**창업연계공학설계입문**

|  |  |
| --- | --- |
| 프로젝트 명 | *창업연계공학설계입문* |
| 팀 명 | *ASA* |
| 문서 제목 | 4차 과제 보고서 |

|  |  |
| --- | --- |
| **Version** | 1.7.4 |
| **Date** | 2017-11-16 |

|  |  |
| --- | --- |
| **팀원** | 신민욱(조장) / 20171640 |
| 안중민/ 20171644 |
| 안희운/ 20171646 |
| **지도교수** | 한재일 교수 |
| **분반** | 3 분반 |

|  |
| --- |
| **CONFIDENTIALITY/SECURITY WARNING**  이 문서에 포함되어 있는 정보는 국민대학교 소프트웨어융합대학 소프트웨어학부 및 소프트웨어학부 개설 교과목 공학설계입문 수강 학생 중 프로젝트 “창업연계공학설계입문”을 수행하는 팀 “ASA”의 팀원들의 자산입니다. 국민대학교 소프트웨어학부 및 팀 “ASA”의 팀원들의 서면 허락없이 사용되거나, 재가공 될 수 없습니다. |

**문서 정보 / 수정 내역**

|  |  |
| --- | --- |
| **Filename** | ASA-4차과제-창업연계공학입문설계.docx |
| **원안작성자** | 신민욱, 안희운 |
| **수정작업자** | 신민욱, 안희운, 안중민 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 수정날짜 | 대표수정자 | Revision | 추가/수정 항목 | 내 용 |
| 2017-11-06 | 신민욱 | 1.0 | 최초 작성 | 팀 기초 내용 작성 |
| 2017-11-07 | 신민욱 | 1.1 | 서론 작성 | 서론 기초 내용 작성 |
| 2017-11-09 | 신민욱 | 1.2 | 아이디어 | 아이디어 기초 내용 작성 |
| 2017-11-09 | 신민욱 | 1.4 | 추진, 건의 | 추진계획, 건의사항 기초내용 작성 |
| 2017-11-10 | 신민욱 | 1.4.1 | 추진계획 | 추진계획 추가 작성 |
| 2017-11-10 | 신민욱 | 1.4.2 | 회의록 | 회의록 기초 작성 |
| 2017-11-10 | 신민욱 | 1.5 | 수행내용 | 수행내용 기초 내용 작성 |
| 2017-11-10 | 안희운 | 1.6 | 수행내용 | 수행내용 추가 작성 및 오타 수정 |
| 2017-11-11 | 안희운 | 1.6.1 | 수행내용 | 수행내용 추가 작성 |
| 2017-11-11 | 안희운 | 1.7 | 회의록 | 회의록 작성 및 수행내용 수정 |
| 2017-11-11 | 안희운 | 1.7.1 | 회의록 | 회의록 작성 |
| 2017-11-11 | 안희운 | 1.7.2 | 회의록 | 회의록 추가작성 |
| 2017-11-12 | 안희운 | 1.7.3 | 수행내용 | 수행내용 수정 및 작성 |
| 2017-11-13 | 안희운 | 1.7.4 | 건의사항 | 건의사항 추가 작성 |
| 2017-11-13 | 안희운 | 1.7.4 | --- | 오타 수정 및 마무리 |

**목 차**

[**1 서론 4**](#_Toc497924334)

[**2 기본 아이디어 5**](#_Toc497924335)

[**2.1 H/W 디자인 방향 5**](#_Toc497924336)

[**2.2 S/W 디자인 방향 5**](#_Toc497924337)

[**3 수행 내용 6**](#_Toc497924338)

[**3.1 직교형 장애물 인식을 이용한 직각 코스 주행 6**](#_Toc497924339)

[**3.2 장애물 인식 및 회피 6**](#_Toc497924340)

[**3.3 곡률 코스 처리 6**](#_Toc497924341)

[**4 향후 추진계획 7**](#_Toc497924342)

[**4.1 향후 계획의 세부 내용 7**](#_Toc497924343)

[**5 애로 및 건의사항 8**](#_Toc497924344)

[**6 회의록 9**](#_Toc497924345)

# **서론**

정보통신과 산업 기술의 융합으로 이루어지는 4차 산업 혁명이 다가오면서 메이커 운동이 뜨고 있다.

