|  |
| --- |
| 졸업자격실험보고서 |
|  |
| 효과적인 군집 제어 알고리듬 개발 |
|  |
| Develop effective cluster control algorithms |
|  |
| 지도교수 강 동 현 |
|  |
| 동국대학교 과학기술대학 컴퓨터공학과 |
|  |
| 전 종 모 |
|  |
| 2 0 1 6 2 1 4 9 0 2 |

**목 차**

**1. 서론** 6 - 12

1.1 요약 6 - 12

**2. 이론적 배경** 13 - 33

2.1 요약 13 - 33

**3. 시스템 설계** 34 - 64

3.1 요약 34 - 36

3.2 기능 설계 37 - 64

**4. 프로그램 구현** 64 - 86

4.1 요약 64 - 73

4.2 첫 번째 차량 통신 알고리듬 74 – 108

4.3 첫 번째 차량 주행 알고리듬 109 – 117

4.4 첫 번째 차량 영상처리 알고리듬 118 – 119

**5. 실험** 00 - 00

**6. 결론** 00 - 00

**참고문헌** 120 – 121

**요약서** 00 - 00

**그 림 목 차**

[그림1] 교통사고 치사율 조사 6

[그림2] 군집 제어 9

[그림3] 개발 구성도 14

[그림4] ATmega128 핀 포트 14

[그림5] 차동 입·출력 방식 예 27

[그림6] 서보 모터 구성 33

[그림7] PWM의 개념도 35

[그림8] MATLAB으로 구현한PWM 출력 신호(25% 듀티 사이클) 36

[그림9] 차량의 구성도 37

[그림10] 프로젝트 전체 Block-Diagram 37

[그림11] 차선 인식 알고리듬 적용 후 테스트 40

[그림12] 차선 인식 알고리듬 개요 40

[그림13] PIN 설정 41

[그림14] HC SR-04 전방 거리 측정 코드 44

[그림15] 정지 후, 재 동작 코드 45

[그림16] 사용함수 45

[그림17] Canny 알고리듬 적용 46

[그림18] Canny 차선에 좌표 점을 통해 두 차선의 소실점을 구하는 코드 46

[그림19] 소실 점 좌표와 두 차선의 각도 값 계산 47

[그림20] 차선 인식 알고리듬 흐름도 47

[그림21] MFC를 이용한 threshold 함수 분석 - MFC 프로젝트 수정 49

[그림22] 이진화 타입의 콤보박스에 변수를 추가 50

[그림23] 컨트롤 변수 추가 50

[그림24] 컨트롤 변수 추가 콤보 박스 초기화 값 설정 51

[그림25] 실행 화면 51

[그림26] 임계값의 에디트 컨트롤 변수 추가 52

[그림27] 임계값의 슬라이더 컨트롤에 변수 추가 53

[그림28] 슬라이더 컨트롤 초기화 값 설정 53

[그림29] 슬라이더 값을 에디트 컨트롤에 표시하는 코드 추가 54

[그림30] 에디트 컨트롤 값을 슬라이더에 표시하는 코드 추가 54

[그림31] 실행 화면 55

[그림32] 변수 추가 55

[그림33] 함수 내 소스 코드를 수정 56

[그림34] 실행 결과(Type [ Binary ], Threshold Value [ 100 ], Max Value [ 255 ]) 56

[그림35] 필요한 라이브러리 Import 57

[그림36] 데이터 확인 57

[그림37] 가우시안 블러(검소한 가장자리 감지) 58

[그림38] Hough Transform 적용 58

[그림39] 구분 구적법 60

[그림40] 차선 검출 과정 60

[그림41] Gray 적용 63

[그림42] Canny Edge Detection 적용 63

[그림43] Hough Transform 적용 63

[그림44] 영상처리 결과 64

[그림45] 차량 구성도 66

[그림46] 프로젝트 전체 블록 다이어그램 68

[그림47] 주요 구성 요소 68

[그림48] 차량 간 통신 분리의 자유로움 71

[그림49] 통신 단절을 모를 경우 72

[그림50] 통신 단절 메시지 전달 및 초음파 센서를 통한 거리 유지 73

**표 목 차**

[표 1] 안전운전을 하지 않는 원인에 대한 응답 7

[표 2] 차량의 유형별 졸음운전 사고 원인 7

[표 3] NHTSA의 단계 구분 11

[표 4] 부품 명세서 23

[표 5] ATmega128 핀 기능………………………………………………………………………………………………24

**1. 서론**

1.1. 요약

시대가 변해도 해결되지 않는 문제가 있다. 바로 졸음운전이다. 2300만이 넘는 차량이 도로 위를 다니는 현대 사회에서는 졸음운전의 근본적인 문제를 해결하기 위한 솔루션이 큰 관심사로 대두되고 있다. 화물차 교통사고의 발생 원인을 조사해본 결과, ‘졸음운전’이 가장 높은 비율을 차지했다. 또한, 화물차 운전자들의 평균 근무 시간을 조사한 결과 일 근무 시간이 13.6시간으로 확인되었다. 운행 중 졸음 하나만으로도 사고가 발생할 확률이 높지만, 이러한 과로로 인한 졸음 현상이 반복될 경우 운전자들은 축적된 피로를 해소하지 못하여 만성화가 될 수 있고 이러한 문제가 심화되어 ‘습관성 졸음’으로 나타날 수 있기에 교통사고 발생 확률이 매우 높다.

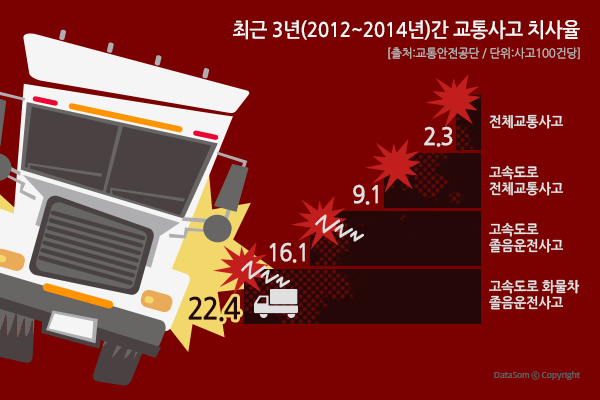


그림 1. 교통사고 치사율 조사[8]

화물차 운전자 같은 경우, 하루 평균 근로시간이 평균 13.6시간으로 확인되었다. 따라서 강도 높은 노동시간과 휴식 부족이 졸음운전과 판단능력 부족으로 이어지고 이것이 연평균 약 3만 건에 달하는 화물차 교통사고를 발생시키는 근본적인 원인으로 확인되었다. 산업이 발전할수록 유통의 중요성은 더욱 강조되기 때문에 기술적 발전을 통해 이를 개선할 필요가 있다.

|  |  |  |
| --- | --- | --- |
| 행동 원인 | 응답자(명) | 비율 |
| 화물의 도착시간을 맞추기 위해 | 34 | 25.0% |
| 지정체로 인한 운행지연 발생 대비 운행 | 26 | 19.1% |
| 조금이라도 일찍 도착하여 휴식시간 확보 | 24 | 17.6% |
| 평소 운전습관 | 20 | 14.7% |
| 도로 운행여건상 불가피 | 16 | 11.8% |
| 도로 위 주변차량이 별로 없어서 | 7 | 5.1% |
| 짧은 시간 내에 많은 화물을 이동하기 위해 | 6 | 4.4% |
| 남들도 안전운전하지 않기 때문 | 3 | 2.2% |
| **합계** | **136** | **100.0%** |

표 1. 안전운전을 하지 않는 원인에 대한 응답

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| (단위:%) | 피로누적 | 식곤증 | 수면부족 | 장거리운전 | 전날과음 |
| 전체 | **51.5** | 27.0 | 12.3 | 8.5 | .8 |
| 자가용 | 37.0 | 48.0 | 7.0 | 7.0 | 1.0 |
| 고속/시외버스 | 61.0 | 16.0 | 16.0 | 7.0 | 0.0 |
| 전세버스 | 53.0 | 24.0 | 9.0 | 12.0 | 2.0 |
| 화물차 | **55.0** | 20.0 | 17.0 | 8.0 | 0.0 |

표 2. 차량의 유형별 졸음운전 사고 원인

군집 제어 시스템이 적용되면, 여러 명의 운전자가 필요 없다. 따라서 화물차 운전자들의 강도 높은 근로시간 문제가 해결될 것이고, 휴식과 삶의 질을 높일 수 있을 것이다. 그리고 1인당 운송량 증가로 인해 기업 및 산업의 관점에서 보았을 때 경제적이라는 장점이 있고, 기계적 반응을 통해 사람의 실수로 인한 사고율도 많이 낮아질 것이다. 또한, 차량에 고가의 자율주행기술 옵션이 포함되어있지 않더라도 통신과 이를 보완해주는 장치들만 추가된다면 네트워크를 통해 일괄적으로 제어하는 형태의 새로운 자율주행 시스템이 가능해진다. 나아가, 차량 간의 네트워크 및 프로토콜 통신이 가능해짐으로 추가적인 기술의 발달에 따라 커넥티드 카 기술이 비약적으로 발전할 가능성이 열린다고 할 수 있다. 예를 들어, 도로 정보나 날씨 정보, 사고 발생 시 주변 차량으로의 알림 등이 가능해지고 이는 차량끼리의 하나의 네트워크가 형성됨을 의미 이외에도 스마트 폰 및 웨어러블 디바이스 등의 연동도 기대하여 볼 수 있다. 현대 사회인들은 대부분 스마트 폰을 가지고 다니기 때문에 스마트 폰과 차량과의 연결이 이루어진다면 생활에서의 편의뿐만이 아닌 도로에서의 안정성 또한 확연하게 증가하게 될 것이다.[1]

본인은 졸음운전 사고 비율이 가장 높은 화물 운송 운전자와 고속버스 운전자를 중점으로 하여 연구를 진행할 계획이다. 통상 단일 목적지에 사람 또는 화물을 운송하는 것이 주목적인 주행이기에 이러한 특성에 중점을 두어 ‘군집 제어 알고리듬’(단일 군집으로 동작하는 자동차가 일정 간격을 유지하여 네트워크를 통해 각 정보를 송수신하면서 자동차를 동작하게 하는 시스템)을 개발한다면 단일 목적지로 운행되는 다수의 자동차를 한 명의 운전자의 제어로 운행되고, 그로 인한 운전시간을 단축되어 졸음운전 사고를 사전에 방지할 수 있는 시스템을 구상했다.

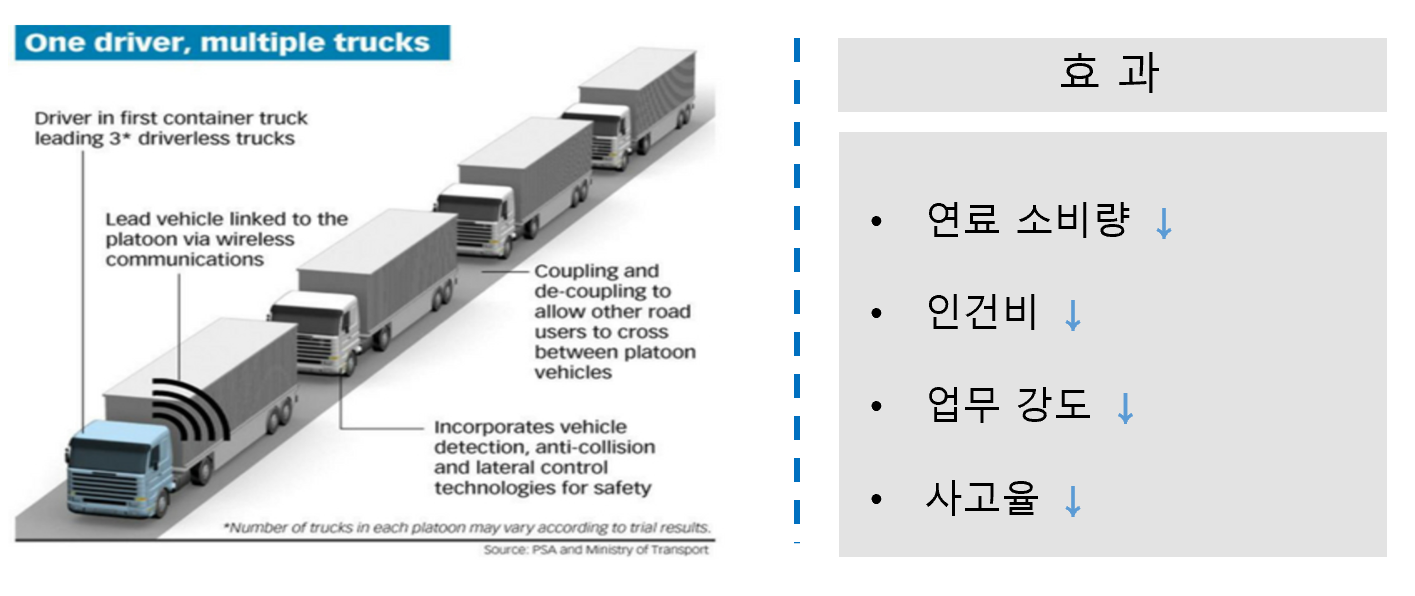


그림2. 군집 제어

1.1.2. 안전성

자동차들은 모두 하나의 네트워크로 제어된다. 후속 차량의 통제는 기계식으로 진행이 되기에

인간이 운전할 경우 대비 사고율이 줄어든다. 도로에서 인간이 반응 가능한 범위를 뛰어넘는

시스템이기에 인간이 반응하기 어려운 상황에서 합리적인 판단이 가능하다. 그 외 눈부심, 졸

음, 심장질환 등과 같이 정상 운행에 방해되는 조건이 사라지기 때문에 안전성에 있어서 상당

한 장점이 있다.[2]

1.1.3. 삶의 질 향상

화물 자동차 운전자나 운송업 관련 운전자들은 장거리 및 장시간 운전을 하는 열악한 환경에

노출되어있다. 이러한 환경 때문에 화물차 휴식 여건이 부족한 실정이다. 그로 인해 사고율은

상승은 물론 삶의 질도 떨어진다.

1.1.4. 경제성

이 군집 제어 시스템이 적용된다면, 여러 명의 운전자가 필요 없게 된다. 가장 앞의 차량에만

운전자가 탑승해서 제어하면, 뒤의 차량은 무인 제어가 가능하다. 따라서 인건비를 획기적으로

줄일 수 있다. 그리고 많은 양의 화물을 한 번에 운송이 가능하므로 기업과 산업 입장에서 경

제적인 소요를 줄일 수 있다.

1.1.5. 연비향상

사람이 조작하는 것이 아니라 기계가 자동차를 제어하기 때문에 급가속, 급제동이 줄어들게 되

고, 연비에 최적화된 운전이 가능해진다. 볼보사의 연구에 따르면 기계에 의한 제어를 통해 연

비를 약 7%가량 향상될 수 있다.

1.1.6. 목표

3대의 RC CAR를 군집제어 시스템을 통해 하나의 군집으로 묶어져서 구동되는 것이 최종 목표이다. 위의 목표를 달성하기 위해 군집 제어 시스템에 주로 사용되는 것은 통신을 통해 차량을 제어하는 것인데, 통신을 통해 차량을 제어하였을 경우 각각의 차량마다 바퀴의 방향이 다르다. 따라서 3대의 차량 모두 직진을 하는 것에 문제가 발생 따라서 최종 목표를 달성하기 위해 달성해야 하는 작은 목표들이 있다.

1.1.6.1. 직진성 보안

각 차량마다 직진하는 방향이 다르기 때문에 똑같은 하나의 명령을 통해 3대를 군집 주행시

키는 것이 문제가 발생 따라서 직진을 보완해야 하고 하프변환을 통해 차선 인식을 하여

직진성을 맞추는 것이 목표이다.[3]

1.6.1.2. 타이밍 보완

3대의 차량이 통신을 통해 명령을 받는 시간은 순식간이다. 따라서 좌회전 명령을 첫차에 내렸

을 때, 타이밍의 보완을 하지 않는다면 뒤에 따라오는 차량은 차선이 좌회전 차선이 아님에도

해당 자리에서 회전을 하게 된다. 따라서 이러한 타이밍을 개선하기 위해 초음파 센서와 적외

선 센서를 통해 거리를 측정하고 이를 통해 앞차의 좌회전 명령이 내려졌을 때의 앞차와의 거

리를 통해 좌회전 명령을 수행하는 시스템을 구현하는 것이 목표다.[5]

1.1.7 본 프로젝트 관련 선행연구 및 기술현황

상용 차량의 군집 제어 기술은 아직 완벽한 상용화가 되지 않았고 여전히 고도화 개발 중인 신기술이다. 자율주행기술, V2X 기술 등 모든 미래형 기술들이 집약된 기술이다. 완성차 기업, 국가, IT 기업 부품 기업 등 다방면으로 연구가 이루어지고 있고, 주로 완성차 기업과 IT 기업가 주도적으로 이끌어 가는 상황이다. 군집 제어와 관련이 깊은 자율주행 기술은 다음과 같다.

