**창업연계공학설계입문**

|  |  |
| --- | --- |
| 프로젝트 명 | *창업연계공학설계입문* |
| 팀 명 | *ASA* |
| 문서 제목 | 5차 과제 보고서 |

|  |  |
| --- | --- |
| **Version** | 1.6 |
| **Date** | 2017-12-03 |

|  |  |
| --- | --- |
| **팀원** | 신민욱(조장) / 20171640 |
| 안중민 / 20171644 |
| 안희운 / 20171646 |
| **지도교수** | 한재일 교수님 |
| **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-5차과제-창업연계공학설계입문.docx |
| **원안작성자** | 신민욱, 안희운 |
| **수정작업자** | 신민욱, 안중민, 안희운 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 수정날짜 | 대표수정자 | Revision | 추가/수정 항목 | 내 용 |
| 2017-11-20 | 신민욱 | 1.0 | 최초 작성 | 최초 작성 |
| 2017-11-20 | 신민욱 | 1.1 | 서론 | 서론 작성 |
| 2017-11-21 | 신민욱 | 1.2 | 건의 사항 | 건의 사항 작성 |
| 2017-11-21 | 신민욱 | 1.3 | 향후 계획 | 향후 계획 작성 |
| 2017-11-22 | 신민욱 | 1.3.5 | 기본 아이디어 | 하드웨어 디자인 작성 |
| 2017-11-30 | 신민욱 | 1.4 | 기본 아이디어 | S/W 디자인 작성 |
| 2017-11-30 | 신민욱 | 1.4.5 | 수행 내용 | 프로그램 코드 첨부 |
| 2017-12-03 | 신민욱 | 1.5.5 | 회의록 | 지난 회의 기록 작성 |
| 2017-12-03 | 신민욱 | 1.5.6 | 오타 수정 | 오타 수정 |
| 2017-12-03 | 신민욱 | 1.6 | 수행 내용 | 코드 설명 작성 |

**목 차**

[**1 서론 4**](#_Toc497924411)

[**2 기본 아이디어 5**](#_Toc497924412)

[2.1 H/W 디자인 방향 5](#_Toc497924413)

[2.2 S/W 디자인 방향 5](#_Toc497924414)

[**3 수행 내용 6**](#_Toc497924415)

[3.1 프로그램 코드 6](#_Toc497924416)

[3.2 코드 설명 6](#_Toc497924417)

[**4 향후 추진계획 7**](#_Toc497924418)

[4.1 향후 계획의 세부 내용 7](#_Toc497924419)

[**5 애로 및 건의사항 8**](#_Toc497924420)

[**6 회의록 9**](#_Toc497924421)

# **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 미로찾기 과제는 검은 라인을 따라 자율 주행하면서 주어진 미로를 골인 지점까지 주행하여 탈출해야 한다.

이를 위해 광센서를 이용하여 라인 트레이서를 구현한다.

또한 다수의 인원이 모여서 진행하므로, 깃허브를 이용하여 팀원들과 효과적으로 코드를 이해하고 전달할 수 있게 하여 서로 간의 의사소통이 활발하게 진행하도록 준비한다.