메이커 운동이란, 메이커들이 일상에서 창의적 만들기를 실천하고 자신의 경험과 지식을 나누고 공유하려는 경향을 의미하며, 라즈베리 파이를 이용한 자율주행 자동차가 이에 부합한다.

이번 장애물 회피 라인 트레이싱 과제는 검은 라인을 따라 방해물을 회피하며 자율 주행해야 한다.

이를 위해 초음파 센서, 광센서를 이용하여 라인 트레이서와 장애물을 회피를 구현한다. 또한 다수의 인원이 모여서 진행하므로, 깃허브를 이용하여 팀원들과 효과적으로 코드를 이해하고 전달할 수 있게 하여 서로 간의 의사소통이 활발하게 진행하도록 준비한다.

다만 현실적인 문제로 인해 미리 설정된 트랙을 따라서만 주행할 수 있고, 광센서의 인식 가능한 개수의 한계로 인해 한정된 경우의 수만 처리할 수 있다.

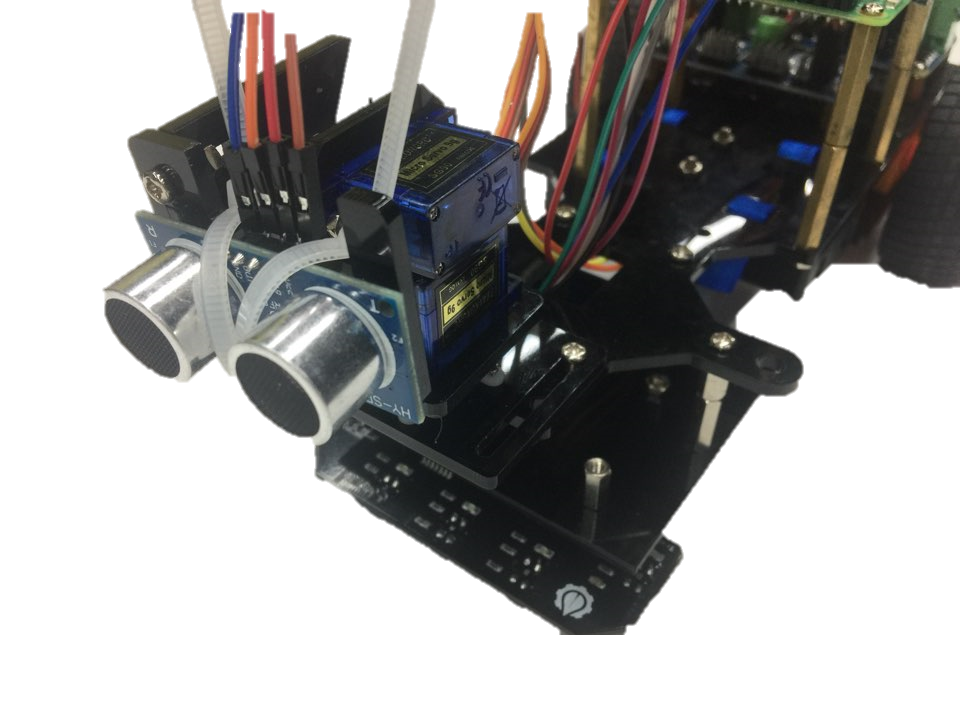
# **기본 아이디어**

구동체가 주어진 검은 선을 따라가며 가로막은 장애물을 피해야 한다.

검은 선은 검은색을 감지하는 센서로 인지하여 주행하도록 한다.

앞에 장애물이 있다면, 장애물을 인지할 수 있는 초음파 센서로 인지하여 ㄷ자 주행을 한다.