1.1.7.1. 자율주행기술

|  |  |  |  |
| --- | --- | --- | --- |
| 준수 | 페 달 | 스티어링 휠 | 운전자 정신 |
| LEVEL0 | 조작 | 조작 | 차량 조작 |
| LEVEL1 | 조작 | 자동 | 차량 조작 |
| LEVEL2 | 자동 | 자동 | 항시 대기 |
| LEVEL3 | 자동 | 자동 | 위급 상황 시 |
| LEVEL4 | 자동 | 자동 | 무인 가능 |

표 3. NHTSA의 단계 구분

상기의 표는 미국안전교통부(NHTSA)에서 정의한 자율주행능력 단계이다. ‘LEVEL4’ 같은 경우 무인으로 완전한 자율주행 기술이다. 2017 프랑크푸르트(IAA)에서 아우디사는 세계 최초로 레벨 3단계에 해당하는 자율주행차량인 A8을 선보였다. 이런 가운데 르노사는 레벨 4단계 수준의 자율주행 기술을 탑재한 콘셉트카 ‘심보오즈(Simbioz)'를 공개했다. 자동차 전문가들은 2021~2024년쯤 자율주행 시대가 열릴 것으로 전망하고 있다.

1.1.7.2. SARTRE

SARTRE는 유럽연합(EU)과 자동차 기업들이 공동으로 군집제어를 개발하는 프로젝트다. 이는 선두의 차량은 운전자가 직접 조종을 하고 무선통신을 이용해서 후발 차량을 선두 차량이 인도하는 시스템이다. 운전자가 원하는 경우 언제든지 군집제어 대열에 합류해서 시스템의 제어를 받을 수 있다. 이미 2012년에 시속 90km로 달리며, 차량 간 6m 간격을 유지하면서 세 대의 자동차 군집제어 시연에 성공했다. 군집제어 자동차의 대수를 조금씩 늘려갈 계획이라고 밝혔다.

1.1.7.3. 현대 자동차 군집제어

현대자동차는 2017년 5월 25일 ‘현대 트럭 & 버스 메가페어’를 개최했다. 이곳에서 상용차의 군집제어 기술의 현 주소를 알리는 동영상을 공개했고, 2018년에 제한된 조건 하에서 세 대의 상용차의 군집제어 시연을 예고했다. 또한, 2020년에 군집 주행 기술 확보를 목표로 한다고 밝혔다.

**2. 이론적 배경**

2.1. 요약

차량 키트, 싱글보드컴퓨터, 아틱을 이용하여 선두 차를 제어하여 후속 차량도 함께 제어할 예

정이다. ‘Putty’를 통해 싱글보드컴퓨터에 원격 접속하여 Linux(Ubuntu)를 통해 소스 코드를

컴파일하고 디버깅할 것이다. 아틱은 허브(Hub) 역할로 스마트폰으로 선두 차량을 제어하거나

앞차의 신호를 받아 후속 차량과 통신하는 역할이다. 앞차의 DC 모터와 Servo 모터를 제어하

여 속도와 방향을 결정 이때 선두 차량의 DC 모터와 Servo 모터의 PWM Value를 허브

인 아틱을 통해 후속 차량으로 전송 이 값을 입력받은 후속 차량은 일정 시간의 지연시간

을 가진 후 선두 차량과 같은 방향으로 움직이는 시나리오로 진행할 예정이다.[5]

2.1.1. DC모터 제어

PWM 제어를 통해 서보모터를 이용한 방향 제어와 DC모터의 속도를 제어.

2.1.2. 거리계산

초음파 센서를 차량 앞에 부착하여 통신 문제를 보완 통신이 불안정한 상황에서 앞 차량

및 장애물과의 충돌을 방지하고자 센서를 부착하여 일정 거리 안에 장애물이 감지되면 차량은

자동 정지 초음파 센서를 이용하여 앞차와의 거리와 방향을 유지 뒤따라오는 RC

CAR와 선두 RC CAR와의 거리를 계속해서 측정하여 속도를 제어 RC CAR 앞부분 양

끝에도 센서를 부착해 앞서가는 RC CAR와 방향이 틀어질 경우 이를 감지하여 앞선 RC CAR

와 방향을 맞춘다. 또한 측정한 거리를 계산하여 각 RC CAR 간의 속도를 계산[6]

2.1.3. 선두 차량 인식 기법

적외선 센서를 부착하여 RC CAR 간 직진성을 보완 RC CAR 전면에 부착된 2개의 적외

선 센서를 이용하여 앞차를 인식 방향이 바뀔 경우, 한쪽 적외선 센서가 인식을 못

이때 방향을 전환하여 앞차를 계속해서 뒤따라 갈 수 있게 [7]

2.1.4. 정확성 향상 기법

싱글보드컴퓨터용 카메라 모듈을 이용하여 주행의 안전성 및 직진성을 보완 적외선 센서

를 통한 방향 제어와 초음파 센서를 통한 속도 측정은 주변 환경 및 장애물에 따라 정확하지

않을 수 있다. 이를 보완하기 위해 싱글보드컴퓨터 카메라 모듈로 차선을 인식하여 주행할 때

의 정확성과 안정성을 높인다. [8]

2.1.5. 지연 최소화 기법

여러 센서들의 동작, 통신 및 입력 값을 받는 과정들이 순차적으로 이루어지면서 latency issue

발생 확률이 높을 것으로 판단된다. 따라서 multi-thread 방식을 이용하여 latency issue를 최

소화할 예정이다. 이를 보완하기 위해 multi-thread를 사용하여 각 센서, 통신 및 영상인식을

처리하는 과정에서 실시간성을 보장하여 latency issue를 최소화[9]

2.1.6. 개발 개요



그림 3. 개발 구성도

군집 주행 시스템은 하나의 자동차를 다른 자동차들이 따라가면서 원하는 목적지까지 안전하게 운행할 수 있는 하나의 자동차 군집을 만드는 것이다. 이렇게 하나의 명령 동작을 여러 대의 차량이 입력 받고 움직이기 위해서는 ‘커넥티드’의 개념, 즉, 통신을 통한 정보의 교환이 진행되어야 한다. 따라서 본인은 현재 소켓 통신[10]을 통하여 차량 간의 서버와 클라이언트의 입장에서 정보를 주고받도록 설정할 것이다. 멀티 스레드를 활용하여 각각의 차량이 별도로 서버와 클라이언트 기능을 동시에 수행하도록 하여 명령을 1:1로 전달받도록 하여 기존에 문제가 되었던 비효율적인 통신의 방법을 적용할 예정이다. 또한, 이러한 통신을 통한 군집 주행 시스템을 통해 군집에 포함되어 있지 않은 차량도 언제든지 군집에 참여할 수 있으며, 군집에 포함되어 있던 차량도 언제든지 자신이 직접 조정할 수 있도록 군집을 빠져 나올 수 있도록 할 것이다. 다음으로 자동차들이 모두 기계를 통해 만들어진다고 하더라도 각각의 기계마다 약간의 차이와 차선이 완벽한 직진이 아니라는 것을 고려하였을 경우 단순한 ‘출발’ 명령 하나로 여러 대의 차량을 조정하는 것에는 무리가 있다. 즉, 모든 차량이 하나의 차선에서 직진을 하는 것에는 군집 주행 시스템 외에 다른 별도의 장치가 필요하다. 따라서 본인은 각각의 차량에 자율주행의 개념이 필요하다고 판단하였고, 영상인식을 통해 차선을 인식하는 하프변환을 사용하여 차선을 검출하고 군집 주행을 수행하는 중간 중간에 보조적인 장치로서 직진성을 유지해주는 장치로 사용하려고 [8] 하지만 현재 적용 예정인 키트는 싱글보드컴퓨터로 OPEN CV를 통해 영상을 처리하는 것만으로도 많은 오류와 딜레이가 발생할 가능성이 높다.[9] 따라서 이러한 딜레이를 줄이기 위해 컴퓨터를 통해 영상 처리를 수행할 것이고, 이후 영상 처리를 통해 수행된 명령을 다시 싱글보드컴퓨터로 가져와 명령만 차량에서 수행하는 기능을 생각하고 있다.[10] 하지만 이렇게 명령을 통신을 통해 전달받은 경우에도 명령을 전달받는 딜레이가 발생할 것이고, 통신이 차단되었을 경우 직진을 하지 못한다는 단점이 있는 것은 분명하다. 하지만 실제 차량에서는 차량 내부에서 직접적으로 영상을 처리하고 직접 명령을 받고 수행할 것이기 때문에 실제 적용에 있어서는 문제가 되지 않을 것으로 판단된다. 현재 군집 주행 시스템은 멀티 스레드를 활용하여 통신을 하며, 통신을 통해 정보의 송수신 기능을 수행하는 기능을 수행할 수 있다. 또한, 보조적인 센서를 활용하여 초음파 센서를 통한 거리 측정으로 일정 거리 이하가 되었을 경우 모터의 동작을 제한하여 사고를 미연에 방지하는 기능을 추가 개발할 예정이다. 마지막으로 일정한 군집주행의 거리를 유지하기 위해 멀어지면 속도를 높이고, 가까워지면 속도를 줄이는 기능을 적용할 것이다.[11]

2.1.7. 관련 연구 조사

1. 영상처리를 이용한 모형자동차의 자율주행 On the self-driving of a model car using image processing: OpenCV 라이브러리를 사용해서 차선인식을 위한 영상처리를 수행. 차선인식 잡음 제거와 계산량 감소를 위한 알고리듬을 고안.

2. 차량 군집 주행에 적합한 멀티 캐스트 프로토콜: 차량 군집 식별자를 동적으로 제어하여 주변 교통 상황에 맞게 차량 군집을 관리하는 Platoon-Multicast 프로토콜을 제안.

3. Selforganizing systems reseaRCh group: 미국 하버드대 자가 조직 시스템 센터는 1000여개의 군집 로봇 무리를 구축하는 군집 로봇 행동제어를 연구.

4. 무인자율주행차량을 위한 경로생성방법 및 위치추정방법에 대한 연구: Design of Path Planning & GPS Estimation Algorithm for Unmanned Autonomous Ground Vehicle: 기존의 GPS를 통한 위치 추적 방식으로 인해 발생하는 오차 10~15m 가량을 극소화 하기 위해 DGPS를 사용 및 알고리듬 제시. 또한 경로 탐색 및 장애물 회피의 효율을 높이기 위한 알고리듬 제시.

5. 행위자기반모형을 활용한 자율주행차량의 교통혼잡 개선효과 분석: Autonomous Vehicle의 군집주행에서의 교통체계의 혼잡개선 효과 및 주행 안정성에 대한 연구.

6. University of Pennsylvania: 미국 펜실베니아대학에서는 생물의 군집 행동 양식을 분석하여 네트워크로 연결된 대규모의 Auto- nomous Vehicle에 적용시키며, 군집 로봇을 제어하는 알고리듬과 방법을 발전 및 원천 기술을 획득하는 것을 목표로 연구.

2.1.8. 개발 시나리오(계획 단계)

➀ 싱글보드컴퓨터에 HDMI포트와 USB포트를 이용해서 모니터, 마우스와 키보드를 연결해서 원격접속이 가능하도록 설정

➁ Putty를 이용하여 싱글보드컴퓨터에 원격 접속 싱글보드컴퓨터에 DC모터와 Servo모터를 연결해 PWM 제어를 하는 코드를 작성한다. 이를 바탕으로 차체가 전진, 후진, 좌회전, 우회전, 정지를 하는 소스 코드를 작성

➂ 앞차를 조종할 수 있는 어플을 MIT app invertor를 이용하여 만든다. 이 어플리케이션을 아틱 클라우드를 통해 아틱과 연동 또한 아틱을 자동차에 탑재된 싱글보드컴퓨터와 연동

➃ 완성된 어플을 이용하여 스마트 폰으로 라즈베리가 장착된 자동차에 명령을 내린다. 앞차는 기본적인 기능으로 전진, 후진, 좌회전, 우회전이 가능하고 속도를 조절할 수 있도록 설계

➄ 선두 차량은 구동 중에 DC모터와 Servo모터의 PWM 값을 아틱(Hub)를 통해 후속 차량의 싱글보드컴퓨터에 전송 뒤따라오는 후속 차량들은 보드에 앞 차와 같은 구동 코드를 입력하고 DC모터와 Servo모터의 PWM 값을 선두 차량으로부터 받아온다. 그러면 후속 차량들은 약간의 딜레이 후 선두차량의 모터제어를 따라가게 된다.(Tracking)

➅ 2번까지 완료되면 3번째 차에도 2번째 차와 같은 코드를 입력 3번째 차는 2번째 차의 정보를 아틱을 통해 입력받는다. 따라서 맨 앞의 차를 조종하는 것만으로도 뒤따라오는 차들을 동시에 제어할 수 있도록 한다.

2.1.9. 부품 명세서

|  |  |  |
| --- | --- | --- |
| 항 목 | 규 격 | 개 수 |
| IC | Atmega128 | 1 |
| Max 232 | 1 |
| 레귤레이터 | 12V to 5V 컨버터 | 1 |
| 센서 | HC-SR04 (초음파) | 6 |
| TCRT5000 (적외선) | 2 |
| 저항 | 10k | 1 |
| 4.7k | 10 |
| 3.3k | 10 |
| 2k | 10 |
| 100 | 10 |
| 모터 | DC 모터 | 2 |
| Servo 모터 | 1 |
| L298N (드라이버) | 2 |
| 싱글보드컴퓨터 | 라즈베리파이(변경될 수도 있음) | 2 |
| 배터리 | 9 V | 2 |
| 1.5 V | 4 |

표 4. 부품명세서

2.2.10. 부품 소개

**ATmega128**

ATmega128은 ATMEL사가 개발한 8비트 AVR마이크로 컨트롤러의 megaAVR 패밀리 계열중 하나의 프로세서입니다. 이 프로세서는 64핀으로 구성되었으며, TQFP형 패키지의 저 전력 8비트 CMOS 마이크로컨트롤러이다. 내부 구조는 RISC 아키텍처(RISC like ARChitecture)구조로 이루어져 있으며 대부분 1클록에서 수행되는 어셈블리어 명령어를 가지고 있어서 입력 클록 1MHz에서 1MIPS의 처리능력을 가진다. 한편, ATmega128은 최적화된 전력 소모 구조를 가지고 있어서 다른 마이크로컽트롤러에 비해 상대적으로 작은 전력을 전력을 소모하며, 처리 능력도 뛰어나다. 한편, ATmega128은 다음과 같은 특징들을 가지고 있다.

● 개선된 RISC 구조이다

   - 대부분 단일 클록에서 실행되는 133개이 명령어이다.

