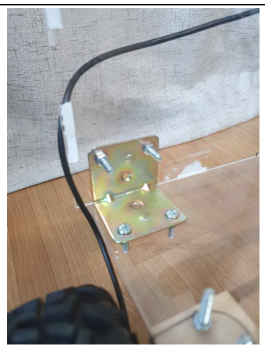
4.4 완성된 차량

차량의 외부구조

-카메라 장착 차량

[그림 4-7(a)] 웹캠 쪽 사진

카메라 캘리브레이션과 영상인식을 위한 웹캠을 장착하고 모터장착부 차량의 체스판 인식에 용이하도록 가장 최상단에 위치시켰다.



[그림 4-7(b)] ㄱ자 경첩 사진

차량의 앞면과 밑바닥면의 안정적인 고정을 위해 ㄱ자 경첩을 사용했다. 앞면과 밑바닥면의 아크릴에 구멍을 뚫고, ㄱ자 경첩과 나사로 체결해 단단하게 고정시켰다.

[그림 4-7(c)] 바퀴쪽 나무 사진

바퀴축과 차량의 밑면을 고정하면서 바퀴의 높이를 고려한 설치를 위해 나무판을 겹쳐 적정한 높이를 만들어주고 나사로 아크릴판과 체결했다.

[그림 4-7(d)] 바퀴쪽 ㄱ자 축 경첩

바퀴축이 흔들리지 않도록 축 지름에 딱 맞는 경첩을 설치하고, 바퀴축 끝부분에 바퀴를 고정했다.

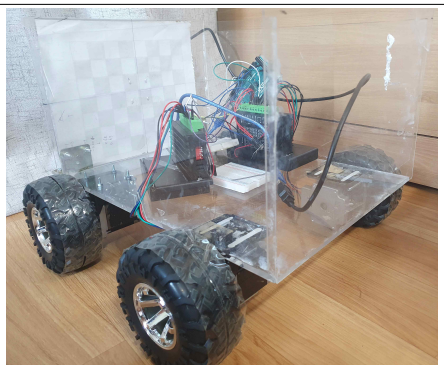
-모터 장착 차량

[그림 4-8(a)] 체스판 쪽 사진

영상인식에 사용되는 체스판을 상하좌우 정중앙을 맞춰 차량 앞부분에 부착했다.

[그림 4-8(b)] ㄱ자 경첩 사진

차량의 앞면과 밑바닥면의 안정적인 고정을 위해 ㄱ자 경첩을 사용했다. 앞면과 밑바닥면의 아크릴에 구멍을 뚫고, ㄱ자 경첩과 나사로 체결해 단단하게 고정시켰다.



[그림 4-8(c)] 옆면 뒷면 사진

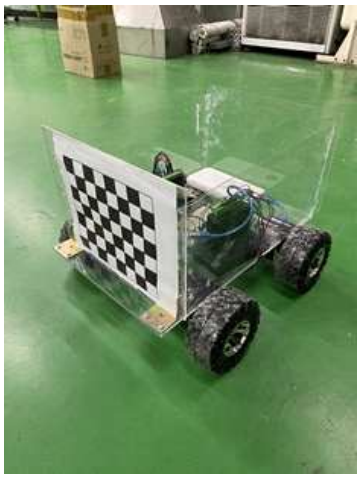
차량 내부에 아두이노와 모터드라이버 배터리 등의 부품을 위치시켜야하므로 옆면과 뒷면을 부착하여 안정성을 향상시키고 차량의 모습과 가깝게 완성했다.

[그림 4-8(d)] 차량하부 모터와 브라켓

밑바닥면의 아크릴과 모터를 연결시켜주는 브라켓을 설치하고, 4개의 바퀴 모두를 제어해야하므로 4개의 바퀴 모두 모터와 바퀴를 연결했다.

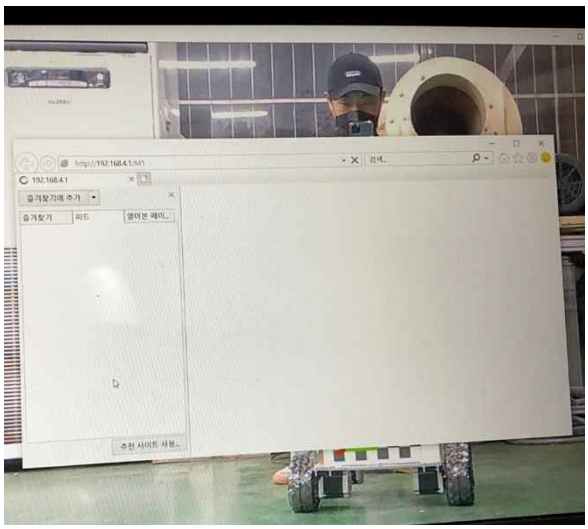
[그림 4-8(e)] 바퀴테이프사진

바퀴의 마찰력이 모터의 토크보다 커져 모터를 손상시키지 않도록 바퀴에 마찰이 없는 테이프를 부착하여 마찰을 줄이도록 했다.



[그림 4-9] 완성된 차량

[그림 4-9]은 2륜구동에서 뒷바퀴에도 모터를 설치하여 4륜구동으로 설계한 차량이다. 모터드라이버 두 개를 사용하여 4개의 모터를 제어하기 위해 앞바퀴와 뒷바퀴의 회전속도나 회전방향을 동일하게 설계하였다. 또한 시리얼 통신을 이용하였던 노트북과 차량 간의 통신을 와이파이 통신을 사용하여 무선통신을 구현하였다. 노트북에서 아두이노에 주던 전력 공급은 보조배터리를 사용하여 해결하였다. 이후, Opencv를 실행시켜 Opencv 결과 값을 무선통신으로 아두이노로 보냈고 아두이노는 그 값에 따라 정확한 주행을 하는 것을 확인하였다.



[그림 4-10] 주행 테스트

차량의 회전에 따라 노트북에서 아두이노로 M1, M2, M3 신호를 보내도록 설계하였다. [그림 4-10]은 차량의 주행을 테스트하는 과정 중 노트북 화면에서 볼 수 있는 영상이며, 차량을 왼쪽으로 회전하도록 하는 M1 신호가 출력되고 있는 것을 확인할 수 있다.