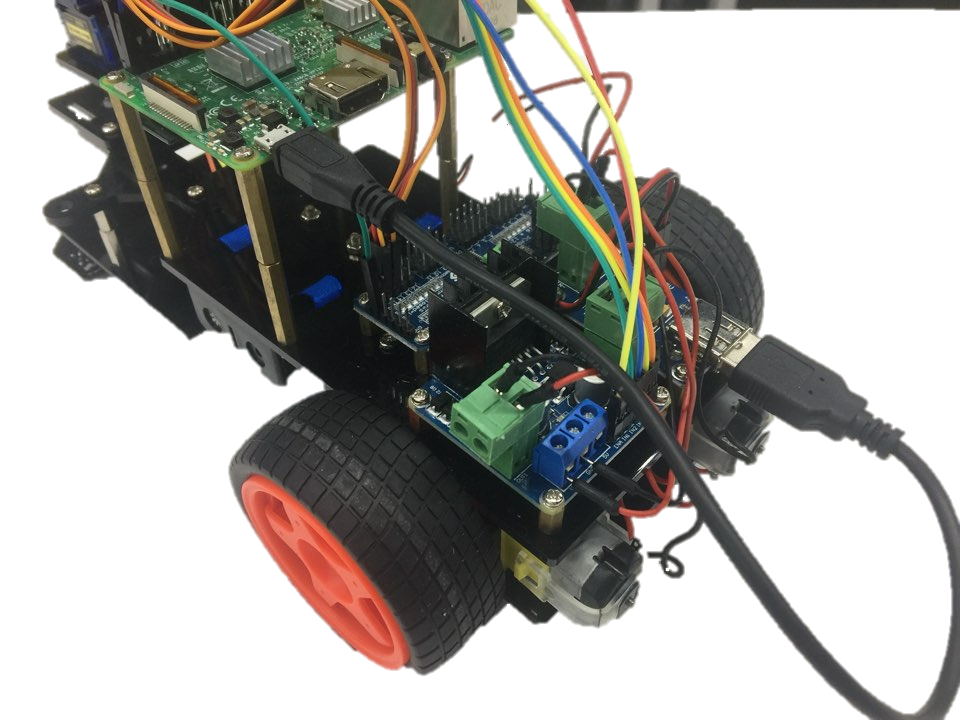
## **H/W 디자인 방향**



<그림1:구동체 앞부분(front)>

하단의 검은 선을 인식하기 위해 광센서를 바닥에서 가장 가까운 곳에 탑재하고, 장애물을 인식하기 위해 장애물의 높이에 부합한 구동체의 앞부분에 초음파 센서를 탑재한다.

또한 앞바퀴는 볼을 장착하여 유동적으로 움직이게 조치하였다.



<그림2:구동체 뒷부분(back)>

구동체가 안정적으로 주행하기 위해 뒷바퀴의 모터로 속도를 제어하여 커브 턴도 구현한다.

## **S/W 디자인 방향**

기본적인 주행뿐만 아니라, 구동체의 움직임에 따라 다양한 변수가 존재하므로 최대한 많은 경우의 수를 고려해야 한다.

광센서에서 받아오는 5개의 값(0, 1)과 초음파 센서의 거리값을 이용하여 while, if, elif를 사용하는 알고리즘을 수행해야 한다.

def go\_forward\_any(speed1, speed2):

"""

Forward module

:param speed: Motor running speed

"""

leftmotor(forward0)

leftmotor(forward1)

GPIO.output(MotorLeft\_PWM, GPIO.HIGH)

rightmotor(forward0)

GPIO.output(MotorRight\_PWM, GPIO.HIGH)

LeftPwm.ChangeDutyCycle(speed1)

RightPwm.ChangeDutyCycle(speed2)

자주 쓰는 구동체 주행 코드는 모듈(def)로 패키징하여 코드의 재사용성을 향상한다.