   - 8비트로 구성된 32개의 범용 작업 레지스터

   - 16MHz에서 16MIPS(Million Instruction Per Second)의 처리율

   - 2 사이클에서 실행되는 내장 곱셈장치

● 프로그램 메모리와 데이터 메모리

   - ISP가 지원되는 128K바이트의 플래시 메모리(10,000번의 쓰기/지우기 횟수)

   - 부트 코드 영역

   - 4K바이트의 EEPROM(100,000번의 쓰기/지우기 횟수)

   - 4K바이트의 내부 SRAM

   - 소프트웨어에 의한 LOCK 동작

● JTAG(IEEE standard 1149.1과 호환) 인터페이스

   - JTAG를 통한 Boundary Scan 방법

   - On chip Debug 지원

   - JTAG 인터페이스를 통한 Flash, EEPROM, 퓨즈, Lock 비트 설정

●내장 입출력 장치

   - 별도로 동작하는 프리스케일러와 출력 비교 장치로 구성된 2개의 8비트 타이머/카운터

   - 프리스케일러, 출력 비교 장치와 입력 캡처로 구성된 2개의 16비트 타이머/카운터

   - 외부 오실레이터에 의한 Real Time Counter

   - 2개의 8비트 PWM 채널

   - 2비트에서 16비트까지 조절이 가능한 6개의 PWM 채널

   - 8채널의 10비트 ADC

     ＊8개의 싱글 입력

     ＊7개의 차동 입력

     ＊1x, 10x, 200x 이득으로 이루어진 2개의 차동 입력

   - 바이트 중심의 TWO-Wried Interface(동기 2선식 통신)

   - 2개의 USART(Universal Synchronous &  As ynchron

     ous Receiver/Transmitter, 범용 동기/비동기 통신)

   - SPI(Serial Peripheral Interface, 동기 3선식 통신)

   - 별도의 오실레이터로 이루어진 Watchdog Timer

   - Analog Comparator

   - 53개의 범용 입출력 포트

●특수 동작 기능

   - Power-on RESET

   - Brown-out 검출기

   - 외부 및 내무 인터럽트 소스

   - 6개의 Sleep mode(Idle mode, ADC Noise Reduction mode, Power-save

     mode,Power-down mode, Standby mode, Extended Standby mode)

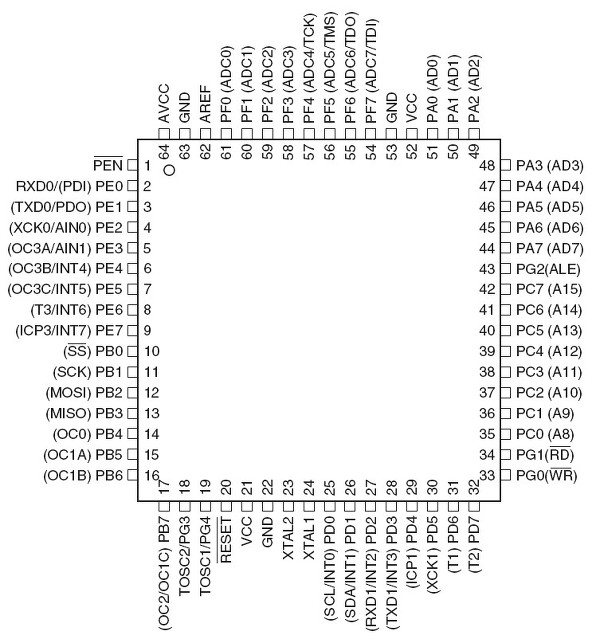


그림4. ATmega128 핀 포트

▶AVR ATmega128 핀 기능

|  |  |  |
| --- | --- | --- |
| 포트A(PA7~PA0) | 핀44~51 | 8비트 양방향성  입출력 포트, 비트별로 내부 풀업 저항을 연결할 수 있으며, 외부 메모리를 연결할 경우에는 어드레스(A7~A0)/데이터 버스(D7~D0)로 멀티플레스 된다. |
| 포트B(PB7~PB0) | 핀10~17 | 내부 풀업 저항이 있는 8비트 양방향성 입출력   포트로, 여러가지의 별도 기능을 가진 포트이다. |
| 포트C(PC7~PC0) | 핀35~42 | 내부 풀업 저항이 있는 8비트 양방향성 입출력   포트로, 외부 메모리를 연결할 경우에는 상위 어   드레스(A15~A8)버스로 동작한다. |
| 포트D(PD7~PD0) | 핀25~32 | 내부 풀업 저항이 있는 8비트 양방향성 입출력   포트로, 여러가지의 별도 기능을 가진 포트이다. |
| 포트E(PE7~PE0) | 핀2~9 | 내부 풀업 저항이 있는 8비트 양방향성 입출력 포트로,외부 인터럽트 등 여러가지의 별도 기능을 가진 포트이다. |
| 포트F(PF7~PF0) | 핀54~61 | 8비트 입출력 포트로 ADC에 대한 아날로그 입력 이다. |
| 포트G(PG7~PG0) | 핀18,19   33,34,43 | 내부 풀업 저항이 있는 5비트 양방향성 입출력 포트로, 여러가지의 별도 기능을 가진 포트이다. |
| /RESET | 핀20 | 리셋 입력 |
| XTAL1 | 핀24 | 반전 발진 증폭기 및 내부 클록 회로에 대한 입력 |
| XTAL2 | 핀23 | 반전 발진 증폭기로부터의 출력 |
| TOSC1 | 핀19 | 반전 타이머/카운터 발진 증폭기에 대한 입력 |
| TOSC2 | 핀18 | 반전 타이머/카운터 발진 증폭기에 대한 출력 |
| /WR | 핀33 | 외부 SRAM에 쓰기 신호 |
| /RD | 핀34 | 외부 SRAM 읽기 신호 |
| ALE | 핀43 | 외부 메모리가 enable될 때 사용되는 Address   Latch Enable |
| /PEN | 핀1 | 저전압 직렬 프로그래밍 모드에 대한 프로그래밍 enable 핀 |
| Vcc | 핀21,52 | 공급 전압 핀 |
| GND | 핀22,53 | 그라운드 핀 |
| AGND | 핀63 | 아날로그 그라운드 |
| AVcc | 핀64 | ADC 및 포트 F에 대한 공급 전압 |
| A RE F | 핀62 | ADC 레퍼런스 전압 |

표5. AVR ATmega128 핀 기능

**Max 232**

RS-232의 경우 주고 받는 선이 각각 1개다. 문제는 RS-232는 주변노이즈 영

향을 많이 받는다. 자동차나 주변 산업기기들에서 발생하는 전기적 노이즈가 유입되면

에러가 발생한다. 그래서 노이즈 영향을 줄이기 위해 전압을 ±12V를 사용한다. 신호

전압을 높이면 아무래도 주변 노이즈 영향을 줄일수 있기 때문이다. 그러다 보니 0~5V

로 사용하는 기기들과 통신을 할때는 전압을 맞추어 줘야 하는데 그때 사용하는 것이

MAX232같은 IC입니다 이들 IC는 전압을 변환 시켜주는 것이 주 목적이다.

위에서 언급한것처럼 RS232는 원리상 주변 노이즈 영향을 많이 받아서 권장 통신거리

는 10m정도입니다. 이보다 먼거리로 통신을 할수 있게 만든것이 488 또는 422,485와

같은 통신 방식이다.

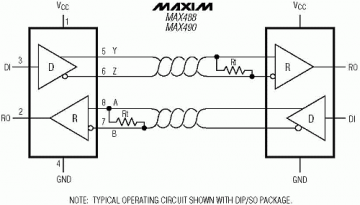


그림5. 차동 입·출력 방식 예

위와 같이 두 가닥선을 이용해 서로 반전된 신호로 주고받는 형태를 차동 입·출력 방

식이라 한다. 이 방식을 사용하면 주변에서 노이즈가 유입되어도 노이즈만 제거할 수 있다.

그리고 임피던스를 맞추어 고속통신도 가능하다. 요즘 트랜드는 거의 이 방식이다. (USB, LAN, SATA...) 따라서 MAX488의 경우는 전압을 변환해주기보다는 신호를

언발라스신호에서 발란스 신호로 바꾸어주고 또한 수신쪽에서는 차동입력을 이용해

노이즈는 제거하고 신호만 통과시켜주는 중요한 기능이 들어간 IC라고 볼 수 있다.

**HC-SR04**

초음파 거리 측정 센서 HC-SR04로 약 2-400cm 정도의 측정거리를 제공하며 3mm의

정확도를 가진다. 제품에는 송신기, 수신기, 제어회로가 포함되어 있다. 센서에는 4개의

핀이 있는데 VCC, Trig, Echo, GND가 있으며, 10us동안 Trig 핀을 high 상태로 만들

면 센서는 거리탐지를 시작한다. 거리탐지가 끝나고 센서 앞에 물체가 발견된다면,

Echo핀은 HIGH가 됩니다. 탐지된 거리에 따라 HIGH를 유지하는 시간이 다른데 이를

통하여 다음과 같이 거리를 계산할 수 있다.

Distance = ((Duration of high level)\*(Sonic :340m/s))/2

**결선**

VCC: 5V Supply

Trig: Trigger Pulse Input

Echo: Echo Pulse Output

GND: 0V Ground

**스펙**

power supply :5V DC

quiescent current : <2mA

effectual angle: <15°

ranging distance : 2cm – 500 cm (400cm)

resolution : 0.3 cm

**TCRT5000 (적외선)**

TCRT5000 적외선 반사 센서를 사용

탐지 거리 : 1mm ~ 8mm 적용, 초점 거리 2.5 mm

comparator 출력, 15mA 이상의 driving ability, 깨끗한 신호 파형

멀티 턴으로 정밀도 전위차계 감도 조정

운영 전압 3.3V - 5V

출력 형태 : 디지털 스위치 출력 (0 과 1)

쉬운 설치를 위한 고정 볼트 구멍

작은 보드 PCB 크기 : 3.2cm X 1.4cm

폭 넓은 전압의 LM393 comparator 사용

**라즈베리파이3B+**

Processor : BroaDCom BCM2387 chipset, 1.2GHz Quad-Core ARM Cortex-A53

802.11 b/g/n Wireless LAN

Bluetooth 4.1 (Bluetooth Classic and LE)

Memory : 1GB LPDDR2

GPU : Dual Core VideoCore IV® Multimedia Co-Processor

- OpenGL ES 2.0, hardware-accelerated OpenVG, 1080p30 H.264 high-profile decode

Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA type

40pin extended GPIO

3.5 mm audio jack

Composite video

10/100 Ethernet

CSI-2 : camera port for connecting the Raspberry Pi camera

DSI : display port for connecting the Raspberry Pi touch screen display

Micro SD Slot

Micro USB power souRCe

Deminsion : 85 x 56 x 17mm

Power : Micro USB socket 5V/2.1A

**MG-996R (서보 모터)**

Weight: 55g  
Dimension: 40.7×19.7×42.9mm  
Stall torque: 9.4kg/cm (4.8v); 11kg/cm (6.0v)  
Operating speed: 0.19sec/60degree (4.8v); 0.15sec/60degree (6.0v)  
Operating voltage: 4.8~ 6.6v  
Gear Type: Metal gear  
Temperature range: 0- 55deg  
Servo Plug: JR (Fits JR and so on)  
Dead band width: 1us  
servo wire length: 32cm  
servo arms &screws included servo arm  


그림6. 서보 모터 구성

**3. 시스템 설계**

3.1. 요약

본 개발의 목표는 군집주행 시스템 알고리듬 설계 및 증명하는 것으로 초음파센서와 적외선센서 그리고 카메라 모듈을 이용하여 군집주행을 구현한다.

3.1.1. PWM\* 제어를 통해 서보모터를 이용한 방향 제어와 DC모터의 속도를 제어한다.

3.1.2. 초음파 센서를 차량 앞에 부착하여 통신 문제를 보완한다. 통신이 불안정한 상황에서 앞 차량 및 장애물과의 충돌을 방지하고자 센서를 부착하여 일정 거리 안에 장애물이 감지되면 차량은 자동 정지한다.

3.1.3. 초음파 센서를 이용하여 앞차와의 거리와 방향을 유지한다. 즉, 뒤따라오는 실험용 자동차와 선두 실험용 자동차와의 거리를 계속해서 측정하여 속도를 제어한다. 실험용 자동차 앞부분 양 끝에도 센서를 부착해 앞서가는 실험용 자동차와 방향이 틀어질 경우 이를 감지하여 앞선 실험용 자동차와 방향을 맞춘다. 또한 측정한 거리를 계산하여 각 실험용 자동차 간의 속도를 계산한다.

3.1.4. 적외선 센서를 부착하여 실험용 자동차 간 직진성을 보완한다. 실험용 자동차 앞쪽 부착된 2개의 적외선 센서를 이용하여 앞차를 인식한다. 주행 중 방향이 바뀔 경우 한쪽 적외선 센서가 인식을 못하게 되어, 이때 방향 조정을 통해 앞차를 계속해서 뒤따라 갈 수 있게 한다.

3.1.5. 카메라 모듈을 이용하여 주행의 주행성 및 안정성을 보완한다. 적외선센서를 통한 방향 통제와 초음파센서를 통한 속도감지는 주행 환경에 따라 상이하다. 이를 보완하기 위해 카메라 모듈로 차선을 인식하여 정확성을 높인다.

3.1.6. 멀티 스레드를 사용하여 영상인식을 처리하는 과정에서 실시간성을 보장한다.

**\*PWM**[11]

PWM이란(Pulse Width Modulation, 펄스 폭 변조) 효율적인 스위칭 제어의 기본이 되는,

디지털 출력으로 아날로그 회로를 제어하는 기술이다. 초기에는 통신용으로 개발된 기술이었으나, 전류, 전압 제어용으로 효과적인 방식이었기 때문에 현재는 통신보다는 DC쪽 전력제어나 모터 제어 기술로 사용되고 있다. 간단히 설명하자면, 일정한 주기 내에서 Duty비를 변화시켜서 평균 전압을 제어하는 방법이다.

​

PWM의 개념도

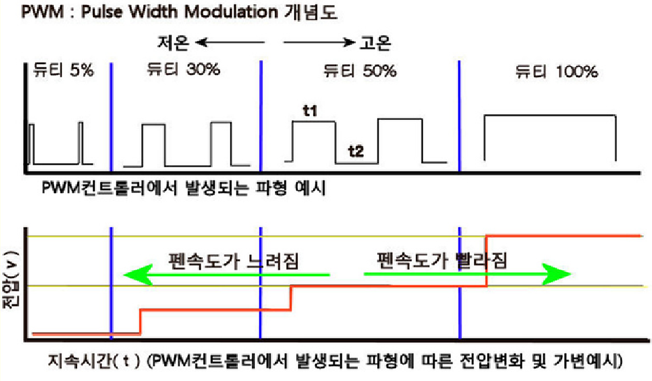


그림7. PWM의 개념도

콤퍼레이터(전압비교회로)의 +입력에 삼각파를 입력하고, - 입력에 제어 신호를 입력한다.

그러면 콤퍼레이터의 출력에는 입력 신호의 레벨 변화에 따라서 펄스 폭이 다른 출력이 나타나게 된다. [11]

​

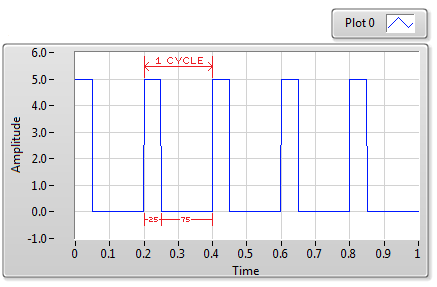


그림8. MATLAB으로 구현한PWM 출력 신호(25% 듀티 사이클)

PWM은 주기가 일정하고, 제어입력 신호에 따라서 펄스 폭의 듀티비가 변화하는 것을 의미한다. 출력 트랜지스터 같은 경우, ON하고 있는 동안에만 모터에 전류가 흐르게 된다.

듀티비가 클 수록 모터에 전류가 흐르는 전체 시간이 길어진다. 즉, DC 모터 등의 속도제어를 입력 신호의 변화 만으로 간단히 할 수 있다는 것이다. [11]

(모터가 빠른 주파수의 변화에 기계 반응을 하지 않는다는 것을 이용하는 것이다.)

DC모터나 LED 같은 경우 위와 같이 신호를 인가하면 실제로 속도나 밝기가 변하는 것을 확인할 수 있다. 따라서 ON/OFF를 계속 반복하게 될 경우, ON시간을 길게 하느냐, OFF시간을 길게 하는 것에 따라, 모터의 속도나 LED의 밝기의 변화를 확인할 수 있는 것이다. DC모터의 경우는 전압 변화에 기계적 반응이 느리기 때문에 LED의 경우는 인간의 눈으로 인지할 수 있는 속도보다 빠르게 On/Off 된다. 즉, 속도 및 밝기가 조절되는 것이다.