5. 차량제어 알고리즘

5.1 카메라 캘리브레이션(Opencv)

5.1.1 카메라 캘리브레이션이란

우리가 실제 눈으로 보는 세상은 3차원이지만 이것을 카메라로 찍으면 2차원의 이미지로 변하게 된다. 3차원의 점들이 이미지 상에서 어디에 맺히는지는 기하학적으로 생각하면 영상을 찍을 당시의 카메라의 위치 및 방향에 의해 결정된다. 이때, 카메라의 위치 및 방향을 카메라 외부 파라미터라고 하는데 실제 이미지는 사용된 렌즈, 렌즈와 이미지 센서와의 거리, 렌즈와 이미지 센서가 이루는 각 등 카메라 내부의 기구적인 부분에 의해서 변하기 때문에 외부 파라미터에 영향을 준다. 따라서 3차원 점들이 영상에 투영된 위치를 구하거나 역으로 영상좌표로부터 3차원 공간좌표를 복원할 때에는 이러한 내부 요인을 제거해야만 정확한 계산이 가능해진다. 그리고 이러한 내부 요인의 파라미터 값을 구하는 과정을 카메라 캘리브레이션이라 부른다. 즉, 캘리브레이션은 카메라 외부 파라미터를 구하기 위한 하나의 과정이라 할 수 있다.

5.1.2 카메라 캘리브레이션 코드 설명

```
# Criteria of chess board
termination = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 100)
objp = np.zeros((6*7,3),np.float32)
objp[:,:2] = np.mgrid[0:6, 0:7].T.reshape(-1,2)
objpoints=[]
imgpoints=[]
```

[그림 5-1] 체스판 인식 코드

[그림 5-1]은 체스판을 인식하는 코드이다. cv2.TERM_CRITERIA_EPS는 체스판을 찾기까지 반복 수행하는 횟수이며, cv2.TERM_CRITERIA_ITER은 체스판 인식의 정확도이다. 체스판의 정확도가 100에 도달할 때 까지 10회 반복하여 체스판을 찾는다.
np.zeros((6*7,3,np.float32) 는 체스판의 크기를 결정한다. 여기서는 6*7 체스판을 사용하였다.

```
# Video start
cap = cv2.VideoCapture(1)
cap.set(cv2.CAP_PROP_FRAME_WIDTH,1920)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT,1080)
count = 0
while True:
    #Make video to frames
    fet, frame = cap.read()
    #Make color of frames to gray
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

[그림 5-2] 카메라 실행 및 흑백화

[그림 5-2]는 카메라를 실행하고 이를 흑백화 하는 과정이다. 해상도는 1920x1080으로 설정하였다. cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)는 카메라를 통해 받은 프레임을 RGB에서 흑백으로 변환하는 것이다.

```
#Find corner of chess board
ret, corners = cv2.findChessboardCorners(gray,(6,7),None)
# If find corners of chess board
if ret:
    objpoints.append(objp)
    cv2.cornerSubPix(gray,corners,(11,11),(-1,-1), termination)
    imgpoints.append(corners)
    cv2.drawChessboardCorners(frame,(6,7),corners,ret)
    count += 1
    print('[%d]'%count)
#Show image of chess board corners
cv2.imshow('img',frame)
```

[그림 5-3] 체스판의 코너 인식 및 그림 출력

[그림 5-3]은 체스판의 코너를 인식하고 이를 그림으로 출력하는 과정이다. Opencv에는 체스판의 코너를 찾는 cv2.findChessboardCorners라는 자체 함수가 존재한다. 이를 이용하여 6*7 체스판의 코너를 찾고 이를 이미지로 표시하였다.

```
#Get the value of calibration
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
                                                   gray.shape[::-1],None,None)
#Save the value to .npz file
np.savez('calib.npz',ret=ret, mtx=mtx, dist=dist, rvecs=rvecs, tvecs=tvecs)
count = 0
print('It has been stored₩n')
```

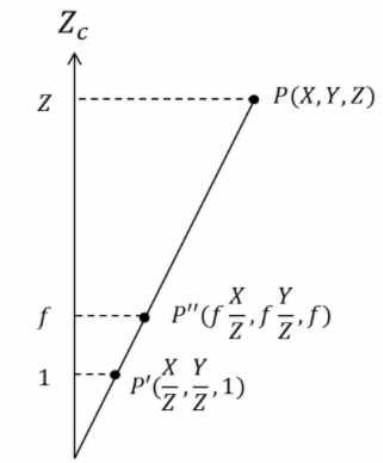
[그림 5-4] 카메라의 내부 파라미터 계산

[그림 5-4]는 카메라의 내부 파라미터를 계산하는 과정이다.
Opencv에는 objpoints와 imgpoints를 입력받아 내부 파라미터 ret, mtx, dist, rvecs, tvecs를 계산해주는 cv2.calibrateCamera라는 자체 함수가 존재한다. 이를 이용하여 내부 파라미터를 계산하고 이를 npz파일에 'calib'이라는 파일명으로 저장하였다.

5.2 카메라 자세추정(Opencv)

5.2.1 카메라 자세추정이란

앞에서 캘리브레이션을 통해 카메라 내부 파라미터를 구하였다. 카메라 자세추정은 2D 영상을 3차원으로 해석하는데 있다. 카메라를 중심으로 3차원 좌표계(카메라의 광학축 방향이 Z축, 오른쪽이 X축, 아래쪽이 Y축)를 설정했을 때, 외부의 한점 P(X,Y,Z)에 대한 영상좌표를 한번 생각해보자.



[그림 5-5] 2D 좌표계와 3D 좌표계 변환

[그림 5-5]에서 P(X,Y,Z)는 카메라로부터 (수직방향으로) Z만큼 떨어진 거리에 있는 점이므로 카메라로부터의 거리가 1인 가상의 정규 이미지 평면에서의 좌표는 삼각형의 닮음비를 이용하면 P'(X/Z,Y/Z,1)이 됨을 쉽게 알 수 있다. 그런데 실제 카메라 영상은 초점거리 f만큼 떨어진 이미지 평면에 투사되므로 P에 대응되는 이미지 좌표는 P''(f*X/Z, f*Y/Z, f)가 되고 이를 픽셀좌표계로 변환하면 우리가 원하는 최종 영상좌표를 얻을 수 있다.
하지만 Opencv에서는 이러한 복잡한 계산과정을 solvePnP라는 함수로 모두 대신할 수 있다. solvePnP 함수는 카메라의 내부파라미터를 넣으면 자동으로 외부파라미터를 계산해준다.

5.2.2 카메라 자세추정 코드 설명