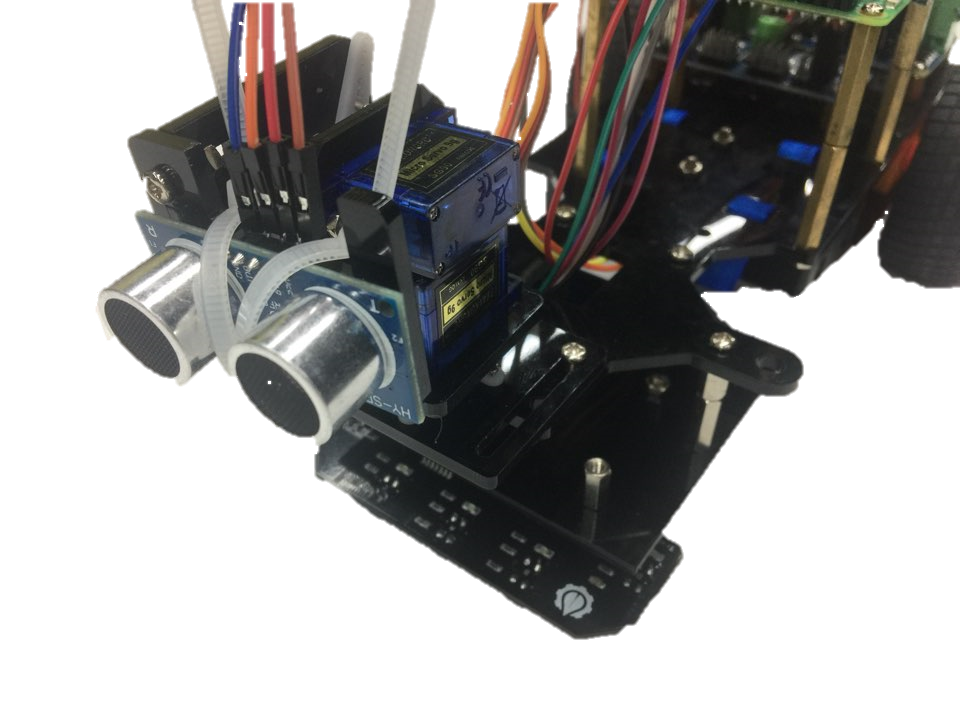
다만 현실적인 문제로 인해 특정한 미로 알고리즘으로만 주행할 수 있고, 광센서의 인식 가능한 개수의 한계로 인해 한정된 경우의 수만 처리할 수 있다.

# **기본 아이디어**

구동체가 주어진 미로로 이루어진 검은 선을 따라가며 탈출구를 찾아야 한다.

검은 선은 검은색을 감지하는 센서로 인지하여 선을 따라가도록 주행하며, 주행할 검은 선이 끊겨서 있다면 막다른 길로 인지하여 구동체로 180도를 돌아서 다시 주행한다.

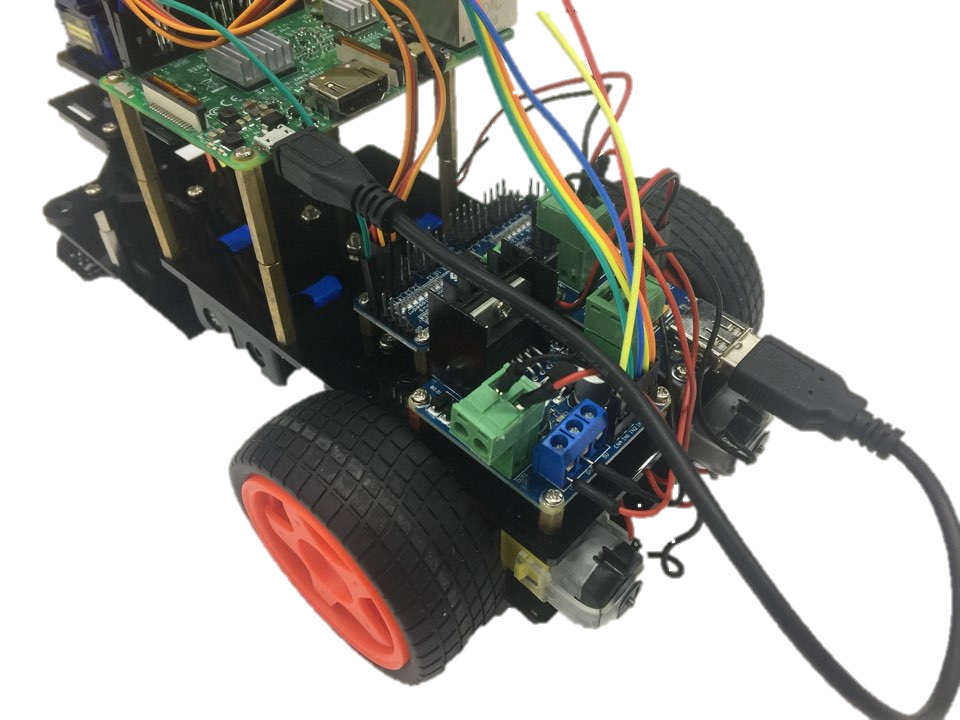
## **H/W 디자인 방향**



하단의 검은 선을 인식하기 위해 광센서를 바닥에서 가장 가까운 곳에 탑재하고,

현재 과제에서는 필요 없지만, 이전 과제에서 쓰였던 초음파 센서도 탑재했다.