# Main

try:

while True:

print("leftmostled detects black line(0) or white ground(1): " + str(GPIO.input(leftmostled)))

print("leftlessled detects black line(0) or white ground(1): " + str(GPIO.input(leftlessled)))

print("centerled detects black line(0) or white ground(1): " + str(GPIO.input(centerled)))

print("rightlessled detects black line(0) or white ground(1): " + str(GPIO.input(rightlessled)))

print("rightmostled detects black line(0) or white ground(1): " + str(GPIO.input(rightmostled)))

distance = getDistance()

print('distance= ', distance)

# when the distance is above the dis, moving object forwards

if (distance > dis):

linetracing()

# when the distance is below the dis, moving object stops

else:

stop()

time.sleep(1)

leftPointTurn(34, 0.38)

time.sleep(1)

go\_forward(38, 0.3)

time.sleep(1)

rightPointTurn(33, 0.35)

time.sleep(1)

go\_forward(42, 0.6)

time.sleep(1)

rightPointTurn(31, 0.35)

time.sleep(1)

go\_forward(30, 0.28)

time.sleep(1)

leftPointTurn(28, 0.35)

while, if, else로 초음파 센서의 거리 값을 설정된 변수와 비교하며 설정된 변수보다 멀리 있을 경우, 모듈로 묶은 라인 트레이싱 코드를 사용하여 주행한다.

라인 트레이싱 또한 광센서로 지속적으로 인식 (0,1) 값을 얻어야 한다.

# Keyboard Interrupt

except KeyboardInterrupt:

# the speed of left motor will be set as LOW

GPIO.output(MotorLeft\_PWM, GPIO.LOW)

# left motor will be stopped with function of ChangeDutyCycle(0)

LeftPwm.ChangeDutyCycle(0)

# the speed of right motor will be set as LOW

GPIO.output(MotorRight\_PWM, GPIO.LOW)

# right motor will be stopped with function of ChangeDutyCycle(0)

RightPwm.ChangeDutyCycle(0)

# GPIO pin setup has been cleared

GPIO.cleanup()

만약 실행 중에 키보드로 Ctrl+c를 눌렀다면 예외처리가 진행되어 모터의 파워를 낮춰주고, 멈출 준비를 하면서 pin 설정을 클리어해준다.

# **수행 내용**

구동체가 시계방향으로 도는 경우랑 반시계방향으로 도는 경우에 따른 차이가 발생하므로 각각의 상황에 대한 코드를 따로 작성하였다.

## **직교형 장애물 인식을 이용한 직각 코스 주행**

구동체가 주행하면서 광센서가 검은색을 인식하면 광센서가 많이 반응하는 방향으로 움직일 수 있도록 광센서가 반응하는 각각의 상황에 맞춰 뒷바퀴의 속도를 다르게 조정하였다.

a = int(GPIO.input(leftmostled))

b = int(GPIO.input(leftlessled))

c = int(GPIO.input(centerled))

d = int(GPIO.input(rightlessled))

e = int(GPIO.input(rightmostled))

각각의 광센서가 검은색에 반응할 경우 0, 하얀색에 반응할 경우를 1로 하였고 구동체가 나아가는 방향으로 보아 가장 좌측의 광센서가 a, 가장 우측의 광센서가 e가 되도록 변수를 설정하였다.

elif (a == 1) & (b == 0) & (c == 0) & (d == 0) & (e == 1):

go\_forward\_any(40, 40)

elif (a == 1) & (b == 1) & (c == 0) & (d == 1) & (e == 1):

go\_forward\_any(40, 40)

만약 검은색 선이 구동체의 가운데에 위치하게 될 경우 구동체의 뒷바퀴의 좌우 모터의 속도를 동일하게 설정해 직진할 수 있도록 한다.

elif (a == 1) & (b == 1) & (c == 1) & (d == 1) & (e == 0):

go\_forward\_any(40, 27)

elif (a == 1) & (b == 1) & (c == 1) & (d == 0) & (e == 0):

go\_forward\_any(40, 30)

우측의 광센서들만 검은색을 인식할 경우 검은색 선이 구동체에 우측에 위치한다는 것을 의미한다. 그러므로 구동체의 가운데에 검은 선이 있어야하니 구동체를 구동체의 가운데에 선이 올 때까지 우측으로 움직여 주어야 한다. 그래서 뒷바퀴의 좌측 모터의 속도를 우측보다 빠르게 하여 구동체를 우측으로 움직여준다.