```
#Bring the value of .npz file
with np.load('calib.npz') as X:
    ret, mtx, dist, _, _ = [X[i] for i in ('ret', 'mtx', 'dist', 'rvecs', 'tvecs')]
# Iteration and Accuracy for find 6x7 chess board
termination = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
objp = np.zeros((6*7,3),np.float32)
objp[:,:2] = np.mgrid[0:6, 0:7].T.reshape(-1,2)
axis = np.float32([[3,0,0], [0,3,0], [0,0,-3]]).reshape(-1,3)
objpoints=[]
imgpoints=[]
```

[그림 5-6] npz파일 불러오기

[그림 5-6]은 캘리브레이션에서 저장한 카메라 내부 파라미터가 들어있는 npz파일을 불러오고 다시 체스판을 찾는 과정이다. 코드의 delay를 최대한 낮추기 위하여 정확도를 낮추는 대신 반복횟수를 늘렸다.

```
cap=cv2.VideoCapture(1)
# Frame size 1920x1080
cap.set(cv2.CAP_PROP_FRAME_WIDTH,1920)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT,1080)
while True:
    ret, frame = cap.read()
    current_time = time.time() - prev_time
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray,(6,7),None)
```

[그림 5-7] 카메라 실행 및 체스판의 코너 찾기

[그림 5-7]은 카메라를 실행하고 체스판의 코너를 찾는 과정이다. 카메라의 해상도는 전과 동일하게 1920x1080으로 설정하였다.

```
if (ret == True) and (current_time>1./FPS):  #For control frame of camera
    prev_time = time.time()
    cv2.cornerSubPix(gray,corners,(11,11),(-1,-1), termination)
    #Find the rvecs tvecs of chess board in video
    _, rvecs, tvecs, inliers = cv2.solvePnPRansac(objp, corners, mtx, dist)
    imgpts, jac = cv2.projectPoints(axis, rvecs, tvecs, mtx, dist)
    #Draw the axis
    frame = draw(frame, corners, imgpts)
    cv2.imshow('frame',frame)
```

[그림 5-8] 카메라 외부 파라미터 계산 및 축 그리기

앞에서 언급하였던 solvePnP라는 함수를 이용하여 카메라의 외부 파라미터를 계산하였다. cv2.solvePnPRansac은 체스판의 코너와 카메라의 내부 파라미터를 입력받아 카메라의 외부 파라미터를 계산해주는 Opencv 자체 함수이다. 외부 파라미터는 rvecs와 tvecs가 있는데, rvecs는 회전변환 벡터이며, tvecs는 평행이동 벡터이다.
이후, 이 외부 파라미터를 이용하여 체스판과 카메라 사이의 좌표계를 그렸다.
여기서 current_time>1./FPS는 카메라의 프레임을 조절하는 것이다. 와이파이 신호를 받아 움직이는 모터의 동작 시간을 감안하여 프레임을 1초에 1번씩만 받았다. 이를 통해 노트북이불필요한 프레임을 받아 계산하는 과정을 줄였다.

```
# Function of Drawing the x axis blue, y axis green, z axis red
def draw(img, corners, imgpts):
    corner = tuple(corners[0].ravel())
    img = cv2.line(img, corner, tuple(imgpts[0].ravel()), (255,0,0), 5)
    img = cv2.line(img, corner, tuple(imgpts[1].ravel()), (0,255,0), 5)
    img = cv2.line(img, corner, tuple(imgpts[2].ravel()), (0,0,255), 5)
    return img
```

[그림 5-9] 카메라와 체스판 사이의 좌표계를 그리는 함수

x축은 파란색 선, y축은 초록색 선으로 나타내어 체스판의 각도를 나타내었고 z축은 빨간색 선으로

카메라의 위치를 나타내었다.

```
# Send Signal to arduino for control motor movement

if ((rvecs[0]<-0.18) or (rvecs[1]>0.18)) and (tvecs[2]>80):

    webbrowser.open('192.168.4.1/M1',new=1)
    time.sleep(1)
    os.system("taskkill /im iexplore.exe /f")
    print("M1")


elif((rvecs[0]>0.18) or (rvecs[1]<-0.18)) and (tvecs[2]>80):

    webbrowser.open('192.168.4.1/M2',new=1)
    time.sleep(1)
    os.system("taskkill /im iexplore.exe /f")
    print("M2")


elif(rvecs[0]<0.18) and (rvecs[0]>-0.18) and (rvecs[1]<0.18) and (rvecs[1]>-0.18) and (tvecs[2]>80):

    webbrowser.open('192.168.4.1/M3',new=1)
    time.sleep(1)
    os.system("taskkill /im iexplore.exe /f")
    print("M3")