​

3.2. 기능 설계

3.2.1. 개요

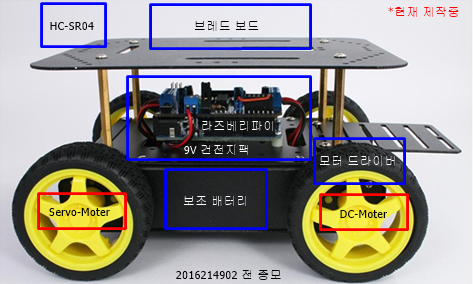


그림9. 차량의 구성도

그림10. 프로젝트 전체 Block-Diagram

현대인은 4차 산업 혁명의 시대에 살고 있다. 단순한 기계 조작을 넘어 기계와 기계간의 통신을 통한 대화, 사람과 기계간의 대화가 이루어지면서 영화에서나 보던 일들이 하나 둘 이루어지고 있는 세상에 살고 있다. 자동차 산업에서도 하나의 운송 수단을 만드는 것을 넘어 자동차가 주변을 인식하고, 사용자의 편의를 도와주는 스마트 자동차 체계가 늘어나고 있는 추세에서 본인은 ‘Connected’ 개념을 통해 군집 주행 시스템을 구성하기로 했다. 군집 주행 시스템은 하나의 자동차를 다른 자동차들이 따라가면서 원하는 목적지까지 안전하게 운행할 수 있는 하나의 자동차 군집을 만드는 것이다. 이렇게 하나의 명령 동작을 여러 대의 차량이 입력 받고 움직이기 위해서는 ‘Connected’의 개념, 즉, 통신을 통한 정보의 교환이 이루어져야 한다. 따라서 본인은 현재 소켓 통신을 통하여 차량 간의 서버와 클라이언트의 입장에서 정보를 주고받도록 설정했다. 기존에는 Internet of Things(이하 IoT)의 개념을 통해 하나의 허브에서 서버의 역할을 수행하고 다른 여러 대의 차량이 클라이언트로 서버에 입장하여 정보를 주고받도록 설정했다. 하지만 이러한 구성은 허브로부터 순차적으로 명령이 발생할 경우에도 하나의 차량이 다른 차들의 명령까지 모두 수신하는 현상이 나타났으며 모든 명령이 전달되는 시점까지 첫 번째 차량이 변화된 정보를 받지 못했다. 따라서 미비한 점을 보안하기 위하여 멀티 스레드를 활용하여 각각의 차량이 별도로 서버와 클라이언트 기능을 동시에 수행하도록 하여 명령을 1:1로 전달받도록 하여 기존에 문제가 되었던 비효율적인 통신의 방법을 해결할 것이다. 또한, 이러한 통신을 통한 군집 주행 시스템을 통해 군집에 포함되어 있지 않는 차량도 언제든지 군집에 참여할 수 있으며, 군집에 포함되어 있던 차량도 언제든지 자신이 직접 조정할 수 있도록 군집을 빠져나올 수 있는 시스템을 구축할 것이다. 다음으로 자동차들이 모두 기계를 통해 만들어진다고 하더라도 각각의 기계마다 약간의 차이와 차선이 완벽한 직진이 아니라는 것을 고려하였을 경우 단순한 ‘출발’ 명령 하나로 여러 대의 차량을 조정하는 것에는 무리가 있다. 즉, 모든 차량이 하나의 차선에서 직진을 하는 것에는 군집 주행 시스템 외에 다른 별도의 장치가 필요하다. 따라서 본인은 각각의 차량에 자율주행의 개념이 필요하다고 판단했고, 영상인식을 통해 차선을 인식하는 ‘Hough transform’[12]\*을 사용하여 차선을 검출하고 군집 주행을 수행하는 중간 중간에 보조적인 장치로서 직진성을 유지해주는 장치로 사용할 것이다. 그러나 현재 사용하고 있는 키트가 라즈베리 파이로 OPEN CV를 통해 영상을 처리하는 것만으로도 많은 오류와 딜레이가 발생한다. 따라서 이러한 딜레이를 줄이기 위해 컴퓨터를 통해 영상 처리를 수행할 것이고, 이후 영상 처리를 통해 수행된 명령을 다시 라즈베리 파이로 가져와 명령만 차량에서 수행하는 기능을 계획하고 있다. 그러나 이렇게 명령을 통신을 통해 전달받는 경우에도 명령을 전달받는 지연이 발생할 것이고, 통신이 차단되었을 경우 직진을 하지 못한다는 단점이 있는 것은 명백한 사실이다. 그러나 실제 차량에서는 차량 내부에서 직접적으로 영상을 처리하고 직접 명령을 받고 수행할 것이기 때문에 실제 적용에 있어서는 문제가 되지 않을 것으로 판단된다. 현재 군집 주행 시스템은 멀티 스레드를 활용하여 통신을 하며, 통신을 통해 정보를 송수신 하는 기능을 수행할 수 있다. 또한, 보조 초음파 센서를 통한 거리 측정으로 일정 거리 이하가 되었을 경우 모터의 동작을 제한하여 사고를 미연에 방지하는 기능을 추가적으로 구현할 것이다. 마지막으로 일정한 군집 주행의 거리를 유지하기 위해 멀어지면 속도를 높이고, 가까워지면 속도를 줄이는 기능을 구현중이다.

3.2.1.1. Hough transform: 영상에서 (x,y) 좌표공간의 픽셀들은 (r,θ) 매개변수 공간에서 곡선의 형태로 나타난다. 또한 (x,y) 좌표공간에서 같은 직선상에 존재하는 픽셀들의 경우 (r,θ) 매개변수 공간에서 교점을 가지게 된다. 허프 변환 기법은 이러한 특징을 이용하여 영상의 특징픽셀들을 (x,y) 좌표공간에서 (r,θ) 매개변수 공간으로 사상(mapping)시킨 후, 보팅 과정을 통해 교점을 찾아 직선 성분을 추출하는 방법이다.

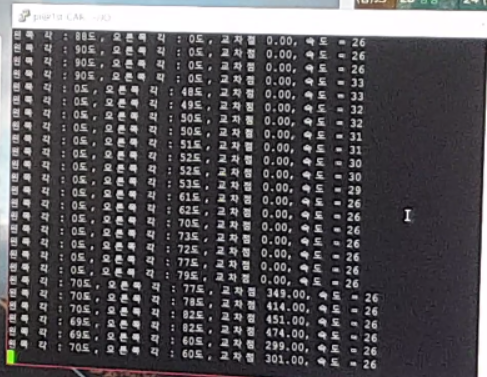


그림11. 차선 인식 알고리듬 적용 후 테스트

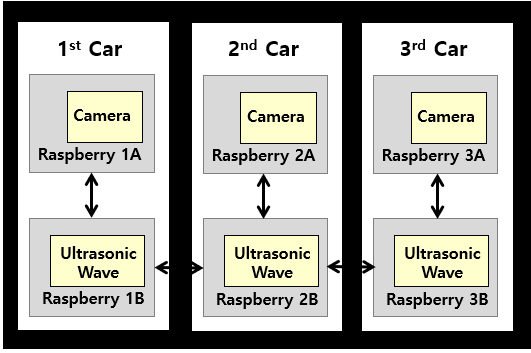
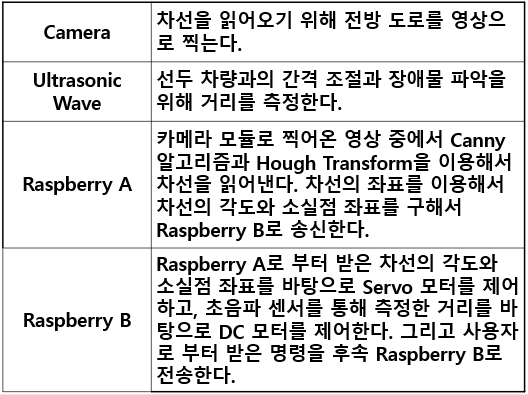


그림12. 차선 인식 알고리듬 개요

3.2.2. 아날로그 PIN 및 디지털 PIN 설정

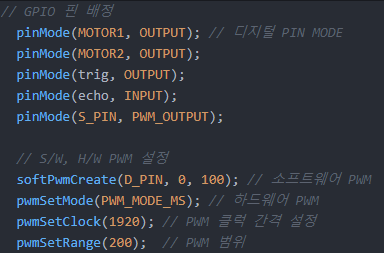


그림13. PIN 설정

라즈베리파이에는 Hardware PWM Pin으로 하나의 PIN만을 제공한다. 따라서 서보 모터를 두 개를 사용하거나, DC 모터도 PWM 제어를 하고 싶을 경우 추가적인 PIN이 필요하다. 이와같이 여러 PIN을 사용하는 방법은 다음과 같다. [11]

**1. 확장 보드를 구매해서 Hardware PWM PIN 추가 구성.**

**2. Software 적으로 구현된 PWM PIN.**

본인은 2. Software로 구현된 PWM PIN을 사용할 것이다. 따라서 Hardware PWM 대비 분해능이 부족하며, 정확성 측면에서 부족한 점이 많지만 DC모터는 속도 조절만 하면 되기에 큰 이슈가 없을 것이라고 판단했다. 따라서 ‘softPwmCreat()’ 함수를 사용해서 Software로 구현된 PWM의 주기를 설정한다. 이 주기의 몇 퍼센트를 사용할 것인지 설정하는 것이 본 프로젝트의 핵심인 PWM을 하는 방식이라고 할 수 있다. [11]

서보 모터의 경우 정확한 각도 제어가 가능해야만 RC CAR의 정확한 방향 조정이 가능하기 때문에 Hardware PWM Pin을 사용했다. ‘pinMode()’ 함수에서 PWM\_OUTPUT으로 설정함으로써 Analog 핀으로 설정한다. 그리고 ‘pwmSetMode()’ 함수를 통해서 PWM의 MODE를 설정해준다. 그리고 PWM을 사용할 때 중요한 주기와 Range를 설정해줘야 하는데, 본 프로젝트에 적용되는 서보 모터(MG-996R)는 20ms의 주기와 50hz의 주파수를 가지고 있다. 그리고 라즈베리파이가 제공하는 기본적인 PWM 주파수는 19.2Mhz이다. 따라서 이를 새로 분주 해야만 한다. 클럭 분주 공식은 다음과 같다. [11]

**분주된 주파수: 19.2Mhz / PwmCLOCK \* PWM Range**

따라서, pwmClock을 1920, Range를 200으로 설정하면, 50hz를 얻을 수 있다.

**19.2Mhz / 1920 \* 200**

또한, pwmRange가 200이고 이 분주된 파의 주기는 20ms 이므로 본인이 설정한 Hardware PWM은 단위 1당 0.1ms이 된다. 서보 모터 설정과 PWM 설정을 마쳤고, 나머지 DC 모터와 초음파 센서를 pinMode를 통해 설정해준다. 초음파 센서의 경우 하나는 파를 보내는 OUTPUT이고 하나는 수신하는 부분이므로 INPUT이 된다.

3.2.3. Server

통신을 사용하기 때문에 모든 기능을 구현하기에 앞서 TCP/IP 통신을 하기 위한 설정을 가장 먼저 수행하여야 한다. 설정이 오류 없이 수행한 수에는 통신에 참여할 Clinet가 접속하길 기다리며, 여러 Clinet가 동시 접속을 시도하더라도 Server가 모두 받을 수 있다. [12] 멀티 스레드를 사용하였기 때문에 통신에 필요한 Client는 한 대의 차량이다. 한 대의 차량과의 접속이 완료되면 서버의 기능을 하는 차량은 외부로부터의 명령을 대기하고 명령이 발생할 경우에 명령을 전달한다. 또한, server와 client간의 Round trip delay(이하 RTT) 속도를 맞추어야 한다. 보내는 속도와 받는 속도의 차이가 발생하기 때문이다. 따라서 한번의 명령이 들어왔을 경우에만 보내는 식의 구조를 사용했다. 개발초기의 경우에는 외부로부터의 명령이 들어올 경우에만 거리 측정이 가능하여 일부의 오작동이 발생하였지만 현재에는 멀티 스레드를 활용하여 다양한 경우에도 거리를 특정할 수 있다. 다음으로 외부로부터의 명령이 수신된다면, 그 즉시 server는 6개의 명령어 중 어떠한 명령인지 확인한 후 각각의 명령에 맞는 명령어를 recv\_data\_1에 입력한다. 그 후 라즈베리 파이와의 통신을 통해 명령을 주고 받는 것을 반복하며 첫 번째 자동차에서 마지막 자동차까지 명령을 순차적으로 전달하는 기능을 수행하도록 한다.

3.2.3. Client

서버의 경우와 마찬가지로 TCP/IP 통신을 하기 위한 설정을 가장 먼저 수행하여야 한다. 설정이 error없이 수행한 후에는 통신 대기중인 Server에 접근하여 접속을 시도한다. [12] 접속이 완료되면 Server로 부터의 명령을 대기한다.

3.2.4. HC-SR04

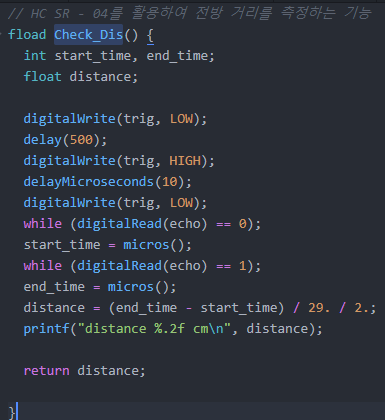


그림14. HC SR-04 전방 거리 측정 코드

본인이 사용한 HC SR-04 초음파 센서는 두 개의 Pin이 있다. 신호를 발생시키는 송신부인 ‘Trig’와 반사된 신호를 받는 수신부인 ‘Echo’ 등이 있다. 위와 같이 Trig 핀을 ‘digtalWrite’를 통해 ON/OFF를 반복하면서 신호를 발생시킨다. 그리고 ‘Echo’핀을 ‘digitalRead’를 통해 반사된 신호를 수신했는지 확인한다. 이때 ‘Echo’가 신호의 ‘수신한 시간 - 처음 시간’을 계산하여 신호가 이동한 거리를 구하고, 계산 결과 값을 ‘1/2’ 계산하여 전방 물체와의 거리를 식별한다. 따라서 그림12의 코드의 ‘Check\_Dis()’ 함수를 통해서 전방 물체와의 거리를 구할 수 있다는 것이다. 그리고 이 거리를 이 함수의 ‘return’으로 반환한다. [12]

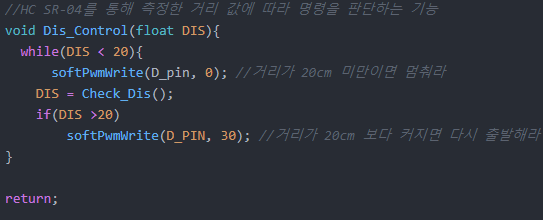


그림15. 정지 후, 재 동작 코드

상기 그림13의 함수는 그림12의 코드상에서 ‘Check\_Dis()’ 함수를 통해 측정한 전방 물체와의 거리를 입력 인수로 받아와서 거리에 따른 명령을 내리는 함수다. 본인의 기준으로 식별한 거리가 20cm 미만이면 충돌할 위험이 있다고 판단하여, ‘softPwmWrite()’ 함수를 통해 DC 모터의 속도를 0으로 변경하여 정지한다. 다시 거리 식별을 했을 때 거리가 20cm를 초과하면 DC 모터의 속도를 올려서 출발시키는 기능을 하는 함수다.



그림16. 사용 함수

개발 구상단계 당시 ‘main’ 함수에서 위와 같이 함수를 호출하여 거리를 측정하고 그 거리를 바탕으로 명령을 내리는 방식으로 진행했다. 따라서 데이터 송수신 중, 지연이 발생하여 거리 측정 주기가 굉장히 지연되는 상황을 식별했다. [12] 또한, 거리를 계속 측정해야 하는 특성상 데이터 송수신 중간에 상기 그림 14의 함수들을 호출하여 거리 측정 빈도를 높였으나 역시나 비효율적인 결과였다. 그러나 현재는 멀티 스레드를 활용하여 센서를 통한 거리 측정을 RC CAR 구동 코드와 독립적으로 수행하도록 구성했다. [15]

3.2.5. 영상 인식을 통한 차선 인식 구현

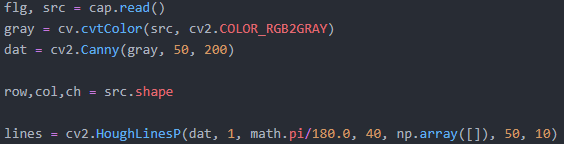


그림17. Canny 알고리듬 적용

첫 번째로 카메라 모듈을 활용하여 수신한 영상 데이터를 흑백 처리해보았다. 흑백 처리된 영상을 통해 차선 인식을 하면 더욱 처리 속도가 빨라진다. 이후 Canny 알고리듬\*을 사용해서 영상 속에서 명암 구분을 통해 Edge(이하 에지)를 파악했다. 이후 그 Edge들을 하프변환을 통해 가장 유력한 선을 뽑아낸다. [13]

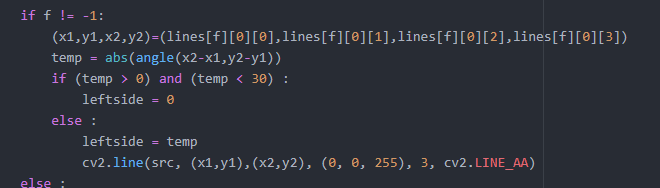


그림18. Canny 차선에 좌표 점을 통해 두 차선의 소실점을 구하는 코드

차선을 인식한 이후에는 차선에 좌표 점 두 개를 구한다. 이 좌표를 이용해서 차선의 각도와 두 차선의 소실점을 구할 수 있다. 위의 코드는 좌표점을 구하고 이를 이용해서 각도를 구하고 있다. 아래의 코드는 좌표점을 통해 두 차선의 소실점을 구하는 코드이다. [15]

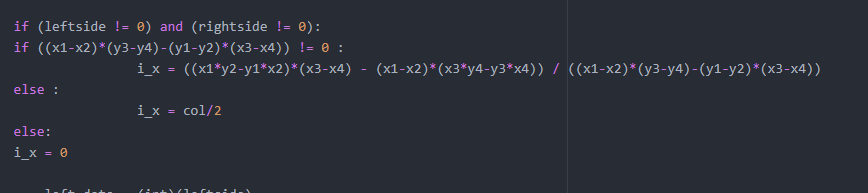


그림19. 소실 점 좌표와 두 차선의 각도 값 계산

이 소실점 좌표와 두 차선의 각도 값을 제어용 라즈베리파이에 TCP/IP 소켓통신을 이용해서 송신한다.

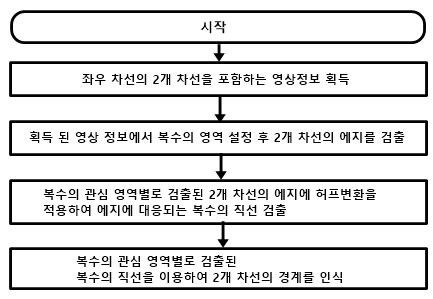


그림20. 차선 인식 알고리듬 흐름도[13]

3.2.5.1. Canny 알고리듬[15]

3.2.5.1.1. Canny 알고르듬이란?

Canny 에지 검출 알고리듬은 1986년도에 John F. Canny에 의해 고안되어 대중적으로 많이 사용되고 있다. Canny 알고리듬은 에지 검출 알고리듬에 있어서 다음 세가지의 기준을 내놓는다.