elif (a == 0) & (b == 0) & (c == 1) & (d == 1) & (e == 1):

go\_forward\_any(13, 50)

elif (a == 0) & (b == 1) & (c == 1) & (d == 1) & (e == 1):

go\_forward\_any(12, 50)

마찬가지로 좌측의 광센서들만 검은색을 인식할 경우 검은색 선이 구동체에 좌측에 위치한다는 것을 의미한다. 그러므로 구동체의 뒷바퀴의 우측 모터의 속도를 좌측보다 빠르게 하여 구동체를 좌측으로 움직여준다.

**.**

## **장애물 인식 및 회피**

if (distance > dis):

linetracing()

else:

stop()

time.sleep(1)

rightPointTurn(34, 0.38)

time.sleep(1)

go\_forward(38, 0.3)

time.sleep(1)

leftPointTurn(35, 0.35)

time.sleep(1)

go\_forward(42, 0.8)

time.sleep(1)

leftPointTurn(32, 0.35)

time.sleep(1)

go\_forward(30, 0.28)

time.sleep(1)

rightPointTurn(28, 0.35)

위 코드로 인식 결과를 부등호(>)로 계속 비교하여, 초음파 센서와 장애물 사이의 거리가 설정된 dis값보다 적으면 장애물을 피해 좌측으로 ㄷ자 주행한다. 그리고 다시 검은색 선을 따라 주행을 계속한다. 그리고 계속 테스트해본 결과 바퀴 두개를 이용하는 포인트 턴이 바퀴 하나를 이용하는 스윙 턴보다 구동체의 회전율의 오차가 적은 것이 분명해 포인트 턴을 이용하여 ㄷ자 주행을 구현하였다.

## **곡률 코스 처리**

구동체가 직진 주행을 하다 곡선 코스를 만난다면, 위에 3-1에서와 같이 검은색 선을 인식하면서 주행한다. 하지만 센서의 반응 속도가 재빠르지 않아 검은색 선을 따라가다 곡률 코스에서 바깥쪽으로 검은색 선을 벗어나 버리는 일이 많이 발생했다.

elif (a == 0) & (b == 0) & (c == 0) & (d == 1) & (e == 1):

go\_forward\_any(40, 50) #반시계방향

elif (a == 1) & (b == 1) & (c == 0) & (d == 0) & (e == 0):

go\_forward\_any(50, 35) #시계방향

그래서 구동체가 곡률 코스를 만나면 회전을 더 잘 할 수 있도록 하였다. 반시계방향의 회전일 경우, 곡률 코스를 만나고 구동체의 광센서가 미처 반응하지 못해 직진주행을 할 시, 구동체의 좌측에서부터의 광센서 3개가 검은색 선을 반응하므로 구동체가 반시계방향으로 돌 수 있도록 하였고 , 시계방향의 회전일 경우에는 구동체가 시계방향으로 회전할 수 있도록 하였다.

elif (a == 1) & (b == 1) & (c == 1) & (d == 1) & (e == 1):

go\_forward\_any(0, 40) #반시계방향

elif (a == 1) & (b == 1) & (c == 1) & (d == 1) & (e == 1):

go\_forward\_any(40, 0) #시계방향

또한 구동체가 곡률 코스에서 완전히 검은색 선을 벗어날 경우에는 다시 코스에 복귀해야만 한다. 그래서 반시계방향의 경우에는 좌회전을, 시계방향의 경우에는 우회전하여 다시 코스에 복귀할 수 있도록 하였다.

# **향후 추진계획**

## **향후 계획의 세부 내용**

현재, 광센서의 개수 제한과 성능상의 문제로 세밀한 조정이 불가능하여 예상 범위에서 벗어난다면 최적화된 움직임으로 라인에 복귀하지는 못한다.