# Minimum distance to recognize chessboard with camera

elif(tvecs[2]<80):
    print("Minimum distance")
    webbrowser.open('192.168.4.1/M4',new=1)
    time.sleep(1)
    os.system("taskkill /im iexplore.exe /f")
    print("M4")
```

[그림 5-10] 모터 제어를 위해 아두이노에 신호를 보내는 코드

앞에서 언급했듯이 rvecs는 회전변환 벡터이며 tvecs는 평행이동 벡터이다. 즉, rvecs는 체스판의 회전을 나타내며, tvecs는 체스판과 카메라와의 거리를 나타낸다고 할 수 있다. 여러번의 실험을 통하여 rvecs는 rvecs[0], rvecs[1], rvecs[2]로 이루어져 있으며, 그 중에서도 rvecs[0]와 rvecs[1]가 회전 각도에 직접적인 연관이 있다는 것을 알아냈다. 또한 tvecs도 tvecs[0], tvecs[1], tvecs[2]로 이루어져 있는데, tvecs[2]만 거리와 직접적인 관계가 있다는 것을 발견하였다. 따라서 rvecs[0]와 rvecs[1] 그리고 tvecs[2]만을 사용하여 모터를 제어하였다.

rvecs[0]와 rvecs[1]이 회전각과의 직접적인 연관이 있다는 것을 발견한 이후, 어떤 값에서 차량을 회전시키면 될지 결정하기 위해 여러 번의 실험을 하였다. 그 결과, 알아낸 것은 rvecs[0]와 rvecs[1]는 체스판과 카메라와 일직선상에 있으면 0에 가까웠고, 체스판이 회전할수록 절대값이 점점 커진다는 것이었다. 따라서 우리는 rvecs의 절대값이 0.18을 초과하면 모터가 회전하여 체스판을 다시 일렬로 정렬시키기로 하였다.

[그림 2-10]의 위 세 개 if문은 그 코드에 해당된다. 또한, 체스판을 설치한 차량이 카메라에 매우 가까워지면 카메라가 체스판을 제대로 인식하지 못하는 상황이 발생한다. 따라서, tvecs[2]의 값에 제한을 두어 일정 값 이하에서는 멈출 수 있도록 하였다.

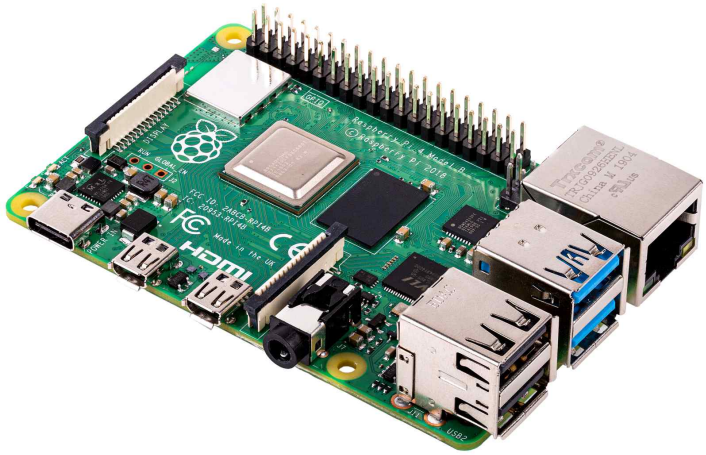
여기서 webbrowser.open은 python에서 webbrowser를 여는 코드이다. 아두이노와의 와이파이 통신을 위해 192.168.4.1 라는 주소를 할당하였다. Opencv에서는 상황에 따라 그 주소를 열어 차량을 제어한다.

192.168.4.1/M1은 차량을 왼쪽으로 회전시키는 주소이며, 192.168.4.1/M2은 오른쪽으로 회전시키는 주소이고, 192.168.4.1/M3는 직진하는 주소이다. 마지막으로 192.168.4.1/M4는 차량을 정지시키는 주소이다.

5.3 라즈베리파이

5.3.1 설정 과정

초기 프로토타입의 구상 과정에서는 완전한 독립 임베디드 시스템으로 동작하는 차량의 설계를 구상하였다. 따라서 설계 과정에서 PC의 사용은 완전히 배제하게 되었고, 이에 따라 미니 임베디드 컴퓨터인 라즈베리파이를 해결점으로 선택하게 되었다.



[그림 5-11] 라즈베리파이

설계 과정에서 pin 수가 부족한 라즈베리파이를 보완하기 위해 아두이노를 USB를 이용한 Serial Communication으로 연결하여 OpenCV를 이용한 프로그램 연산은 라즈베리 파이에서, 모터의 동작은 아두이노로 동작하려고 하였다.

raspbianOS의 설치와 opencv의 설치를 완료하고 프로그램을 돌리며 결과의 출력을 확인해보았다. 하지만, Opencv 에서 언급을 하였듯, 부팅 초기에는 정상적으로 작동을 하였지만, 발열로 인한 성능저하로 불규칙하게 delay가 발생하게 되었다. 이를 해결하기 위해 fps의 값도 낮추고 영상의 해상도도 낮추었지만 해결이 되지 않았다. 따라서 외부 장치인 fan을 이용하여 발열을 잡으려고 하였으나, 성능에 조금 개선이 되었을 뿐 delay를 예측할 수 없었다.

따라서 모터를 직접 라즈베리파이에 연결하여 설계를 진행하려고 하였으나, 설계 초반인 점을 고려하여 앞으로 얼마나 많은 핀을 추가적으로 사용할지 예측이 불가능한 상황이었기 때문에 선택을 하지 않게 되었다.

5.3.2 대안 설정

더 고사양의 미니 임베디드 컴퓨터를 구매하여 제품에 연결을 하여 문제점을 해결하려고 하였다. 하지만, 어느정도 성능 이상의 제품을 구매하여야 우리가 원하는 방향대로 동작을 안정적으로 할지 알 수 없었다. 또한, 제품의 프로토타입은 아무리 완성품이 아니라고 하더라도 어느정도 이상의 완성도가 있어야 한다고 생각을 했다. 따라서 우리는 설계의 과정에서 PC를 사용하여 고성능을 필요로 하는 OpenCV의 연산을 진행하기로 하였다. PC와 아두이노의 연결은 라즈베리파이를 사용한 것과 마찬가지로 유선 연결을 하려 하였으나, 이는 위에서 언급한 완성도의 측면에서 상단한 오점을 남겼다. 따라서 우리는 이를 해결하기 위해 별도의 Wifi module을 사용하여 모터의 동작을 제어하기로 하였다.

5.4 통신

5.4.1 아두이노 Mega 사용 이유

설계 과정에서 사용한 MCU(microprocessor control unit)인 아두이노 Mega는 시리얼 통신이 4개까지 가능하다. 이는 Hardware Serial Communication 핀이 1개만 존재하는 아두이노 UNO보다 더 많은 핀을 확보한 것이다. USB 연결을 통한 Serial Communication은 Serial 0핀과 연동이 되어 USB를 연결하여 아두이노와 통신을 하는 경우, 사용이 불가하다. 따라서 UNO를 사용하여 설계를 진행하는 경우, software serial communication의 방법으로 기타 MCU와 소통을 진행하여야 하고, 이는 hardware communication보다 성능이 떨어지는 방법이므로 선택을 하지 않았다.

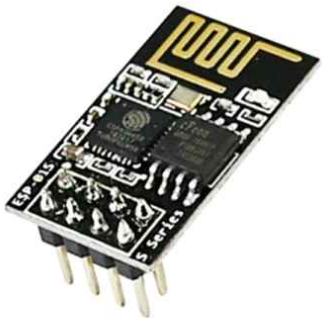
[그림 5-12] 아두이노 Mega 2560

위의 이유로 프로토타입의 제작 과정에서 Mega의 Serial1, Serial2, Serial3 사용을 하여 기타 아두이노 또는 기타 MCU와 통신을 할 수 있도록 여유롭게 설계를 진행하였다. 추후 코드의 설명을 진행하겠지만, Python의 연산 결과에 따른 모터의 동작을 제어하기 위하여 ESP8266 Wifi Module을 사용을 하였는데 이는 Serial3을 이용하여 USB를 이용한 통신 Serial0과의 거리를 확보하여 혹시 모르게 발생할 기기적 오류로부터 조금 더 유연하게 대처할 수 있도록 설계를 진행하였다.

5.4.2 아두이노 보드 업로드

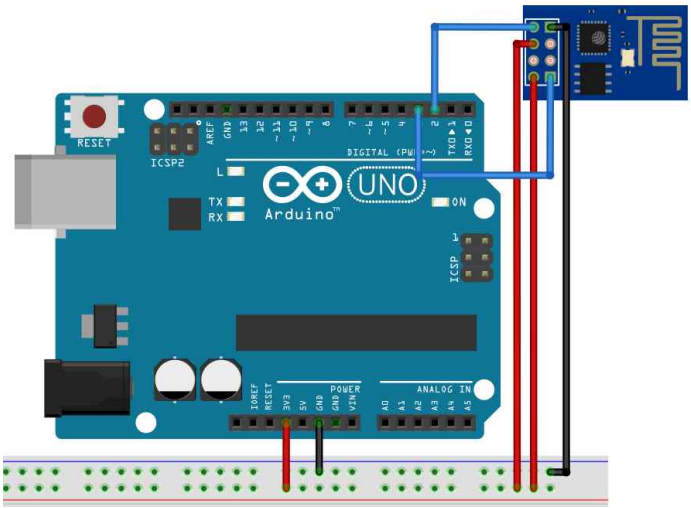
아두이노는 MCU로 프로그래밍을 진행한 후 다른 프로그램을 업로드 하지 않으면 최종 업로드를 진행한대로 동작을 하게 된다. 따라서 PC에서 ARUINO IDE프로그램을 사용하여 모터, WIFI module 관련 코드를 업로드 진행하였다. 이 과정은 UART(Universal asynchronous receiver/transmitter) 통신 프로토콜을 사용한다. 이후의 동작은 아두이노의 동작 원리에 따라 setup과 loop로 나뉘어 setup은 초기에 필요한 필수 세팅 정보, loop는 동작을 반복하는 부분으로 나뉘어 설계를 진행한다.

5.4.3 ESP8266 (ESP-01) 모듈 코드



[그림 5-13] ESP8266 모듈

프로토타입 제작 과정에서 사용한 ESP8266 모듈이다. 커넥터를 따로 구입을 진행하지 않아, 아두이노 UNO를 사용하여 baud rate를 9600으로 조절하고, Mega에 연결하여 Python 연산의 결과를 Serial Communication으로 받아들여 코딩을 통해 저장되어있는 모드별 동작에 따라 모터가 동작할 수 있도록 설계를 진행하였다. baud rate는 9600외에 더 높은 속도로 동작하게 하여 딜레이를 줄일 수 있었지만, 정확도와 속도는 trade-off 관계에 있기 때문에 더 빠르게 동작하는 것보다는 더 안정적으로 코드가 동작하는 것이 더 좋은 방향이라고 판단을 하였다. 따라서, baud rate는 9600으로 설정을 하였다. [그림 5-14]은 ESP 8266모듈의 설정을 위해 Arduino Uno와 연결을 진행한 결선도이다.



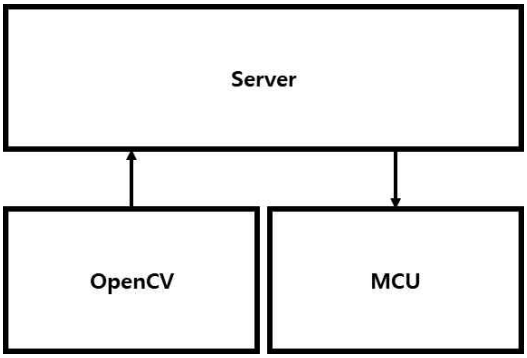
[그림 5-14] 모듈과의 결선도