또한 앞바퀴는 볼을 장착하여 유동적으로 움직이게 조치하였다.



구동체가 안정적으로 주행하기 위해 뒷바퀴의 모터로 속도를 제어하여 포인트 턴을 구현한다.

## **S/W 디자인 방향**

기본적인 주행뿐만 아니라, 미로 구조에 따른 다양한 변수가 존재하므로 최대한 많은 경우의 수를 고려해야 한다.

광센서에서 받아오는 5개의 값(0, 1)을 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)

이와 같이 자주 쓰는 구동체 주행 코드들은 모듈(def)로 패키징하여 코드의 재사용성을 향상한다.

# Main

try:

while True:

# Importing the determine way modules code

determining\_cross()

모듈로 묶은 갈림길 결정 코드를 사용하여 주행한다.

이번 과제에서는 지속적으로 갈림길을 판별하는 코드로 작동하며, 특정 조건문으로 판별하여 라인 트레이싱 모듈은 부분적으로 동작하게 한다.

그러므로 라인 트레이싱 모듈은 보조 코드로 동작하면서 광센서에서 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**

## **프로그램 코드**

# Import raspberry pi's GPIO module and time module

import RPi.GPIO as GPIO

import time

# Disable warning text

GPIO.setwarnings(False)

# GPIO mode setting

GPIO.setmode(GPIO.BOARD)

# Trig, echo pin number

trig = 33

echo = 31

# Trig, echo in/out setting

GPIO.setup(trig, GPIO.OUT)

GPIO.setup(echo, GPIO.IN)

def REVERSE(x):

"""

Reverse for forward

:return: boolean

"""

if x == True:

return False

elif x == False:

return True

# Forward boolean

forward0 = True

forward1 = False

# Backward define

backward0 = REVERSE(forward0)

backward1 = REVERSE(forward1)

# Left motor pin number

MotorLeft\_A = 12

MotorLeft\_B = 11

MotorLeft\_PWM = 35

# Right motor pin number

MotorRight\_A = 15

MotorRight\_B = 13

MotorRight\_PWM = 37

def leftmotor(x):

"""

Left motor main define

"""

if x == True:

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

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

elif x == False:

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

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

else:

print

'Config Error'

def rightmotor(x):

"""

Right motor main define

"""

if x == True:

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

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

elif x == False:

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

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

# Left motor's GPIO out port setting

GPIO.setup(MotorLeft\_A, GPIO.OUT)

GPIO.setup(MotorLeft\_B, GPIO.OUT)

GPIO.setup(MotorLeft\_PWM, GPIO.OUT)

# Right motor's GPIO out port setting

GPIO.setup(MotorRight\_A, GPIO.OUT)

GPIO.setup(MotorRight\_B, GPIO.OUT)

GPIO.setup(MotorRight\_PWM, GPIO.OUT)

# Motor power up setting

LeftPwm = GPIO.PWM(MotorLeft\_PWM, 100)

RightPwm = GPIO.PWM(MotorRight\_PWM, 100)

def rightSwingTurn(speed1, speed2, running\_time):

"""

Right swing turn main module

:param speed1: Left speed

:param speed2: Right speed

:param running\_time: motor running time

"""

leftmotor(forward1)

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

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

LeftPwm.ChangeDutyCycle(speed1)

RightPwm.ChangeDutyCycle(speed2)

time.sleep(running\_time)

def leftSwingTurn(speed1, speed2, running\_time):

"""