다수의 광센서를 장착하고 센서의 출력값을 세밀하게 받아온다면, 라인에서 벗어나는 상황을 세밀화해 최적화된 움직일 수 있을 것이다.

또한 소프트웨어적으로도 머신러닝을 이용하여 다양한 데이터를 축적해놓고 구동한다면 예상치 못한 순간 대처가 유연할 것이다.

# **애로 및 건의사항**

라즈베리 파이와 모터, 센서들이 같은 전원 공급처인 건전지를 사용하여 구동체의 구동 시간이 낮다.

만약 보조 전원을 공급할 방안이 있다면, 구동체의 구동 시간을 증가시킬 수 있을 것이다.

또한, 배터리가 소모될 때 전압이 급격히 떨어지는 구간이 있어 라즈베이 파이와의 전압이 맞지 않아 구동체가 느려지는 현상이 발생한다.

그리고 소프트웨어적으로 단일 동작을 수행하는 타 보드(아두이노)와 다르게 다른 서비스도 같이 활성화되어 있는 라즈베리파이의 라즈비안(rpi+debian)운영체제의 특성상 오작동의 빈도가 상대적으로 높다.

이와 같은 상황은, 마이크로 컨트롤러에 비슷한 보드로 실습하여 오작동이 적게 실습할 수 있을 것이다.

# **회의록**

|  |  |  |  |
| --- | --- | --- | --- |
| **일 시** | 2017-11-08 | **차수** | 1 |
| **장 소** | 국민대학교 7호관 442호실 | | |
| **참석자** | 신민욱, 안중민, 안희운 | | |
| **불참자** | NULL | | |
| **안 건** | 곡률 주행과 장애물인식, 그리고 주행 속도에 대하여 | | |
| **회의내용** | <곡률 주행>  라인 트레이싱의 경우 구동체를 구동해 본 결과 이상이 없었음. 하지만 곡률 주행의 경우 광센서가 검은색 선을 벗어난다는 것을 즉각 반응하지 못하여 구동체가 라인 바깥으로 벗어나서 주행을 하는 경우가 많이 발견되었음. 그래서 예방적으로 모든 광센서가 하얀색을 가리킬 경우에도 검은 선을 따라갈 수 있도록 회전을 하는 코드를 작성하였음.그리고 최대한 라인을 따라갈 수 있도록 전체적으로 구동체의 뒷바퀴의 좌우측 모터의 속도를 줄임.  <장애물 인식 주행>  구동체의 초음파 센서가 장애물을 인식한 후 회전을 할 때 시험을 하는 아크릴판과 바퀴 사이의 마찰이 부족하여 바퀴가 헛도는 현상이 많이 발견되었음. 그러나 이 현상을 고칠 수단은 아크릴판을 교체하는 것밖에 없으므로 바퀴가 헛도는 경우를 제외하고 회전률을 수정하는 것으로 결정함. 또한 과제 평가 시간 때 즉각적으로 오차를 수정해가면서 수행하기로 함. 또한 만일 배터리의 전력이 부족하여 구동체의 회전율이 줄어들 것을 염려해 여분의 배터리를 충전해 놓을 것을 결정함.  <주행 속도>  주행 속도를 올리는 것 또한 평가의 대상이 될 수 있기에 주행 속도를 올리는 것을 논의해 보았으나 곡률 주행에서 구동체가 트랙을 벗어날 것을 우려해 속도는 조정하지 않음. | | |
| **결과물** | 구동체의 광센서가 검은색 선을 감지하면 검은색 선을 따라 직진 주행을 함. 직진 주행을 하면서 구동체의 전방에 부착된 초음파 센서가 방해물을 탐지하면 ㄷ자 턴을 이용하여 방해물을 지나치고 다시 검은색 선을 따라 주행함. 그리고 곡률 주행의 경우 구동체의 광센서가 반응할 수 있도록 속도를 낮추었음. 그리고 만약 모든 광센서가 검은색을 가리키지 못할지라도 다시 검은색 선을 따라가도록 만들어 구동체가 성공적으로 곡률 주행을 할 수 있도록 함. | | |