```
//Wifi setting
    sendData("AT+RST\r\n",2000,DEBUG);
    sendData("AT+CWMODE=2\r\n",1000,DEBUG); // configure as access point (working mode: AP+STA)
    sendData("AT+CWSAP=\"AVSC_CAR\",\"avscpassword\",11,3\r\n",1000,DEBUG); // make AP with Password
    sendData("AT+CIFSR\r\n",1000,DEBUG); // get AVSC_CAR ip address
    sendData("AT+CIPMUX=1\r\n",1000,DEBUG); // configure for multiple connections
    sendData("AT+CIPSERVER=1,80\r\n",1000,DEBUG); // turn on server on port 80
```

[그림 5-15] 아두이노 Mega의 Setup부 코드 내용

[그림 5-15]은 아두이노 부팅시(초기 차량의 시동시) setup문을 통해 초기화를 진행하는 부분의 코드이다. setup부에서 초기화를 진행한 이유는, 실험적인 data를 통해 결정한 내용이다. setup을 하지 않고 실행을 진행한 결과, 모듈이 이상하게 작동하는 경우가 있었기 때문이다. 이 문제점은 opencv 연산의 결과는 wifi module을 통해 정상적으로 전달이 되고 있지만, 아두이노에는 전달되지 않는 아주 치명적인 이상 행동이었다. 따라서 우리는 동작의 안정성을 위해 차량의 시동이 켜지는 경우 다소 시간이 걸리더라도 안정성을 위해 모듈을 초기화 하고 동작에 사용을 하는 것이 옳다 판단했다.

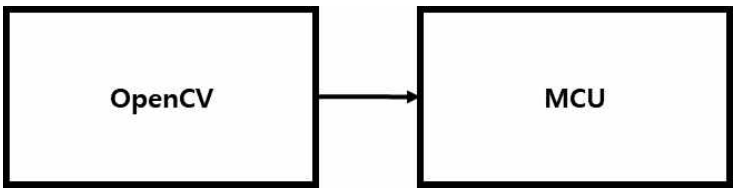
또한, 3번째 줄의 "sendData("AT+CWSAP=₩"AVSC_CAR₩",₩"avscpassword₩", 11,3₩r₩n" ,1000,DEBUG);"는 모듈의 이름 및 비밀번호를 설정하는 부분이다. 처음 계획한 것은 App에서 초기 부팅 시, 사용자의 입력 값을 받아들여 차량의 이름 및 모듈의 비밀번호를 설정할 수 있는 부분을 설계하는 것이었다. 하지만, 해당 부분을 맡아 설계를 진행하려고 한 학우의 연락두절로 인해 어쩔 수 없이 미래 확장성만을 열어둔 채 모듈의 이름 및 비밀번호는 고정값으로 두게 되었다.



[그림 5-16] 클라우드 통신

OpenCV의 연산 결과를 wifi 모듈을 사용하여 받아들이는 것에는 크게 2가지 방법이 있었다. 첫 번째로 , [그림 5-16]와 같이 모듈을 서버화 하여 서버에 원하는 값을 업로드 한 후, 이를 MCU로 전달하는 방법이다.

처음에는 이 방법으로 설계를 진행하려 하였으나, 미래의 공유 자동차 플랫폼을 예상하고 이에 대응하는 모듈형 자동차를 만드는 엔지니어로서 서버에 업로드하여 이 값을 불러와 차량이 움직이게 하는 경우, 발생할 수 있는 문제점은 크게 두가지였다. 첫 번째는 통신의 안정성에서 발생하였다. 통신은 회사가 가진 서버가 해결할 수 없는 문제이다. 자연재해, 주변 지형 등 다양한 곳에서 간섭이 발생할 수 있고, 원인 모를 딜레이가 발생할 수 있다. 사용자가 차량에 탑승하고 있을 때 딜레이가 발생한다면 큰 인명 사고로 이어질 수 있기 때문에 위험하다고 판단했다. 두 번째 발생할 문제점은 플랫폼의 문제점에 있었다. 사업가로서는 최대 이윤을 위해서는 차량에 공유 플렛폼 용 서버와 차량 통신용 서버 2개를 증설하지 않고, 하나의 서버에서 통신을 진행할 것이다. 따라서, 서버를 이용하는 경우 클라우드를 이용한 공유 플랫폼이라는 점에서 서버가 해킹을 당하는 경우 차량의 안전을 보장할 수 없다는 판단을 하였다.



[그림 5-17] 1대 1 통신

위에서 생각한 이유로, 우리는 모듈을 호스트화 하여 MCU와 OpenCV의 결과값이 1대1로 통신을 할 수 있게 설정을 하였다. 이에 대한 자세한 내용은 OpenCV 설명에서 나와있듯이 ESP8266모듈이 가지는 고유의 서버 주소인 192.168.4.1에 원하는 모드 값을 업로드 함으로써 ESP8266모듈이 이 값을 MCU에 전달할 수 있도록 설계를 진행하였다. 보안을 위해서는 별도의 암호화 및 복호화의 과정이 있으면 더 완벽한 보안이 되었을 것이다. 이에 대응하기 위해서 1개의 아두이노 MCU을 더 사용하여 암호화를 하고, 암호화된 데이터를 복호화 하여 작동이 될 수 있게 설계를 진행하려고 하였으나, 이는 결합부 안전성에 집중하자는 프로젝트의 취지와는 많이 벗어나는 내용인 것 같아서 추가하지 않았다.