낮은 에러율: 실제 에지만을 탐지하는 능력

낮은 오차율: 실제 에지와 탐지된 에지의 픽실 거리 차이를 최소화

최소한의 응답성: 각 에지에 대해서는 한 번만의 검출[13]

3.2.5.1.2. Canny 함수

void Canny(InputArray image, OutputArray edges, double threshold1, double threshold2, int apertureSize=3, bool L2gradient=false )

image: 그레이 이미지

edges: 변환한 에지 이미지

threshold1: 에지 임계값

threshold2: threshold1 \* ratio

apetureSize: 커널 사이즈

3.2.5.1.3. MFC에 Canny Edge 슬라이더 컨트롤 변수 추가하기

다이얼로그에 이진화 타입을 보여줄 콤보박스, 임계값과 변환값을 표시해 줄 에디트 컨트롤과 슬라이더 컨트롤을 추가한다.

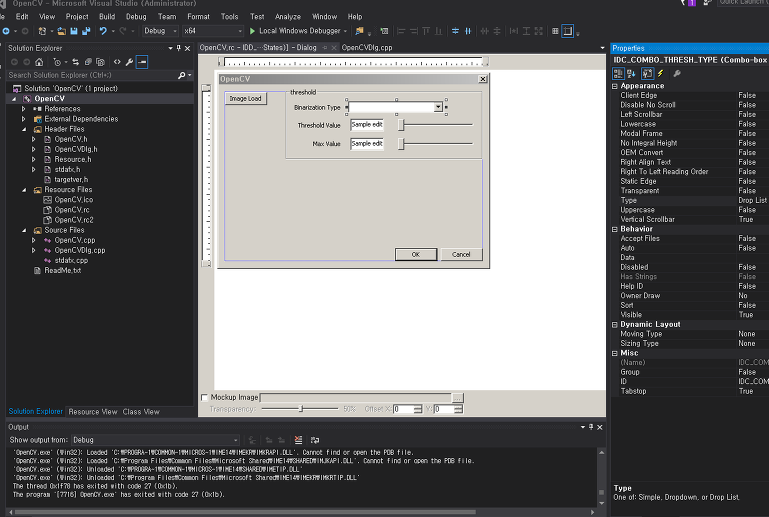


그림21. MFC를 이용한 threshold 함수 분석 - MFC 프로젝트 수정

콤보박스 속성 -> Type를 Drop List로, Sort를 False로 설정

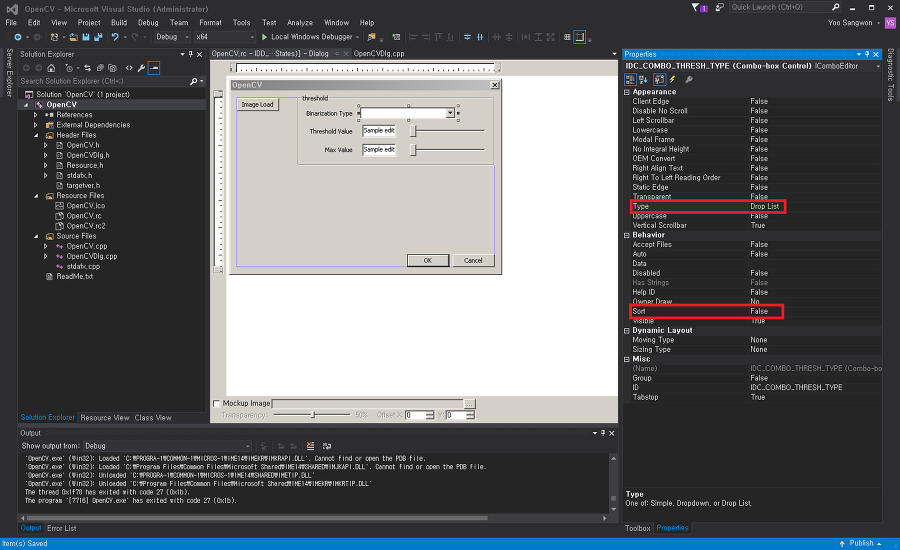


그림22. 이진화 타입의 콤보박스에 변수를 추가

콤보박스 마우스 우 클릭 -> Add Variable -> 컨트롤 변수(m\_combo\_thresh\_type) 추가

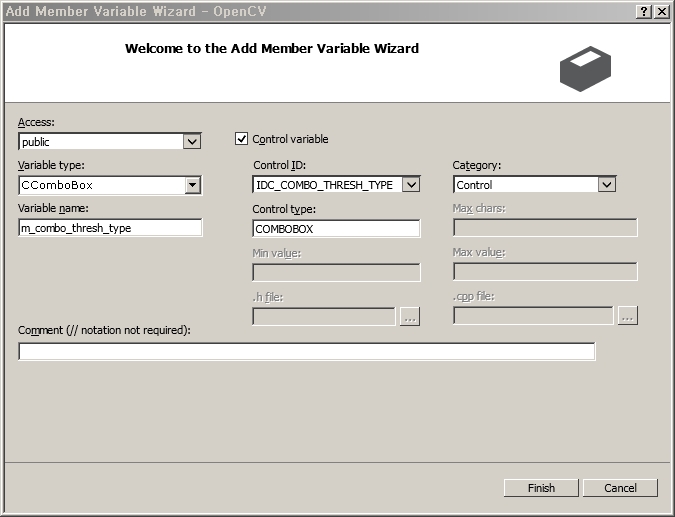


그림23. 컨트롤 변수 추가

BOOL COpenCVDlg::OnInitDialog() 함수 내에 콤보 박스 초기화 값 설정

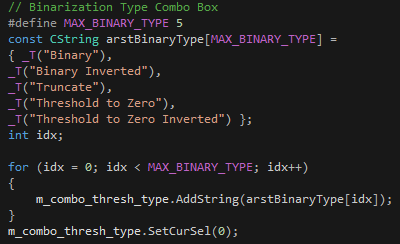


그림24. 컨트롤 변수 추가 콤보 박스 초기화 값 설정

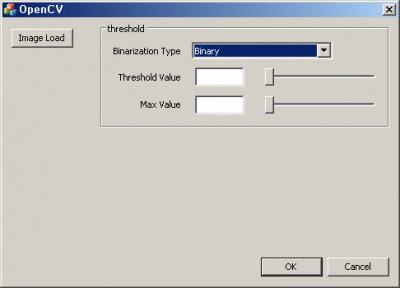


그림25. 실행 화면

에디트 컨트롤 마우스 우측 클릭 -> Add Variable -> m\_slider\_thresh\_value 추가

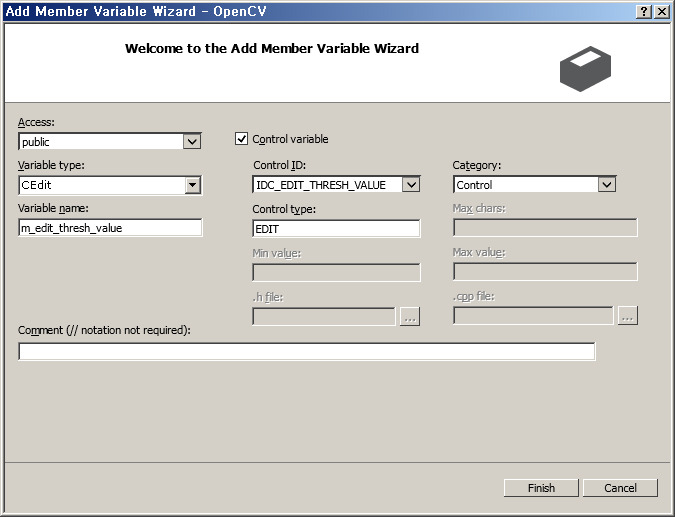


그림26. 임계값의 에디트 컨트롤 변수 추가

슬라이드 컨트롤 마우스 우측 클릭 -> Add Variable -> m\_slider\_thresh\_value 추가

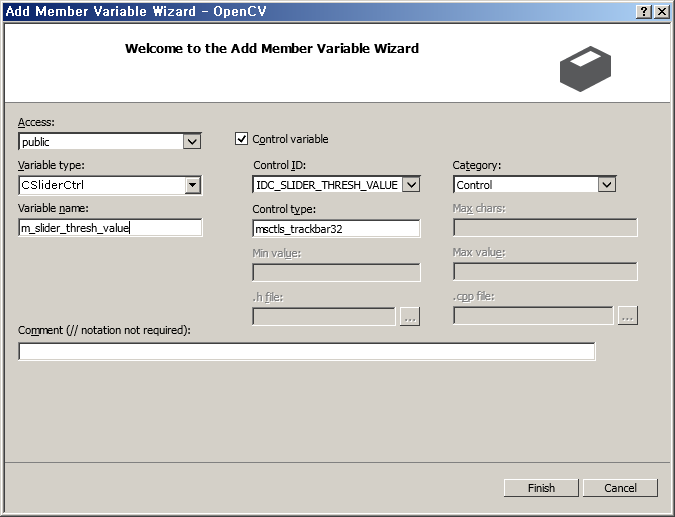


그림27. 임계값의 슬라이더 컨트롤에 변수 추가

BOOL COpenCVDlg::OnInitDialog() 함수 내에 슬라이더 컨트롤 초기화 값 설정

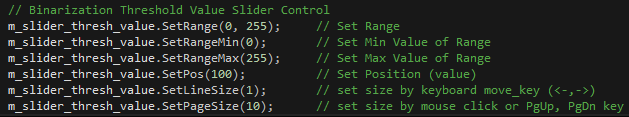


그림28. 슬라이더 컨트롤 초기화 값 설정

라이더 컨트롤 더블 클릭 -> 슬라이더 컨트롤 변경 시 함수에 슬라이더 값을 에디트 컨트롤에 표시하는 코드 추가

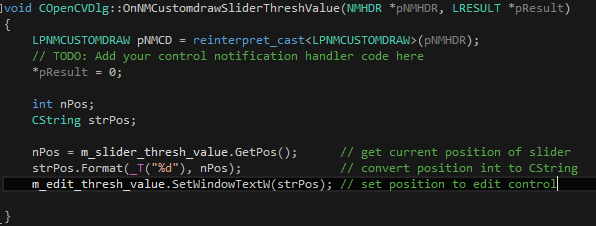


그림29. 슬라이더 값을 에디트 컨트롤에 표시하는 코드 추가

에디트 컨트롤 더블 클릭 -> 에디트 컨트롤 값 변경 시 함수에 에디트 컨트롤 값을 슬라이더에 표시하는 코드 추가

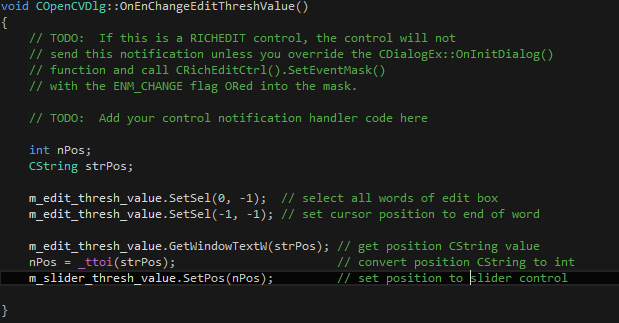


그림30. 에디트 컨트롤 값을 슬라이더에 표시하는 코드 추가

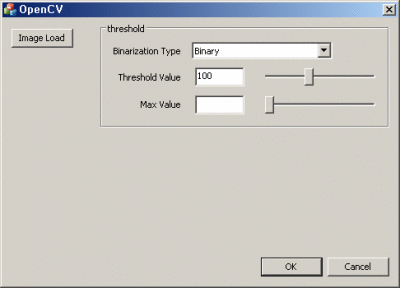


그림31. 실행 화면

변환값의 에디트 컨트롤과 슬라이더 컨트롤도 3)과 같은 방법으로 변수 추가

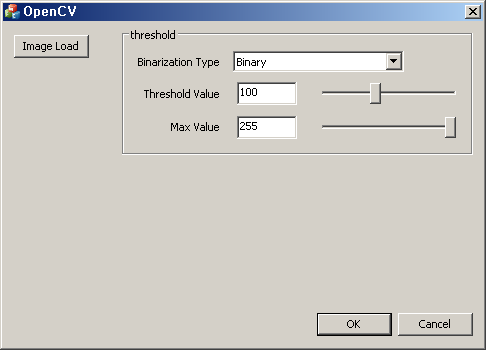


그림32. 변수 추가

Threshold함수 변수에 설정 값 불러오는 코드 추가

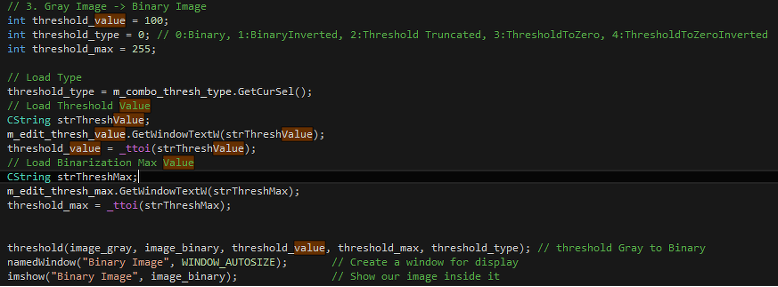


그림33. oid COpenCVDlg::OnBnClickedButtonImgLoad() 함수 내 소스 코드를 수정



그림34. 실행 결과(Type [ Binary ], Threshold Value [ 100 ], Max Value [ 255 ])

여기까지 만든 툴은 컨트롤 값을 변경하고 매번 이미지를 로드 해야하는 불편함이 있다. 다음은 Threshold 컨트롤 값을 변경할 때 실시간으로 변경된 이미지를 출력하도록 해야 한다. [13]

3.2.5.2. 차선 검출 상세

현재 H/W 사양으로는 직접 실험할 수 있는 환경이 안된다. 따라서 차선 이탈 경고 장치 인 차선 이탈 경고 (LDW)에서 사용할 수 있는 레인 감지 구현을 목표로 한다. ‘LaneDetection' 이란 차선을 찾아주는 알고리듬이다. [13]

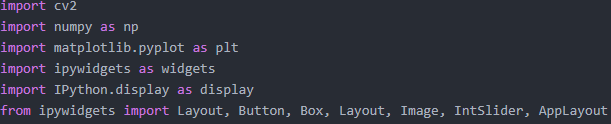


그림35. 필요한 라이브러리 Import

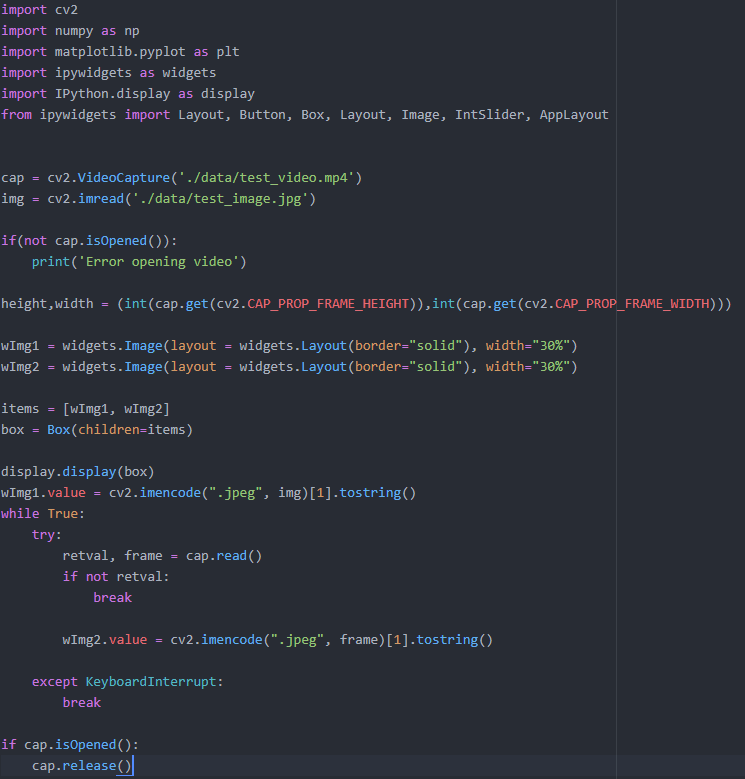


그림36. 데이터 확인

3.2.5.3. 에지 검출

1. 그레이 스케일로 변환

2. 노이즈 제거를위한 가우시안 블러 사용 (가우스 블러를 통해 가우시안 블러 선택)

3. Canny 에지 감지

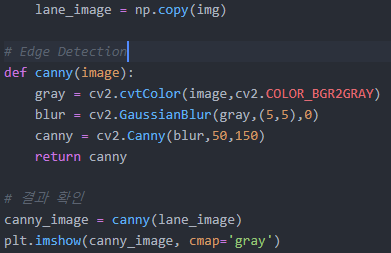


그림37. 가우시안 블러(검소한 가장자리 감지)

3.2.5.3. Hough Transform 적용



그림38. Hough Transform 적용[16]