Left swing turn main module

:param speed1: Right speed

:param speed2: Left speed

:param running\_time: motor running time

"""

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

rightmotor(forward0)

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

LeftPwm.ChangeDutyCycle(speed2)

RightPwm.ChangeDutyCycle(speed1)

time.sleep(running\_time)

def rightPointTurn(speed, running\_time):

"""

Right point turn main module

:param speed: motor running speed

:param running\_time: motor running time

"""

leftmotor(forward1)

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

rightmotor(backward0)

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

LeftPwm.ChangeDutyCycle(speed)

RightPwm.ChangeDutyCycle(speed)

time.sleep(running\_time)

def rightPointTurn\_any(speed):

"""

Right point turn main module

:param speed: motor running speed

"""

leftmotor(forward1)

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

rightmotor(backward0)

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

IO = 1

while IO == 1:

LeftPwm.ChangeDutyCycle(speed)

RightPwm.ChangeDutyCycle(speed)

time.sleep(0.01)

c = GPIO.input(centerled)

IO = c

def leftPointTurn(speed, running\_time):

"""

Left point turn main module

:param speed: Motor running speed

:param running\_time: Motor running time

"""

rightmotor(forward0)

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

leftmotor(backward1)

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

LeftPwm.ChangeDutyCycle(speed)

RightPwm.ChangeDutyCycle(speed)

time.sleep(running\_time)

def leftPointTurn\_any(speed):

"""

Left point turn main module

:param speed: Motor running speed

"""

rightmotor(forward0)

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

leftmotor(backward1)

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

IO = 1

while IO == 1:

LeftPwm.ChangeDutyCycle(speed)

RightPwm.ChangeDutyCycle(speed)

time.sleep(0.01)

c = GPIO.input(centerled)

IO = c

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)

def go\_forward(speed, running\_time):

"""

Forward module (time limit)

:param speed: Motor running speed

:param running\_time: Motor running time

"""

leftmotor(forward0)

leftmotor(forward1)

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

rightmotor(forward0)

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

LeftPwm.ChangeDutyCycle(speed)

RightPwm.ChangeDutyCycle(speed)

time.sleep(running\_time)

def stop():

"""

Stop module

"""

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

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

LeftPwm.ChangeDutyCycle(0)

RightPwm.ChangeDutyCycle(0)

def pwm\_setup():

"""

Pwm setup module

"""

LeftPwm.start(0)

RightPwm.start(0)

# Running pwm setup

pwm\_setup()

def pwm\_low():

"""

Pwm low module

"""

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

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

LeftPwm.ChangeDutyCycle(0)

RightPwm.ChangeDutyCycle(0)

GPIO.cleanup()

def linetracing():

"""

Line tracing module

0 = black, 1 = white

"""

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))

stop()

time.sleep(0.1)

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

go\_forward\_any(33, 38)

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

go\_forward\_any(33, 38)

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

go\_forward\_any(38, 33)

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

go\_forward\_any(38, 33)

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

go\_forward\_any(38, 38)

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

go\_forward\_any(38, 38)

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

leftPointTurn\_any(36)

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

rightPointTurn\_any(36)

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

leftPointTurn\_any(36)

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

rightPointTurn\_any(36)

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

go\_forward\_any(32, 32)

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

go\_forward\_any(32, 32)

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

go\_forward\_any(32, 32)

def determining\_cross():

"""

Determine cross road module

"""

a = int(GPIO.input(leftmostled)) # d

b = int(GPIO.input(leftlessled)) # b

c = int(GPIO.input(centerled)) # a

d = int(GPIO.input(rightlessled)) # c

e = int(GPIO.input(rightmostled)) # e

linetracing()

time.sleep(0.5)

if e == 0:

go\_forward(40, 0.45)

rightPointTurn(37, 0.5)

rightPointTurn\_any(37)

elif a == 0:

while a == 1:

linetracing()

if c == 0:

linetracing()

elif c == 1:

go\_forward(40, 0.45)

leftPointTurn(37, 0.5)

leftPointTurn\_any(37)