```
//Serial Communication with Wifi Module
String sendData(String command, const int timeout, boolean debug) {
    String response = "";
    Serial3.print(command);
    long int time = millis();

    while( (time+timeout) > millis()) {
        while(Serial3.available()) {
            char c = Serial3.read();
            response+=c;
        }
    }

    if(debug) Serial.print(response);
    return response;
    }
```

[그림 5-18] wifi module과의 통신을 위한 함수

아두이노로 데이터를 전송하기 위해서는 AT command를 사용하여 전송을 해 주어야 한다. 아두이노에서 상태를 wifi module로 전송을 하는 경우 딜레이가 1~2초정도 발생하므로, 우리는 프로젝트에서 아두이노의 상태를 PC나 외부 장치로 확인을 해야 하는 과정이 필요한지에 대한 고민을 진행하였다. 이 과정이 필요 없다고 판단을 하고 생략을 진행하였다.
모듈을 초기 설정하는 과정에서 AT Command를 하나하나 시리얼 모니터에 입력하는 것은 많은 불편함을 야기하고, 독립적으로 작동이 되어야 하는 MCU에 외부 장치를 연결하는 것은 프로토타입과의 거리가 멀다고 생각을 하였다. 따라서 우리는 sendData 함수를 이용하여 AT command를 setup시에 AT Command와 함께 우리가 원하는 동작 코드로 작동할 수 있도록 설계를 진행하였다.

5.5 모드별 동작 코드

```
if (Serial3.available()) {
    if (Serial3.find("+IPD,")) {
        income_wifi = Serial3.readStringUntil('\r');
        String wifi_temp = income_wifi.substring(income_wifi.indexOf("GET /")+5, income_wifi.indexOf("HTTP/1.1")-1);