3.2.5.3.1 Hough Transform 적용[16]

자율주행에 이용되는 기술 중 영상처리는 널리 산업에 서 이용되어 왔다. 자율주행은 운전자가 탑승하지 않기 때문에 시각적인 정보들을 카메라로 받아들인다. 이에 따라 영상처리 기술이 사용되며 처리과정에 오류가 생길 경우에 차량 탑승자의 생명에 치명적일 수도 있다. 최근 에는 제한적 자율주행 기술을 탑재한 차량들이 증가함에 따라 영상처리과정이 사고를 방지하기 위해 더욱 중요하 다. 물론 현 자율주행 단계는 운전자의 주행을 보조하는 방식이기 때문에, 차선 이탈 시, 경고를 울리는 시스템이나 핸들이 조금씩 움직이는 보조 시스템이다. 쉐보레 기업의 차선이탈 경고 및 차선 유지 보조시스템이 이에 해당한다. 다가오는 2020년대 초까지 자율주행의 단계를 4단계, 즉 운전자가 보조적인 부분을 맡는 단계까지 기 술을 완성시킨다는 입장을 발표했기 때문에 안전성이 중 요하다. 따라서 본 내용에서는 탑승자의 안전을 위해 영상처리 중 차선 검출 알고리듬을 이용하여 차선 인식률을 높이는 연구를 하고자 한다. 차선 검출은 보통 에지 검 출로 영상의 노이즈를 제거하고 라인 검출을 통해서 선 분을 검출한다. 본 논문에서는 에지 검출에는 앞에서 설명한 Canny Edge 알고리듬, 선분 검출에는 Hough Transform 알고리듬을[14] 사용한다. 차선을 잘 검출하기 위해 연구한 방향들을 살펴보면, RANSAC 알고리듬을 이용해서 검출한 연구, HSV 컬러모델을 이용한 검출에 대한 연구[11], 클러스터링을 통해 검 출한 연구 등이 있으나, 차선 검출 관련의 대부분의 논문 들이 실시간 검출에는 Hough Transform과 Canny Edge를 이용했다. 이번 항에서는 차선의 검출률을 높이는 방 안을 모색하던 중 구분 구적법처럼 검출하는 선분의 기준을 변화하는 것에 대한 내용을 기술한다. [16]

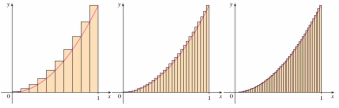


그림39. 구분 구적법[6]

그림33과 같이 구분 구적법은 곡면의 넓이를 구할 때, 직 사각형의 너비 구간을 충분히 좁게 만들어 넓이의 합을 구하여 그 근삿값을 얻는다. 유사하게 Hough Transform을 적용할 때, 검출하는 직선의 기준을 최소의 기준까지 줄인다면, 영상에 존재하는 모든 에지를 검출할 수 있다. 하지만, 이 방식의 문제점은 에지 검출율이 높아지는 대신, 차선이 아닌 에지까지 검출이 가능해지는 것이다. [14]

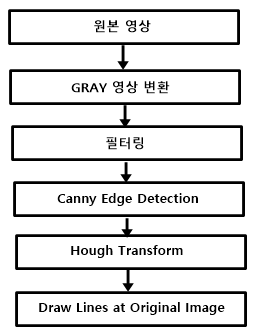


그림40. 차선 검출 과정[14]

영상처리 과정은 그림34과 같다. 원본 영상을 입력한 후, Gray영상으로 변환한다. Gray 영상변환 이외에도 영상의 이진화를 통해서도 적용 가능하다. 영상의 이진화는 완전 한 흑과 백의 형태로만 값이 존재하기 때문에 Threshold 값에 매우 민감할 수 있다. 필터링은 Gaussian Smoothing 같이 Blur효과를 통해 노이즈를 제거하는 것을 의미한다. Canny Edge 검출 기법은 Low Threshold와 High Threshold의 차이를 이용해서 Edge를 검출해내는데, 이러한 과정을 전 처리 과정이라고 한다. 전 처리 과정에서 Threshold를 높게 잡을 경우 노이즈는 잘 제거되나, 차선 도 마찬가지로 에지 검출율이 떨어진다. [14] 따라서 적절한 Threshold값 설정이 중요하다. Canny Edge를 적용한 영상을 Hough Transform 알고리듬을 적용한다. Hough Transform 알고리듬은 x-y평면의 한 점을 지나는 직선의 식을 다음과 같이 표현한다[16]



위 공식을 x-y평면의 관점에서 생각하면 x와 y가 변수이며 θ와 r이 상수이다. 하지만, θ와 r이 변수가 되는 θ-r평면에서는 θ와 r이 변수이다. 이 경우 x, y를 지나는 모든 직선을 표현할 수 있다. θ- r평면에서 다음과 같은 각각 한 점을 지나는 두 곡선이 존재한다고 한다.



이 두 곡선이 교차하여 교점을 가질 경우, 이 교점의 의미는 x-y평면에서 점 p1(x0, y0)을 지나는 한 직선과 점 p2(x1, y1)을 지나는 한 직선이 같다는 것이다. 따라서 직선을 찾은 후, 직선이 지나는 두 정점을 Edge에서 검 출하여 원본 이미지에 선분을 그리는 처리과정을 거친다. 본 내용에서는 Python을 이용해서 영상처리 과정을 구현한다. 추가적으로 라이브러리는 OpenCV 라이브러리를 이용한다. OpenCV는 영상처리에 대한 각종 알고리듬을 함수로 제공한다. 본 내용에서는 Canny Edge와 Hough Transform, Convert Gray Color, Blur, Line함수를 사용한다. OpenCV에서 제공하는 확률 허프변환 함수를 이용하여 최소길이의 범위를 점차 좁혀서, 소요시간을 측정한다. 소요시간은 실시간으로 처리하는 것을 가정해 야하기 때문에 초당 60 frame의 영상처리 소요시간으로 한다. 실험을 시행하는 하드웨어 사양은 다음과 같다.

**프로세서: AMD Ryzen 5 3600X 6-Core Processor, 3793Mhz**

**메모리(RAM):16.0GB**

**그래픽카드: Radeon RX 590 (8GB)**

우선 테스트 목적으로 입력은 jpg 혹은 png 등의 확장자 파일로 도로사진을 입력을 받아, 처리한다. 출력은 각 이미지 별로 새로운 윈도우를 띄워 출 력한다. 출력하는 이미지는 원본 이미지, Gray 이미지, Canny Edge 기법이 적용된 이미지, Hough Transform이 적용된 직선이 검출된 원본 이미지이다. 위와 같은 과정을 통해 구현한 결과 출력 영상은 다음과 같다.



그림41. Gray 적용

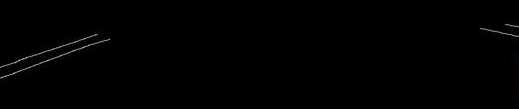


그림42. Canny Edge Detection 적용



그림43. Hough Transform 적용

Canny Edge 알고리듬은 Threshold값을 적절하게 설정하면 그림 36과 같이 깔끔하게 Edge가 검출된다. Edge 영상을 다시 Hough Transform 함수를 이용해 검출된 선분의 두 정점들을 행렬에 저장한다. 이 때 함수의 파라미터로 최소 직선길이를 지정한다. 직선으로 검출된 부분들을 파란 색 선분으로 표시했다. 결과 값으로는 입력을 받는 시점부터 알고리듬으로 변환이 끝나는 시점까지의 처리시간을 결과 값으로 출력한다. [14]



그림44. 영상처리 결과

관심영역 영역 내 경계 검출된 점들로 허프 변환을 통하여 차선을 검출한다. 이때 검출된 선분들의 기울기 값의 평균을 계산한다. 차량은 차선의 중앙값을 중심축을 ‘0’ 으로 하여 임계값을 적용하여 차량의 속도 및 방향 제어를 한다. [16]

**4. 프로그램 구현**

4.1. 요약

4.1.1. 기존 목표

3대의 RC CAR를 군집주행 시스템을 통해 하나의 군집으로 묶어져서 구동되는 것이 최종 목표이다. 위의 목표를 달성하기 위해 군집 주행 시스템에 주로 사용되는 것은 통신을 통해 차량을 제어하는 것인데, 통신을 통해 차량을 제어하였을 경우 각각의 차량마다 바퀴의 방향이 다르기 때문에 3대의 차량 모두 직진을 하는 것에 문제가 발생한다. 따라서 최종 목표를 달성하기 위한 프로그램 개발을 진행한다.

4.1.2. 직진성 보안

각 차량마다 직진하는 방향이 다르기 때문에 똑같은 하나의 명령을 통해 3대를 군집 주행시키는 것이 문제가 발생한다. 따라서 직진을 보안해야 하고 이를 위해 영상 인식 기능을 추가한다. 캐니 추출법과 하프변환을 통해 차선인식 및 각도를 계산하여 직진성을 높이는 작업을 진행한다.

4.1.3. 직진성 보안 달성도

직진성을 보안하기 위해 OPEN CV를 통한 영상처리 기술을 사용했다. 기존의 목표에서는 Canny 알고리듬을 통해 선의 edge를 파악하여 Hough-Transform으로 직선을 검출했다. 이후 검출된 선을 왼쪽, 오른쪽 선의 각도만으로 차선 인식 자율 주행을 실시했다. 하지만 카메라의 위치 변화, 각도간의 차이가 크지 않아 테스트를 진행할 때 마다 값들이 변하는 경우가 발생했다. 이를 보안하기 위해 두 직선이 다 보이는 경우에는 소실점을 활용하여 자율주행을 진행하였고, 각도와 소실점을 이용하여 차선인식 자율주행 프로그램 개발을 진행한다. [17]

4.1.4. 차량 시스템 개요

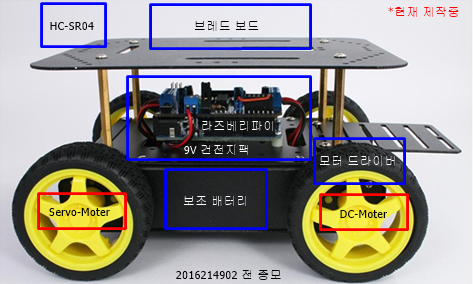


그림45. 차량 구성도

하나의 라즈베리 파이로 영상인식과 차량 제어 두 가지 모두를 수행하는 것은 성능 부족으로 인하여 영상 처리에 많은 Delay가 발생하였고 영상처리의 딜레이 때문에 정밀한 자율주행이 이루어지지 않았다. 따라서 총 2대의 라즈베리 파이를 활용하여 영상처리와 차량 제어를 각각의 보드에서 진행한 후 두 대의 라즈베리파이의 통신을 통해 차선 관련 정보를 받도록 구성했다. [17] 카메라 모듈을 통해 도로 상황을 영상으로 읽어온다. OPENCV 라이브러리를 통해 본인은 캐니 추출법과 하프변환 함수를 사용했다. 캐니 추출법을 통해 직선의 edge를 먼저 파악한다. 이후 파악한 직선의 edge를 통해 직선을 인식하는 하프 변환을 수행했다. 하프 변환을 통해 파악한 두 개의 차선의 좌표 값을 이용하여 두 차선의 소실점 좌표를 구한다. 그리고 각 차선의 좌표를 통해 차선의 각도를 구한다. 소실점 좌표와 각 차선의 각도 값을 모터 제어용 라즈베리파이로 TCP/IP 소켓통신을 이용해 전송한다. 통신을 통해 받아온 두 차선의 각도 값과 소실점 값을 이용해서 서보모터를 조향한다. 기존 각도만으로 서보모터를 조향하여 자율주행을 진행하였을 경우 직선과 좌, 우 회전의 각도 값이 크게 차이나지 않아 정밀한 주행이 불가능 했다. 따라서 이후 차선이 두 개 모두 보일 경우에는 소실점 좌표 값을 이용하고, 한 개의 차선만 인식될 경우에는 각도 값을 이용해서 서보 모터의 조향 값을 결정하여 자율주행을 좀 더 정밀하게 수행하도록 설계를 진행했다. 추가적으로, ‘군집 주행 시스템’의 기본적인 개념으로 군집을 형성한 자동차가 일정한 거리를 유지해야하기 때문에 초음파 센서를 활용해서 전방 거리 측정을 실시한다. 이를 통해 전방 장애물이나 선두 차량 간의 간격을 측정해서 DC모터의 PWM값을 조절한다. 기존 멀티 스레드를 활용하지 않았을 경우에는 통신을 하는 경우에는 DC모터의 PWM값을 조절하지 못하여 거리 측정에 문제가 발생하였지만 멀티 스레드를 활용하여 모든 경우에는 각각의 개별 코드가 개별적으로 동작하도록 했다. 이처럼, 카메라 모듈, 초음파 센서, TCP/IP 소켓 통신을 활용해서 군집주행 시스템을 설계했다.



그림46. 프로젝트 전체 블록 다이어그램

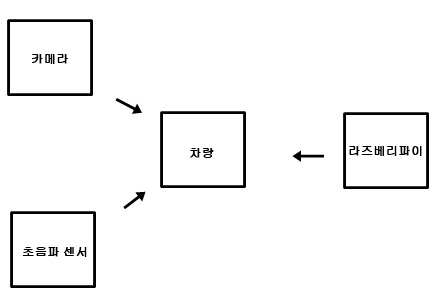


그림47. 주요 구성 요소

프로젝트를 구성하는 구성요소는 그림47과 같이 카메라, 초음파 센서, 라즈베리 파이를 포함하여 DC 모터, Servo 모터, Motor Driver 등 다양한 구성 요소가 존재한다. 하지만 프로젝트의 가장 중심이 되는 구성 요소가 카메라, 초음파 센서, 라즈베리 파이라고 생각하여 위의 세 가지를 제시했다. 기본적으로 RC CAR가 군집을 이루어 하나의 군집 내에서 통제를 받기 위해서는 라즈베리파이의 TCP/IP 통신이 이루어 져야 한다. 통신을 통해 연결이 된다면 각각의 라즈베리 파이는 하나의 외부 명령 혹은 첫차에서의 직접적인 명령을 나머지 차량에 전달할 수 있는 구조를 가지게 된다. 이렇게 통신을 통해 연결된 RC CAR 주행을 할 경우에 하나의 RC CAR가 직선 도록에서 직진을 유지하기 위해서는 미세한 조정이 필요하다. 즉, 단순히 통신으로 명령만 내린다고 여러 대의 자동차가 하나의 도로에서 똑같은 방향으로 직진을 하는 것이 아니다. 따라서 본인은 카메라와 OPEN CV를 사용하여 차선 인식을 하게하고, 그것을 통해 여러 대의 자동차가 단순히 직진이라는 명령만 받아도 하나의 차선 내에서 직진을 할 수 있도록 설계했다.

이렇게 직진성을 보완하기 위해 카메라와 영상처리 기술을 사용하니 라즈베리 파이라는 보드에 기능에 문제가 발생하였고, 이를 보안하기 위해 프로젝트를 구성하는 라즈베리파이의 개수를 증가시켰다. 따라서 카메라와 연결되어 영상처리는 하는 라즈베리 파이와 자동차의 DC모터 SERVO모터와 연결되어 자동차를 제어하는 라즈베리 파이 이 두 가지의 라즈베리 파이가 차량을 구성한다. 영상인식을 통한 차선인식을 하는 라즈베리파이는 파이A라고 하고, 차량을 제어하는 라즈베리 파이를 파이B라고 하였을 때, 파이A가 차선을 인식하고 차선에 대한 소실점과 차선이 가지는 각도를 계산하는 역할을 진행한다. 각 시점의 소실점과 각도를 계산을 한 후에 파이B로 계산한 정보를 차량 간의 통신처럼 TCP/IP통신을 통해 정보를 전달한다. 이렇게 전달된 정보를 활용하여 각각의 소실점의 위치와 직선의 각도로 파이B는 차량의 직진을 유지하도록 SERVO 모터를 제어한다. 따라서 두 개의 라즈베리파이는 통신을 통해 정보를 주고받기 때문에 추가적인 선으로 연결이 필요하지 않게 된다. [17]

4.1.5. 시나리오

4.1.5.1. 군집의 형성과 군집 이탈[17]

● 화물차를 운전하는 운전자가 자신이 화물을 운송해야 하는 목적지의 차량들을 각

각의 IP주소를 통해 차량별로 통신에 들어오도록 설정한다.

● 목적지가 같은 곳이 아니더라도 일부 동선이 겹치는 경우에는 같은 동선에 포함

되어 있는 차량과도 군집을 형성한다. [17]

● 군집이 형성되어 있는 도중에도 추가적인 차량이 군집에 합류할 수 있다.

군집의 기본 단위는 사용하는 네트워크망에 따라 그 개수가 제한이 될 수 있다.

● 군집 주행 시스템의 가장 기본 단계이자 첫 번째 단계로는 군집을 형성하는 것이다. 같은 군집으로 형성되었을 경우 하나의 명령을 통해 모든 자동차들이 같이 제어되기 때문에 움직이는 동선이 같은 차량이 군집에 포함될 것이다. 따라서 기본적으로는 출발지와 목적지가 동일한 경우에 한에 군집을 형성할 것이다. 하지만 일부의 경우에는 목적지는 다르지만 출발지에서 중간 경로까지 같은 동선을 가지는 차량이 있을 수도 있다. 또한, 출발지는 다르지만 중간 지점에서 목적지까지의 동선이 같은 경우도 있을 것이다. 모든 경우를 고려하면, 현재 설계하는 군집 주행 시스템은 군집을 형성하고 이탈하는 구조가 유연해야 된다고 생각하였고 이에 어떠한 경우에도 추가적으로 군집에 잘 들어오고 즉시 이탈할 수 있는 구조를 가지도록 한다.