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

rightPointTurn\_any(42)

else:

linetracing()

# 5-way tracking sensor's pin number

dis = 15

leftmostled = 16

leftlessled = 18

centerled = 22

rightlessled = 40

rightmostled = 32

# GPIO input setting

GPIO.setup(leftmostled, GPIO.IN)

GPIO.setup(leftlessled, GPIO.IN)

GPIO.setup(centerled, GPIO.IN)

GPIO.setup(rightlessled, GPIO.IN)

GPIO.setup(rightmostled, GPIO.IN)

# Main

try:

while True:

# Debug for 5-way tracking sensor code

print("\nleftmostled : " + str(GPIO.input(leftmostled)))

print("leftlessled : " + str(GPIO.input(leftlessled)))

print("centerled : " + str(GPIO.input(centerled)))

print("rightlessled: " + str(GPIO.input(rightlessled)))

print("rightmostled: " + str(GPIO.input(rightmostled)))

# Importing the determine way modules code

determining\_cross()

# 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()

## **코드 설명**

전체적으로 중요 코드에 주석을 달아서 팀원들이 코드만 봐도 이해하도록 코딩하였고, def로 감싼 함수는 대부분 저번 과제에서 쓰인 코드지만, 재사용하여 과제 진행 속도 향상에 도움을 주었다.

이번 과제에서의 주요 함수로는 linetracing()과 determining\_cross()이다.

while True로 무한 루프를 구현하고, determining\_cross()를 동작하게 하여 linetracing()가 기본으로 실행되도록 한다.

그래서 linetracing()가 먼저 실행되다가 e == 0, a == 0, (c == 1) & (b == 1) & (d == 1) 조건에 부합하게 되면 실행 조건이 바뀌게 된다.

만약 위에 조건이 성립되지 않는다면, linetracing()가 실행되고, 끝나면 while True로 인해 determining\_cross()가 다시 실행된다.

# **향후 추진계획**

## **향후 계획의 세부 내용**

현재, 광센서의 개수 제한과 성능상의 문제로 세밀한 조정이 불가능하여 예상하는 검은 선에서 벗어난다면 최적화된 움직임으로 미로를 탈출하지 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 라즈베리 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** | 2017-11-30 | **차수** | 1 |
| **장 소** | 국민대학교 7호관 446호실 | | |
| **참석자** | 신민욱, 안중민, 안희운 | | |
| **불참자** | --- | | |
| **안 건** | 검은 선 인식 문제에 대하여 | | |
| **회의내용** | <확률적 인식 불가 문제>  지난 4차 과제 때는 이러한 문제가 발견되지 않았음. 하지만, 이번 과제에서는 이상하게 높은 확률로 검은 선을 인식하지 못하여, 라인 복귀하지 못하였음. 그리하여 구동체가 잠시 멈추었다가 다시 가는 형태로 구현하여 인식의 안정성을 높임. 주행속도 감소라는 단점은 향후 수정하면 될듯함.  <인식 속도 이슈>  지난 4차 과제에서는 비교적 느린 인식으로 답답했다면, 이번 과제에서는 인식 속도가 오히려 사람의 눈이 센서 디버깅 코드를 못 따라갈 정도로 속도가 지나치게 빨리짐. 다 같이 코드를 점검해본 결과, 이전 과제보다 loop 코드가 짧아짐에 따라 성능 향상이 인식속도가 빨라진 것이라고 유추해냄.  <인식 오류 문제>  잘 주행하다가 광센서의 인식이 010111과 같이 특정 갈림길 사이에 걸친 상황에서는 구동체가 멈추는 현상이 발견됨. 이는 라인 트레이싱 코드에 케이스를 추가하여 직진하도록 수정하면 될듯함. | | |
| **결과물** | 구동체가 주행을 할 때 다소 속도가 느리지만, 정상적으로 길을 인식하여 주행함.  또한, 갈림길을 만났을 때도 오른손 법칙에 따라서 오류를 발생시키지 않고 정상 주행함을 확인했음. | | |