        //Get signal "M1" from Python
        if(wifi_temp == "M1") {
            Serial.println(wifi_temp);
            digitalWrite(dirPin_L, HIGH);
            digitalWrite(dirPin_R, HIGH);
            //Turn on LEDs for "Turn Signals"
            digitalWrite(RedLight,LOW);
            digitalWrite(YelLight_L,HIGH);
            //Move to Left
            for (int i = 0; i < stepsPerRevolution_Turn; i++) {
                for(int j = 0; j < 5; j++){
                    digitalWrite(stepPin_L, HIGH);
                    delayMicroseconds(rotate_speed_DOWN);
                    digitalWrite(stepPin_L, LOW);
                    delayMicroseconds(rotate_speed_DOWN);
                }
                digitalWrite(stepPin_R, HIGH);
                delayMicroseconds(rotate_speed_down);
                digitalWrite(stepPin_R, LOW);
                delayMicroseconds(rotate_speed_down);
            }
              //Turn on LEDs for "Break Signals"
              digitalWrite(RedLight,HIGH);
              digitalWrite(YelLight_L,LOW);
        }
```

[그림 5-19] M1 동작 코드

[그림 5-19]은 ESP8266 모듈을 통해 mode control signal이 들어왔는지 유무를 판단하고, 공백까지의 텍스트를 wifi_temp에 읽어들이고 이 string을 통해 모드를 분주시킨 내용이다. 앞서 설명하였듯, wifi module을 Serial3에 연결을 하였기 때문에 우선 Serial3.available을 통해 Serial3에 신호 변화가 있는지 확인을 하였다.

wifi_temp의 값이 M1인 경우, 양측 모터의 구동이 이루어져야 한다. 따라서, 스테핑 모더의 회전 속도 및 회전 각을 결정하였다.

프로토타입 제작 과정에서 또다른 문제점이 발생하였다. 이는, MCU의 기기적인 한계에서 발생한 문제였다. 아두이노 Mega는 기본적으로 동작이 긴 스태핑 모터를 한번에 하나만 control할 수 있었다. M1은 좌회전 제어 모드로 두가지 모터를 동시에 움직여야 하는 차량 조향 제어의 특성 상 한번에 하나의 모터만 움직이는 문제는 매우 치명적이었다. 따라서 우리는 for loop를 2중으로 사용하고, 시간 간격을 최대한 적게 두어 좌측과 우측의 모터의 회전수에 차이를 두면서도 최대한 연속적으로 움직일 수 있게 만들었다. 좌회전 제어를 위해서 좌측, 우측 모터의 방향을 동일하게 시계방향으로 설정하여 제어할 수 있었다.

이 원리를 이용하여 소스코드를 작성하였고, 그 결과 만족할만한 수준의 좌측 조향 동작을 보여주게 되었다.

또한 최대한 실제 차량과 같은 동작을 보여주기 위해서 후미등과 방향 지시등을 설계 진행하였다. 이는 결합부 작동 시 주변에 위치한 사람들이 차량의 이동을 인지하고 혹시 발생할 사고를 막기 위한 기본적인 설계라고 생각을 하였다.

[그림 5-20]은 M2 신호에 따른 모터의 동작을 설명한 내용이다.

```
else if(wifi_temp == "M2") {
     Serial.println(wifi_temp);
     digitalWrite(dirPin_L, LOW);
     digitalWrite(dirPin_R, LOW);
     //Turn on LEDs for "Turn Signals"
     digitalWrite(RedLight,LOW);
     digitalWrite(YelLight_R,HIGH);
     //Move to Right
     for (int i = 0; i < stepsPerRevolution_Turn; i++) {
         for(int j = 0; j < 5; j++){
              digitalWrite(stepPin_R, HIGH);
              delayMicroseconds(rotate_speed_DOWN);
              digitalWrite(stepPin_R, LOW);
              delayMicroseconds(rotate_speed_DOWN);
         }
         digitalWrite(stepPin_L, HIGH);
         delayMicroseconds(rotate_speed_down);
         digitalWrite(stepPin_L, LOW);
         delayMicroseconds(rotate_speed_down);
     }
        //Turn on LEDs for "Break Signals"
        digitalWrite(RedLight,HIGH);
        digitalWrite(YelLight_R,LOW);
}
```

[그림 5-20] mode2 일때의 모터 동작 코드

각 모드의 시작에 RedLight를 off 시킨 이유는 모드의 입력을 받아 차량이 동작하는 경우는 다시 주행이 시작되는 경우이므로, 브레이크등이 꺼지고 방향 지시등(YelLight)가 켜져야 주변에 위치한 사용자들이 안전할 것이라 판단하였다.

mode2(M2)는 우회전이 필요한 경우이다. 이 부분의 설계는 M1과 반대로 좌측, 우측 모터의 방향을 동일하게 반시계방향으로 설정하여 방향이 좌측을 향할 수 있게 만들었다. 이는 mode1(M1)의 설계와 마찬가지로 for loop을 사용하여 설계를 진행하였으며 원하는대로 동작을 확인하였다.