그림48. 차량 간 통신 분리의 자유로움

4.1.5.2. 차량 간 통신이 끊어진 경우[17]

● 첫 번째 차량에서 군집을 제어하는 사람에게 통신의 끊김을 알리는 메시지 출력

● 첫 번째 차량의 stop명령을 받지 못하므로 첫 번째 차량이 stop을 하였을 때, 차량 간의 거리 유지를 돕는 초음파 센서를 통해 stop명령이 내려지도록 설계[17]

● 차량 간의 통신이 끊어진 경우에는 첫 차의 명령을 통해 따라오면 나머지 군집의 자동차들이 통신만으로 명령을 받았기 때문에 이후의 명령을 받을 수 없게 된다. 따라서 위와 같은 경우가 발생하였을 때, 각각의 차량은 제어를 받지 못하고 사고에 이르게 될 것이다.

상기 항목과 같은 경우를 없애기 위해 각각의 차량들도 어느 정도의 스마트 차량의 기능을 가지고 있어야 된다. 하지만 사고의 예방도 많이 중요하지만 하나의 차량별로 모두 스마트 차량의 기능을 부여하여 군집 주행을 시행하게 된다면 각각의 차량이 모두 가격이 많이 늘어나게 될 것이며 군집 주행 시스템을 만들어도 사용하지 않는 경우가 많이 발생할 것이다.

따라서 금전적인 면과 돌발 상황에서 사고의 발생 위험을 줄이기 위한 방안을 고려했다. 만약 돌발 상황이 발생하여 차량 간의 통신이 끊어진 경우에 제일 먼저 첫 번째 차량에서 군집을 제어하는 사람이 차량 간의 통신이 끊어졌을 알도록 해야 한다. 이를 해결하기 위해 각각의 차량 간의 통신이 끊어졌을 경우 Server의 역할을 하는 차량이 통신이 종료되었음을 나타내는 메시지가 화면에 뜨도록 했다. 이후 차량을 제어하던 운전자는 다시 통신을 연결시키기 위해 차량을 멈출 것이고 이것을 뒤의 차량들은 거리를 유지하기 위해 사용하였던 초음파 sensor를 통해 일정 거리를 유지하기 위해 stop이라는 명령이 직접적으로 가해지게 될 것이다. 이렇게 한다면 추가적인 스마트 차량을 만들 기술들을 추가하지 않아도 돼서 비용적인 측면이 부담되지 않을 것이고, 돌발 상황에서도 사고 발생률이 현저하게 줄어들 것이다. [17]



그림49. 통신 단절을 모를 경우



그림50. 통신 단절 메시지 전달 및 초음파 센서를 통한 거리 유지

4.2. 첫 번째 차량 통신 알고리듬 설계(코드 작성 및 상세화 진행중) [18]

현재는 본 시스템은 2.4Ghz 대역에서 통신을 한다, 5Ghz 대역에서 통신하게 된다면, 전체적인 Delay를 줄일 수 있어서 더욱 좋은 퍼포먼스를 낼 수 있었을 것이라 판단된다.

4.2 에서는 통신 알고리듬, 연결 등, 통신에 관련된 내용을 기술한다.

다음 코드는 첫 차에 내린 명령이 후속 차들에게 통신이 되는 코드다.

4.2.1. 기본 설정[18]

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <netdb.h>

#include <pthread.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#include <assert.h>

#include <softPwm.h>

#include <wiringPi.h>

#include <unistd.h>

#include <math.h>

#define MYPORT 3490 // 2번, 3번차와의 통신 포트

#define CAPORT 5100 // 카메라와의 통신 포트

#define MAXDATASIZE 10 // 카메라와의 통신 메세지 최대 용량

#define SENDSIZE 3 // 차량 간 통신 메세지 최대 용량

#define BACKLOG 4 // 통신 데이터 Queue

#define HIGH 1

#define LOW 0

#define S\_PIN 1 // SERVO PWM

#define D\_PIN 0 // DC PWM

#define MOTOR1 2 // DC direction

#define MOTOR2 3 //

#define trig 5 // Sensor

#define echo 4

#define MIN\_SPEED 27

#define MAX\_SPEED 38

#define COLUMN 640

void Send(void \*ptr);

void Check\_Dis(void \*ptr);

void Video(void \*ptr);

void Detect(void \*ptr);

float distance, i\_x;

int temp;

char change\_lane;

int mag\_l, mag\_r;

int speed = MIN\_SPEED;

int ok;

float direction = 14.9;

int camera\_sock, sockfd, new\_fd;

int main(int argc, char\* argv[]) {

struct hostent \*he;

struct sockaddr\_in camera\_addr, my\_addr, their\_addr;

int sin\_size, status;

int on = 1;

pthread\_t thrd1, thrd2, thrd3, thrd4;

// wiringPi 설치

if (wiringPiSetup() == -1) return 1;

// GPIO 핀 배정

pinMode(MOTOR1, OUTPUT);

pinMode(MOTOR2, OUTPUT);

pinMode(trig, OUTPUT);

pinMode(echo, INPUT);

pinMode(S\_PIN, PWM\_OUTPUT);

// S/W, H/W PWM 설정

softPwmCreate(D\_PIN, 0, 100);

pwmSetMode(PWM\_MODE\_MS);

pwmSetClock(1920);

pwmSetRange(200);

// DC Motor Initialization

digitalWrite(MOTOR1, LOW);

digitalWrite(MOTOR2, LOW);

softPwmWrite(D\_PIN, 0);

if (argc != 2) {

fprintf(stderr, "usage: ./client<host-name>\n");

exit(1);

}

4.2.2. 카메라 RPI와의 연결 (Client)

if ((he = gethostbyname(argv[1])) == NULL) {

herror("gethostbyname");

exit(1);

}

if ((camera\_sock = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("socket");

exit(1);

}

camera\_addr.sin\_family = AF\_INET;

camera\_addr.sin\_port = htons(CAPORT);

camera\_addr.sin\_addr = \*((struct in\_addr \*)he->h\_addr);

bzero(&(camera\_addr.sin\_zero), 8);

if (connect(camera\_sock, (struct sockaddr \*)&camera\_addr, sizeof(struct sockaddr)) == -1) {

perror("connect");

exit(1);

}

printf("------------Connecting at %s-----------\n", argv[1]);

4.2.3. Thread 생성 및 실행[18]

pthread\_create(&thrd1, NULL, (void \*)Check\_Dis, NULL);

pthread\_create(&thrd2, NULL, (void \*)Video, NULL);

pthread\_create(&thrd3, NULL, (void \*)Detect, NULL);

4.2.4. 2번, 3번차 RPI와의 연결 (Server) [18]

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("socket");

exit(1);

}

my\_addr.sin\_family = AF\_INET;

my\_addr.sin\_port = htons(MYPORT);

my\_addr.sin\_addr.s\_addr = INADDR\_ANY;

bzero(&(my\_addr.sin\_zero), 8);

setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &on, sizeof(on));

if (bind(sockfd, (struct sockaddr \*)&my\_addr, sizeof(struct sockaddr)) == -1) {

perror("bind");

exit(1);

}

if (listen(sockfd, BACKLOG) == -1) {

perror("listen");

exit(1);

}

printf("Server : waiting Second Car\n");

while (1) {

sin\_size = sizeof(struct sockaddr\_in);

if ((new\_fd = accept(sockfd, (struct sockaddr \*)&their\_addr, &sin\_size)) == -1) {

perror("accept");

continue;

}

if ((status = pthread\_create(&thrd4, NULL, (void \*)Send, (void \*)&new\_fd)) != 0) {

printf("4th thread create error: %s\n", strerror(status));

exit(0);

}

}

4.2.5. Thread 종료 대기[18]

pthread\_join(thrd1, NULL);

pthread\_join(thrd2, NULL);

pthread\_join(thrd3, NULL);

close(new\_fd);

close(sockfd);

return 0;

}

void Detect(void \*ptr) {

while (1) {

if ((change\_lane == 'r') || (change\_lane == 'l'))

{

ok++;

softPwmWrite(D\_PIN, speed + 10);

if (mag\_r == 0)

{

if (mag\_l < 48)

{

change\_lane = 'i';

softPwmWrite(D\_PIN, speed);

}

}

}

else

{

4.2.6. 두 차선이 모두 보이는 경우의 판단[18]

if (i\_x != 0)

4.2.7. 두 차선의 교점이 오른쪽에 맺힌 경우[18]

if (i\_x > 390)

4.2.8. 두 차선의 교점이 오른쪽에 맺힌 경우(우조향) [18]

{

direction = direction + 0.15;

pwmWrite(S\_PIN, direction);

softPwmWrite(D\_PIN, speed + 1);

}

4.2.9. 두 차선의 교점이 왼쪽에 맺힌 경우[21]

else

{

4.2.10 두 차선의 교점이 왼쪽에 맺힌 경우 (좌조향)

else

{

direction = direction - 0.2;

pwmWrite(S\_PIN, direction);

softPwmWrite(D\_PIN, speed);

}

4.2.11 한 개의 차선, 또는 우측 차선만 보이는 경우[19]

if (mag\_l == 0)

{

if (mag\_r < 67)

{

// 좌회전

pwmWrite(S\_PIN, 10.8);

softPwmWrite(D\_PIN, speed + 20);

}

else

{

// 좌조향(크게)

direction = direction - 0.003;

pwmWrite(S\_PIN, direction);

softPwmWrite(D\_PIN, speed);

}

}

4.2.11 좌측 차선만 보이는 경우[19]

else if (mag\_r == 0)

{

if ((mag\_l > 30) && (mag\_l < 45))

{

pwmWrite(S\_PIN, 17);

softPwmWrite(D\_PIN, speed + 13);

}

else

{ // 우조향(크게)

direction = direction + 0.003;

pwmWrite(S\_PIN, direction);

softPwmWrite(D\_PIN, speed);

}

}

4.2.12 좌측으로 차선을 이탈할 경우[19]

else if ((mag\_l > 30) && (mag\_l < 45))

{

pwmWrite(S\_PIN, 17);

softPwmWrite(D\_PIN, speed + 10);

}

4.2.13 우측으로 차선을 이탈할 경우

else if ((mag\_r > 30) && (mag\_r < 45))

{

pwmWrite(S\_PIN, 10.5);

softPwmWrite(D\_PIN, speed + 20);

}

}

}

4.2.14. 앞 차와의 간격 때문에 멈춘 경우에는 속도를 올리지 않도록 제어[19]

4.2.14.1. 차의 속력이 부족할 경우 속도를 올림

if((temp == 'g')||(temp == 'r')||(temp == 'l'))

{

4.2.14.2. 앞 차와의 간격 때문에 멈춘 경우에는 속도를 올리지 않도록 제어

if (distance < 30)

{

;

}

4.2.14.3. 앞 차와의 간격 때문이 아니라 속도가 부족해서 멈춘 경우에는 가속 제어

else

{

rig[i] = mag\_r;

lef[i] = mag\_l;

if (((i%3) == 0)&&(i != 0))

{

if (((rig[i] - rig[i-3]) <= 3) && ((lef[i] - lef[i-3]) <= 3))

{

cnt++;

}

else

{

cnt = 0;

speed -= 2;

}

}

if (cnt == 3)

{

speed += 2;

cnt = 0;

}

}

if (speed > MAX\_SPEED)

speed = MAX\_SPEED;

else if (speed < MIN\_SPEED)

speed = MIN\_SPEED;

printf("왼쪽 각 : %d도, 오른쪽 각 : %d도, 거리 : %.2lf, 속도 : %d, 부스터 : %d\n", mag\_l, mag\_r, distance, speed, cnt);

if (i == 16)

i = 0;

else

i++;

}

Else

printf("왼쪽 각 : %d도, 오른쪽 각 : %d도, 거리 : %.2lf, 속도 : %d, 부스터 : %d\n", mag\_l, mag\_r, distance, speed, cnt);

if (cnt < 0)

cnt = 0;

}

free(dir\_data);

close(camera\_sock);

}

void Send(void \*ptr) {

char \*send\_data\_1;

int length, change;

send\_data\_1 = (char \*)malloc(SENDSIZE \* sizeof(char));

assert(send\_data\_1);

while (1)

{

temp = getchar();

while (getchar() != '\n');

send\_data\_1[0] = temp;

length = strlen(send\_data\_1);

if (send(new\_fd, send\_data\_1, length + 1, 0) == -1)

{

perror("send");

free(send\_data\_1);

break;

}

printf("send data = %c\n", send\_data\_1[0]);

//------------------------ Right Change ----------------------------//

if (send\_data\_1[0] == 'r')

{

change\_lane = 'r';

pwmWrite(S\_PIN, 17);

softPwmWrite(D\_PIN, speed + 18);

}

//------------------------ Left Change ----------------------------//

else if (send\_data\_1[0] == 'l')

{

change = ok;

change\_lane = 'l';

while(change == ok);

pwmWrite(S\_PIN, 10.9);

digitalWrite(MOTOR1, 1);

digitalWrite(MOTOR2, 0);

softPwmWrite(D\_PIN, speed + 10);

}

//----------------------------- Go ---------------------------------//

else if (send\_data\_1[0] == 'g')

{

digitalWrite(MOTOR1, 1);

digitalWrite(MOTOR2, 0);

softPwmWrite(D\_PIN, speed);

}

//---------------------------- Stop --------------------------------//

else if (send\_data\_1[0] == 's')

{

digitalWrite(MOTOR1, 0);

digitalWrite(MOTOR2, 0);

softPwmWrite(D\_PIN, 0);

if (argc != 2) {

fprintf(stderr, "usage: ./client<host-name>\n");

exit(1);

}

/\*----------------------------- 카메라 RPI와의 연결 (Client) ----------------------------------\*/

if ((he = gethostbyname(argv[1])) == NULL) {

herror("gethostbyname");

exit(1);

}

if ((camera\_sock = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("socket");

exit(1);

}

camera\_addr.sin\_family = AF\_INET;

camera\_addr.sin\_port = htons(CAPORT);

camera\_addr.sin\_addr = \*((struct in\_addr \*)he->h\_addr);

bzero(&(camera\_addr.sin\_zero), 8);

if (connect(camera\_sock, (struct sockaddr \*)&camera\_addr, sizeof(struct sockaddr)) == -1) {

perror("connect");

exit(1);

}

printf("------------Connecting at %s-----------\n", argv[1]);

/\*---------------------------------------------------------------------------------------\*/

// Thread 생성 및 실행

pthread\_create(&thrd1, NULL, (void \*)Check\_Dis, NULL);

pthread\_create(&thrd2, NULL, (void \*)Video, NULL);

pthread\_create(&thrd3, NULL, (void \*)Detect, NULL);

/\*--------------------------- 2번, 3번차 RPI와의 연결 (Server) --------------------------------\*/

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("socket");

exit(1);

}

my\_addr.sin\_family = AF\_INET;

my\_addr.sin\_port = htons(MYPORT);

my\_addr.sin\_addr.s\_addr = INADDR\_ANY;

bzero(&(my\_addr.sin\_zero), 8);

setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &on, sizeof(on));

if (bind(sockfd, (struct sockaddr \*)&my\_addr, sizeof(struct sockaddr)) == -1) {

perror("bind");

exit(1);

}

if (listen(sockfd, BACKLOG) == -1) {

perror("listen");

exit(1);

}

printf("Server : waiting Second Car\n");

while (1) {

sin\_size = sizeof(struct sockaddr\_in);

if ((new\_fd = accept(sockfd, (struct sockaddr \*)&their\_addr, &sin\_size)) == -1) {

perror("accept");

continue;

}

if ((status = pthread\_create(&thrd4, NULL, (void \*)Send, (void \*)&new\_fd)) != 0) {

printf("4th thread create error: %s\n", strerror(status));

exit(0);

}

}

/\*---------------------------------------------------------------------------------------------\*/

// Thread 종료 대기

pthread\_join(thrd1, NULL);

pthread\_join(thrd2, NULL);

pthread\_join(thrd3, NULL);

close(new\_fd);

close(sockfd);

return 0;

}

void Detect(void \*ptr) {

while (1) {

if ((change\_lane == 'r') || (change\_lane == 'l'))

{

ok++;

softPwmWrite(D\_PIN, speed + 10);

if (mag\_r == 0)

{

if (mag\_l < 48)

{

change\_lane = 'i';

softPwmWrite(D\_PIN, speed);

}

}

}

else

{

//--------- 두 차선이 모두 보이는 경우 ---------//

if (i\_x != 0)

//--- 두 차선의 교점이 오른쪽에 맺힌 경우 ---//

if (i\_x > 390)

{

// 우조향

direction = direction + 0.15;

pwmWrite(S\_PIN, direction);

softPwmWrite(D\_PIN, speed + 1);

}

//--- 두 차선의 교점이 왼쪽에 맺힌 경우 ---//

else

{

// 좌조향

direction = direction - 0.2;

pwmWrite(S\_PIN, direction);

softPwmWrite(D\_PIN, speed);

}

//--------- 한 개의 차선만 보이는 경우 ---------//

else

{

//------ 오른쪽 차선만 보이는 경우 -----//

if (mag\_l == 0)

{

if (mag\_r < 67)

{

// 좌회전

pwmWrite(S\_PIN, 10.8);

softPwmWrite(D\_PIN, speed + 20);

}

else

{

// 좌조향(크게)

direction = direction - 0.003;

pwmWrite(S\_PIN, direction);

softPwmWrite(D\_PIN, speed);

}

}

//------- 왼쪽 차선만 보이는 경우 ------//

else if (mag\_r == 0)

{

if ((mag\_l > 30) && (mag\_l < 45))

{

pwmWrite(S\_PIN, 17);

softPwmWrite(D\_PIN, speed + 13);

}

else

{ // 우조향(크게)

direction = direction + 0.003;

pwmWrite(S\_PIN, direction);

softPwmWrite(D\_PIN, speed);

}

}

//------- 왼쪽으로 차선을 이탈할 경우 ------//

else if ((mag\_l > 30) && (mag\_l < 45))

{

pwmWrite(S\_PIN, 17);

softPwmWrite(D\_PIN, speed + 10);

}

//------ 오른쪽으로 차선을 이탈할 경우 ------//

else if ((mag\_r > 30) && (mag\_r < 45))

{

pwmWrite(S\_PIN, 10.5);

softPwmWrite(D\_PIN, speed + 20);

}

}

}

if (direction > 15) direction = 15;

else if (direction < 14) direction = 14;

}

}

void Video(void \*ptr) {

char \*dir\_data, \*change\_data;

int length, cnt = 0, i = 0;

int rig[17], lef[17];

dir\_data = (char \*)malloc(MAXDATASIZE \* sizeof(char));

assert(dir\_data);

while (1)

{

if (recv(camera\_sock, dir\_data, (MAXDATASIZE \* sizeof(char)), 0) == -1)

{

perror("Receive");

break;

}

length = strlen(dir\_data);

if (send(camera\_sock, dir\_data, length + 1, 0) == -1)

{

perror("send");

break;

}

change\_data = strtok(dir\_data, "/");

mag\_l = atoi(change\_data);

change\_data = strtok(NULL, "/");

mag\_r = atoi(change\_data);

change\_data = strtok(NULL, "/");

i\_x = atoi(change\_data);

// 앞 차와의 간격 때문에 멈춘 경우에는 속도를 올리지 마라.

//----------------------- 차의 속력이 부족할 경우 속도를 올림 --------------------------//

if((temp == 'g')||(temp == 'r')||(temp == 'l'))

{

// 앞 차와의 간격 때문에 멈춘 경우에는 속도를 올리지 마라.

if (distance < 30)

{

;

}

// 앞 차와의 간격 때문이 아니라 속도가 부족해서 멈춘 경우에는 속도를 올려라.

else

{

rig[i] = mag\_r;

lef[i] = mag\_l;

if (((i%3) == 0)&&(i != 0))

{

if (((rig[i] - rig[i-3]) <= 3) && ((lef[i] - lef[i-3]) <= 3))

{

cnt++;

}

else

{

cnt = 0;

speed -= 2;

}

}

if (cnt == 3)

{

speed += 2;

cnt = 0;

}

}

if (speed > MAX\_SPEED)

speed = MAX\_SPEED;

else if (speed < MIN\_SPEED)

speed = MIN\_SPEED;

printf("왼쪽 각 : %d도, 오른쪽 각 : %d도, 거리 : %.2lf, 속도 : %d, 부스터 : %d\n", mag\_l, mag\_r, distance, speed, cnt);

if (i == 16)

i = 0;

else

i++;

}

else

printf("왼쪽 각 : %d도, 오른쪽 각 : %d도, 거리 : %.2lf, 속도 : %d, 부스터 : %d\n", mag\_l, mag\_r, distance, speed, cnt);

if (cnt < 0)

cnt = 0;

}

free(dir\_data);

close(camera\_sock);

}

void Send(void \*ptr) {

char \*send\_data\_1;

int length, change;

send\_data\_1 = (char \*)malloc(SENDSIZE \* sizeof(char));

assert(send\_data\_1);

while (1)

{

temp = getchar();

while (getchar() != '\n');

send\_data\_1[0] = temp;

length = strlen(send\_data\_1);

if (send(new\_fd, send\_data\_1, length + 1, 0) == -1)

{

perror("send");

free(send\_data\_1);

break;

}

printf("send data = %c\n", send\_data\_1[0]);

//------------------------ Right Change ----------------------------//

if (send\_data\_1[0] == 'r')

{

change\_lane = 'r';

pwmWrite(S\_PIN, 17);

softPwmWrite(D\_PIN, speed + 18);

}

//------------------------ Left Change ----------------------------//

else if (send\_data\_1[0] == 'l')

{

change = ok;

change\_lane = 'l';

while(change == ok);

pwmWrite(S\_PIN, 10.9);

digitalWrite(MOTOR1, 1);

digitalWrite(MOTOR2, 0);

softPwmWrite(D\_PIN, speed + 10);

}

//----------------------------- Go ---------------------------------//

else if (send\_data\_1[0] == 'g')

{

digitalWrite(MOTOR1, 1);

digitalWrite(MOTOR2, 0);

softPwmWrite(D\_PIN, speed);

}

//---------------------------- Stop --------------------------------//

else if (send\_data\_1[0] == 's')

{

digitalWrite(MOTOR1, 0);

digitalWrite(MOTOR2, 0);

softPwmWrite(D\_PIN, 0);

}

//-------------------------- 1st Stop ------------------------------//

else if (send\_data\_1[0] == 'q')

{

digitalWrite(MOTOR1, 0);

digitalWrite(MOTOR2, 0);

softPwmWrite(D\_PIN, 0);

}

}

free(send\_data\_1);

close(new\_fd);

close(sockfd);

}

void Check\_Dis(void \*ptr) {

int start\_time, end\_time;

int state = 1;

while (1)

{

// 송신 Pulse 생성

digitalWrite(trig, LOW);

delayMicroseconds(2);

digitalWrite(trig, HIGH);

delayMicroseconds(10);

digitalWrite(trig, LOW);

// 반사파 시간 측정

while (digitalRead(echo) == 0);

start\_time = micros();

while (digitalRead(echo) == 1);

end\_time = micros();

// 측정한 시간을 토대로 거리 계산

distance = (end\_time - start\_time) / 29. / 2.;

if (distance < 30)

{

digitalWrite(MOTOR1, LOW);

digitalWrite(MOTOR2, LOW);

softPwmWrite(D\_PIN, 0);

state = 0;

}

else if ((temp == 'g')||(temp == 'r')||(temp == 'l'))

{

if (state != 1)

{

digitalWrite(MOTOR1, HIGH);

digitalWrite(MOTOR2, LOW);

softPwmWrite(D\_PIN, speed);

state = 1;

}

}

}

4.3. 첫 번째 차량 주행 알고리듬 소스 코드[19]

주행 코드는 차량 간의 통신 코드가 없다. 따라서 한 대의 자동차가 차선인식을 통해 주행하는 코드다. 코드 및 설명은 다음과 같다.

4.3.1 주행 제어 기본 설정

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <netdb.h>

#include <pthread.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#include <assert.h>

#include <softPwm.h>

#include <wiringPi.h>

#include <unistd.h>

#include <math.h>

#define MYPORT 3490 // 2번, 3번차와의 통신 포트

#define CAPORT 5100 // 카메라와의 통신 포트

#define MAXDATASIZE 10 // 카메라와의 통신 메세지 최대 용량

#define SENDSIZE 3 // 차량 간 통신 메세지 최대 용량

#define BACKLOG 4 // 통신 데이터 Queue

#define HIGH 1

#define LOW 0

#define S\_PIN 1 // SERVO PWM

#define D\_PIN 0 // DC PWM

#define MOTOR1 2 // DC direction

#define MOTOR2 3

#define trig 5 //Sensor

#define echo 4

#define MIN\_SPEED 18

#define MAX\_SPEED 70

#define COLUMN 640

//void First(void \*ptr);

void Check\_Dis(void \*ptr);

void Video(void \*ptr);

void Detect(void \*ptr);

void Boost(void \*ptr);

char \*recv\_data\_1;

char change\_lane;

float distance;

int speed = MIN\_SPEED;

float direction = 14.8, i\_x;

int first\_sock, camera\_sock;

int mag\_l, mag\_r, sub;

int ch;

int main(int argc, char \*argv[]) {

int numbytes;

pthread\_t thrd1, thrd2, thrd3, thrd4, thrd5;

struct hostent \*first\_ip, \*camera\_ip;

struct sockaddr\_in firstcar\_addr, camera\_addr;

// wiringPi 설치

if (wiringPiSetup() == -1) return 1;

// GPIO 핀 배정

pinMode(MOTOR1, OUTPUT);

pinMode(MOTOR2, OUTPUT);

pinMode(trig, OUTPUT);

pinMode(echo, INPUT);

pinMode(S\_PIN, PWM\_OUTPUT);

pinMode(BUZZER, OUTPUT);

digitalWrite(BUZZER, HIGH);

// S/W, H/W PWM 설정

softPwmCreate(D\_PIN, 0, 100);

pwmSetMode(PWM\_MODE\_MS);

pwmSetClock(1920);

pwmSetRange(200);

// DC Motor Initialization

digitalWrite(MOTOR1, LOW);

digitalWrite(MOTOR2, LOW);

softPwmWrite(D\_PIN, 0);

if (argc != 2) {

fprintf(stderr, "usage: ./client<host-name>\n");

exit(1);

}

4.3.2. 카메라 RPI와의 연결 (Client)

if ((he = gethostbyname(argv[1])) == NULL) {

herror("gethostbyname");

exit(1);

}

if ((camera\_sock = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("socket");

exit(1);

}

camera\_addr.sin\_family = AF\_INET;

camera\_addr.sin\_port = htons(CAPORT);

camera\_addr.sin\_addr = \*((struct in\_addr \*)he->h\_addr);

bzero(&(camera\_addr.sin\_zero), 8);

if (connect(camera\_sock, (struct sockaddr \*)&camera\_addr, sizeof(struct sockaddr)) == -1) {

perror("connect");

exit(1);

}

printf("------------Connecting at %s-----------\n", argv[1]);

4.3.3. Thread 생성 및 실행[21]

pthread\_create(&thrd1, NULL, (void \*)Check\_Dis, NULL);

pthread\_create(&thrd2, NULL, (void \*)Video, NULL);

pthread\_create(&thrd4, NULL, (void \*)Detect, NULL);

pthread\_create(&thrd5, NULL, (void \*)Boost, NULL);

4.3.4. 두 번째, 세 번째 차량과 RPI와의 연결 (Server)

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("socket");

exit(1);

}

my\_addr.sin\_family = AF\_INET;

my\_addr.sin\_port = htons(MYPORT);

my\_addr.sin\_addr.s\_addr = INADDR\_ANY;

bzero(&(my\_addr.sin\_zero), 8);

setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &on, sizeof(on));

if (bind(sockfd, (struct sockaddr \*)&my\_addr, sizeof(struct sockaddr)) == -1) {

perror("bind");

exit(1);

}

if (listen(sockfd, BACKLOG) == -1) {

perror("listen");

exit(1);

}

printf("Server : waiting Second Car\n");

while (1) {

sin\_size = sizeof(struct sockaddr\_in);

if ((new\_fd = accept(sockfd, (struct sockaddr \*)&their\_addr, &sin\_size)) == -1) {

perror("accept");

continue;

}

if ((status = pthread\_create(&thrd4, NULL, (void \*)Send, (void \*)&new\_fd)) != 0) {

printf("4th thread create error: %s\n", strerror(status));

exit(0);

}

}

\*/

4.3.5. Thread 종료 대기[21]

pthread\_join(thrd1, NULL);

pthread\_join(thrd2, NULL);

// pthread\_join(thrd3, NULL);

pthread\_join(thrd4, NULL);

pthread\_join(thrd5, NULL);

close(new\_fd);

close(sockfd);

return 0;

}

void Detect(void \*ptr) {

while (1) {

4.3.6. 두 차선이 모두 보이는 경우

if (i\_x != 0) //계속 작성 중(2020. 06. 05)

4.4. 첫 번째 영상처리 소스 코드[20]

영상처리 코드는 PYTHON3를 통해설계. 주행 및 통신코드는 C언어로 설계되었다.

따라서, 영상처리 코드는 컴파일 할 필요가 없다. 영상처리 + 주행 코드만 사용하면 1대의 자동차를 영상처리를 활용해서 자율주행 시키는 코드로 사용할 수 있도록 설계했다.

#START

from \_\_future\_\_ import print\_function

import cv2

import numpy as np

import socket

# relative module

import video

# built-in module

import sys

import math

def angle(dx,dy):

return math.atan2(dy,dx)\*180/math.pi

HOST = '172.29.81.21'

PORT = 5100

if \_\_name\_\_ == '\_\_main\_\_':

print(\_\_doc\_\_)

try:

fn = sys.argv[1]

except:

fn = 0

def nothing(\*arg):

pass

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.bind((HOST, PORT))

s.listen(1)

conn, addr = s.accept()

print('Connected by', addr) //계속 작성 중(2020. 06. 05)

**13 주차 참고문헌**

[1] 탁명환(2017),“다중센서를 이용한 군집로봇의 위치인식 및 주행 제어에 관한 연구”

[2] 손영진(2009), “무인자율주행차량을 위한 경로생성방법 및 위치추정방법에 대한 연구 = Design of Path Planning & GPS Estimation Algorithm for Unmanned Autonomous Ground Vehicle”

[3] 오준호(2012),“차량 군집 주행에 적합한 멀티 캐스트 프로토콜”

[4] 이종덕(2015),“행위자기반모형을 활용한 자율주행차량의 교통혼잡 개선효과 분석”

[5] 황원준(2012), “초음파센서 기반 자율주행 로봇의 장애물 회피기술에 관한 연구, A Study on Obstacle Avoidance Technology of Autonomous Traveling Robot Based on Ultrasonic Sensor”

[6] 김운철(2011), “영상처리를 이용한 모형자동차의 자율주행 On the self-driving of a model car using image processing”

[7] 화물차 운전자의 현실, [2017노동자건강권포럼]운수노동자의 적정노동시간.pdf

[8] 이민채, 한양대학교 대학원,(2014) "자율주행시스템을 위한 다중센서 정보융합기반 실시간 영상처리 시스템 구조, A real-time vision system aRChitecture for autonomous driving systems based on multi-sensor information fusion"

[9] “NXP-DAF 트럭, 트럭 플래투닝 주행 계획 발표...인간보다 30배 빠른 반응 속도 목표”, Hello T 첨단뉴스, 2016.11.11

[10] Socket Programming in C, http://www.geeksforgeeks.org/socket-programming-cc/

[11] 전희종(1994) 대한전기학회 1994년도 하계학술대회 논문집, A Study On PIN Pulse Pattern Optimization In The Space Vector Notation Using Pulse Frequency Modulation

[12] 강병찬(2005) 한국방송공학회 2005년도 학술대회, Hough Transform을 이용한 차선 검출의 고속화에 관한 연구, A Study on high speedization of lane detection using Hough Transform

[13] 박미영(2011), 한국정보통신학회, 이미지 퍼지화 기반 Canny에지 검출기 설계에 관한 연구, A Study on Canny Edge Detector Design Based on Image Fuzzification

[14] 김기석, “허프변환과 차선모델을 이용한 효과적인 차선검출에 관한 연구”, 정보 및 제어 논문집, pp. 34-36, 2009

[15] 안수진, “자율주행차량을 위한 차선인식에 관한 연 구”, 한국 정보기술학회 논문지, 5(1), pp. 136-142, 2007

[16] 우영운, “퍼지 이진화와 허프 변환을 이용한 주행 경로 검출”, 한국컴퓨터 정보학회논문지, 19(2), pp. 31-37, 2014

[17] 박진천, “차량 간 거리 제어를 이용한 군집 주행”, 한국전자통신학회 논문지, 2018년, pp.1071 - 1078

[18] 김태영, “영상 처리와 분석을 통한 자율 주행 알고리즘 연구”, 한국통신학회 학술대회논문지, 2019년, pp.114 – 115

[19] 손영섭, “자율주행 자동차의 전기적 파워 조향 시스템을 위한 제어 기법의 개관”, 제어로봇시스템학회 학술대회논문지, 2015, pp.31 -36

[20] 김운철, “On the self-driving of a model car using image processing”, 2011 제26회 ICROS 학술대회, 2011.05, pp.660 -663(4 pages)

[21] 정석우, “자율주행 자동차의 인공지능”, 기계저널 제57권 제3호, 2017, pp.42 – 45(4 pages)