```
//Get signal "M3" from Python
else if(wifi_temp == "M3") {
    digitalWrite(dirPin_L, LOW);
    digitalWrite(dirPin_R, HIGH);
    digitalWrite(RedLight,LOW);
    //Move forward
    for (int i = 0; i < stepsPerRevolution; i++) {
        digitalWrite(stepPin_L, HIGH);
        delayMicroseconds(rotate_speed);
        digitalWrite(stepPin_L, LOW);
        delayMicroseconds(rotate_speed);
        digitalWrite(stepPin_R, HIGH);
        delayMicroseconds(rotate_speed);
        digitalWrite(stepPin_R, LOW);
        delayMicroseconds(rotate_speed);
    }
    digitalWrite(RedLight,HIGH);
}
```

[그림 5-21] mode3 동작 코드

mode3의 경우 차량의 조향이 필요 없이 좌, 우측 모터가 같은 회전수의 회전을 진행하는 알고리즘을 사용하여 설계를 진행하였다. 왼쪽과 오른쪽의 모터에 같은 회전수를 주고, 시간 간격을 최대한 적게 부여하여 연속적인 직진 동작을 확인하였다.
M3의 경우, 전방 주행이 종료되면 후미등을 동시에 동작하게 함으로써 전방 주행 과정이 마무리되고 후의 동작 처리를 위해 차량이 대기중인 상태임을 보여주었다.

3. 연구 수행 결과의 관련 분야 기여도 및 활용계획

본 연구의 활용계획은 다음과 같다.

・기대효과

- 좁은 땅에 비해 비약적으로 커지는 도시 때문에 야기되는 거주자들의 이동 효율성 저하 해소
- 모터엔진을 기반으로 한 Electrical Vehicle의 형태로 배출가스를 내뿜지 않아 도심 공해 해소
- 여러 대의 결합을 통해서 자율 군집 운행으로 교통과 물류 산업에 혁신적인 변화
- 결합 후 차량 간 배터리 공유 기능으로 배터리 방전으로 인한 전기차 안전성 향상
- 목적에 따라 차량을 생산 가능하기 때문에 배터리 충전용 모듈 자율차를 생산해서 차량들을 따라다니며 무선으로 전력을 공급하는 기능 구현
- 탑승자의 취향과 목적에 따라 분리 결합을 통해서 맞춤형 이동수단 및 공간 제공 가능
- 목적에 의해서 분리 결합을 통해 다양한 형태와 기능 활용
- 이동하는 대중교통 및 모빌리티 환승 거점 허브로 사용
- 다양한 활용성을 통해 현재의 모빌리티 구조가 아닌 새로운 유기적인 모빌리티 생태계를 구축
- 새로운 모빌리티 생태계로 인해 도로가 변화함에 따라 녹지 환경도 새롭게 탈피
- 새로운 기술을 바탕으로 사람과 환경이 어우러진 스마트하고 지속 가능한 도시 유지

• 특허

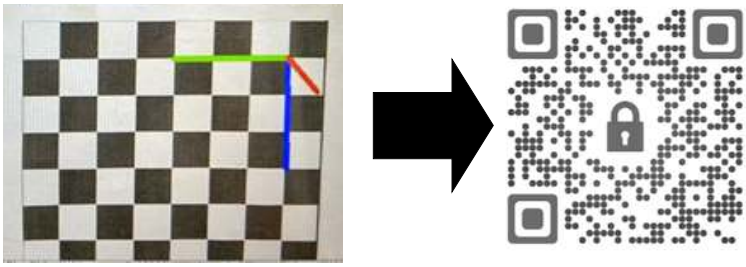
- 자동차 분리 결합형 시스템 특허

• 추후 연구

- 결합 여부에 따른 엔진 공유기능 연구
- 전후방 뿐 모듈 분리/결합으로 용도에 따른 맞춤형 모듈 차량 개발 연구

- 전후방뿐만 아니라 상하/좌우 6방향 결합방법 연구와 개념을 확장시켜 차뿐만 아닌 집과의 분리/결합 연구

- 결합 주행 시 마찰을 줄이기 위해 불필요 바퀴 스페어 기술 연구
- 위급 상황 시 차량을 분리하여 인명피해 최소화 방법 연구
- 분리/결합 시 완충 시스템 보강 연구
- 분리/결합 시 체스판 영상인식을 QR코드 인식과 접목시켜 보안성 및 안정성 향상 연구



- 연구팀 예산 사용 내역서 작성 안내 -

1. 세목별 용도

예산세목	연구장비/재료비	연구활동비	연구과제추진비
용도	재료비	내부시험분석의뢰	회의비
	장비임차료	외부시험분석외릐	국내여비(시내여비)
		문헌구입비	사무용품비
		인쇄/복사비	
		학회/세미나 참가비	
		국내외훈련비	
		연구실환경유지비	

2. 작성예시

(단위: 원)

예산세목	용도	구매품명	금액 (VAT포함)
연구 장비/재료비	재료비	라즈베리키트3	70,000
	재료비	Beaker,Corning(1000-50)	50,000
	재료비	발열패드	52,800
	장비임차료	노트북대여	500,000
연구 활동비	내부시험분석의뢰	공동기기원(원소분석기)	90,000
	외부시험분석의뢰	서울대학교기초과학공동기기원(ICP-AES)	115,500
	문헌구입비	열혈 C 프로그래밍 외 1권	79,000
	인쇄/복사비	연구자료 인쇄	30,000
	연구실환경유지비	이동형 투명유리칠판	200,000
	국내외훈련비	온라인강의	200,000
	학회/세미나 참가비	한국고분자학회 참가비(등록비) *3명	210,000
연구 과제추진비	회의비	회의비	50,000
	회의비	회의비	80,000
	사무용품비	딱풀	600
	시내여비	산업체 방문(석연지)	300,000
	국내여비	한국전기화학회 춘계총회 참석(석연지)	320,000
총계			**2,027,900**

[별첨 1. 연구팀 예산 사용 내역서]

(단위: 원)

예산세목	용도	구매품명	금액 (VAT포함)
연구 장비/재료비	케이블 연결	NETmate USB2.0 연장 케이블 [AM-AF] 블랙 2M	800
	케이블 연결	NETmate USB2.0 연장 케이블 [AM-AF] 화이트 2M	700
	케이블 연결	케이블메이트 USB 2.0 연장 리피터 케이블	11680
	차량 내부 기기	라즈베리파이 공식 7인치 터치스크린	78000
	차량 내부 기기	라즈베리파이4 스타터 키트	128000
	차량 외부 기기	라즈베리파이 카메라 모듈	99000
	차량 내부 기기	라즈베리파이4 아크릴케이스	7800
	차량 바퀴	오프로드바퀴 130파이	120000
	모터 드라이버	TB6600 스태핑 모터 드라이버	38400
	차량 모터	스태핑모터(24H53008N)	180000
	차량 외부	5T 투명 아크릴	18000
	차량 기기	57각 스테핑 모터용 브라켓	9000
	차량카메라	라즈베리파이 카메라 모듈 V2	27000
	차량 기기	기어드모터 고정 브라켓	24500
	차량 기기	라즈베리파이 터치스키린/카메라용 1mm피치 15p FFC 리본케이블	3600
	차량 기기	알루미늄 봉	65000
	차량 카메라	차량용 웹캠	72000
	차량 기기	아두이노 키트	288500
	전류전압측정	디지털멀티미터	58000
	차량 기기	브레드보드	2800
	차량 기기	데스트 소켓 점퍼 케이블 40p	6800
	차량 기기	TL-20001S 20A DMM TEST LEAD	5000
	차량 기기	HM-10 블루투스 4.0 V2 BLE 모듈	26400
	차량 기기	Arudino mega 2560	74000
	차량 기기	ESP8266 시리얼 와이파이 모듈	9600
	차량 기기	코엑스 네오박스	26600
	차량 기기	BB Power	9000
	차량 기기	아두이노 RFID 모듈	10200
	차량 기기	아두이노 TCRT5000	4800
	기타	상품 부가세	73,330
지원금액			**3,800,000**
사용총계			**1,478,510**