

저비용 센서기반 신호등 제어 시스템

현재 국내에서 사용되는 신호등 시스템은 단순히 점등시간과 순서만을 고려하여 차량 및 보행자의 대기여부를 고려하지 않은 채 운영되는 시스템이다. 이러한 차량 및 보행자의 대기여부를 고려하여 대기 시간을 감소시키는 다양한 신호등 시스템이 존재하는데, 그 중 가장 보편적으로 사용되는 방법은 미국 등에서 사용되어지고 있는 금속감지센서를 이용한 신호등 시스템이다(Loop-sensor based traffic light control system). 하지만 이는 현실적으로 가격이 비싸고 공사비용이 많이 든다는 문제점을 가지고 있다. 본 연구는 라즈베리파이와 근접센서를 이용하여 보행자와 운전자 모두의 평균 대기 시간을 줄여주는 저비용 센서기반 신호등의 개념 증명(proof of concept)시스템을 구현하였고, 시뮬레이션 결과를 통해 현재 시스템 대비 보행자와 차량에 대해 각각 약 65%, 20%의 평균 대기 시간 절감 효과를 도출하였다. 이 신호등 시스템을 발전시켜 우리나라의 신호등에도 적용되어 보행자와 차량이 보다 편리하게 신호등을 이용할 수 있게 될 것으로 기대한다.

연구자 : 1-2 꽈민경
1-4 박지효

I. 연구 동기 및 목적

1. 연구 동기

횡단보도를 건너려고 할 때, 도로에 차가 있지 않아도 도로의 신호등에서는 여전히 초록색 불이 켜지고, 횡단보도 신호등에 초록색 불이 켜지려면 도로의 신호등에 초록색 불이 꺼질 때까지 기다려야 하는 등, 우리나라의 불편한 신호등 시스템을 바꾸고 싶은 마음이 항상 있었다. 그래서 횡단보도 신호등에 버튼이 있고 도로에는 금속감지 센서가 있어, 도로에 차가 없을 경우엔 다른 차량이 대기 중인 차로의 신호등에 초록색 불이 켜지도록 되어있는 미국의 신호등 시스템을 생각하게 되었고, 이러한 시스템이 우리나라에서도 사용되길 바랐기에, 센서기반 신호등 시스템을 제작하기로 했다.

그러나 미국에 있는 센서기반 신호등 시스템을 제작하려면 비용이 너무 많이 드는 문제점이 있어, 훨씬 저렴하게 제작하기 위한 방법을 생각해내었다. 참고로 미국의 시스템을 구현하려면 그림 1-(a)와 같이 모든 신호등의 바닥에 Loop sensor를 파묻는 공사를 해야 한다. 이는 시공 비용도 비싸고 공사기간이 오래 걸린다는 단점이 있다. 또한, 보행자는 반드시 횡단보도 버튼(그림 1-(b))을 눌러주어야 동작한다는 단점도 있다.



(a) Loop sensor

(b) 횡단보도 버튼

그림 1 Loop sensor와 횡단보도 버튼 사진, (미국 캘리포니아 마운틴뷰)

이에, 라즈베리파이와 적외선 근접 센서를 이용하면 매우 저렴한 저비용 센서기반 신호등 시스템을 제작하여 보행자와 운전자의 평균 대기 시간을 줄일 수 있을 것이라고 생각했다.

2. 연구 목표

라즈베리파이와 센서를 이용하여 센서기반 신호등 개념 증명 시스템을 만들어 낼 수 있음을 증명한다. 이를 위해 신호등 개념 증명 시스템을 라즈베리파이와 센서를 이용하여 모형으로 실제 구현한다.

또한, 보행자와 운전자의 평균 대기 시간을 현재 신호등 시스템보다 줄일 수 있음을 증명한다. 이를 위해 제안하는 시스템을 모델링하는 시뮬레이터를 프로그래밍하여 평균 대기 시간을 측정한다.

II. 이론적 배경

1. 관련 연구

센서기반 신호등 시스템에 대한 기존 연구 자료를 찾아보았다. 대표적으로 다음 두 논문이 있어, 요약해 나타내보았다.

- Design and Development of Sensor Based Traffic Light System(A. Albagul, M. Hrari, Wahyudi, M. F. Hidayathullah)[5]

이 연구의 주요 목적은 기존보다 개선된 알고리즘을 설계하고 구현하는 것이며, 지능형 교통 신호 시뮬레이터에 대한 시뮬레이션을 제공한다. 개발 된 시스템은 교통 신호가 적절하게 반응하도록 설정함으로서 특정 범위 내의 차량의 존재 또는 부재를 감지 할 수 있다. 수학 함수를 사용하여 녹색 신호가 켜지는 적절한 타이밍을 계산함으로서 시스템이 교통 혼잡 문제를 해결하는 데 도움을 줄 수 있다. 구현된 새로운 타이밍 계획은 현재 교통 신호등 시스템의 개선이 가능하며 실현 가능하고 저렴하다.

- Simulation of "Time-based" Versus "sensor-based" Traffic light System (Abdul-Yasser Abd-Fatah, Rosnah Mohd. Yusuff, Faieza Abdul Aziz, Norzima Zulkifli)[6]

이 논문은 사거리에서 시간 및 센서 기반 교통 신호 제어 시스템의 소프트웨어를 이용한 시뮬레이션을 연구했다. 교차로에서 차량의 대기 시간을 줄이기 위해 신호등 시스템에 센서를 도입했다. 먼저 교통 신호등 시스템의 시뮬레이션을 정상적인 교통 상황에 맞게 개발하고 통계 처리 제어를 사용하여 얻은 데이터를 분석한다. 결과적으로 센서 기반 시스템의 평균 대기 시간은 정상적인 교통 상황에서 각 차량에 대한 시간 기반 시스템보다 62.5% 줄었으며 차가 많은 상태에서는 15% 줄었다. 이것은 상당한 개선을 보여준다.

2. 미국 현지 사례 (예: 캘리포니아 주)

미국에서는 센서기반 신호등 시스템을 이미 사용 중이다. 횡단보도에서 사람들이 버튼을 누른 후 기다리고 있을 때, 도로의 신호등에 초록불이 켜졌는데도 지나가는 차가 없을 경우에는 금속 감지센서가 차가 없음을 감지하게 된다. 그리고 횡단보도 신호등에 초록불을 켜주게 된다.

3. 금속감지 센서: 루프 센서

루프 센서는 금속을 감지하는 센서로, 주로 도로에 부착해 도로 위에 차가 있는지 인지하는 데에 사용한다. 그러나 가격이 많이 나가는 문제점이 있다. 가격은 약 50 ~ 60만원이다. 그럼 2-(a), (b)와 같이 설치하여 사용한다.



(b) 루프 센서(loop sensor)

(a) 루프 센서(loop sensor)

그림 2 루프 센서 사진

4. 비교: 한국 신호등 시스템 [4]

모든 신호등이 미리 정해진 시간 단위로 작동된다. 횡단보도에서 기다리는 사람이 있고 도로에 차가 없어도, 도로 신호등이 빨간불로 바뀔 때까지 기다려야 횡단보도 신호등이 초록불로 바뀐다. 이외에도 불편한 점이 여럿 있다. 경우에 따라 경찰이 수동으로 가능하다.

5. 주요 시스템 구성의 배경설명

그림 3은 주요 시스템의 구성을 나타낸 그림이다. 센서들은 라즈베리파이에 연결되어 있고, LED는 라즈베리파이에 의해 켜지는 것을 나타낸다.

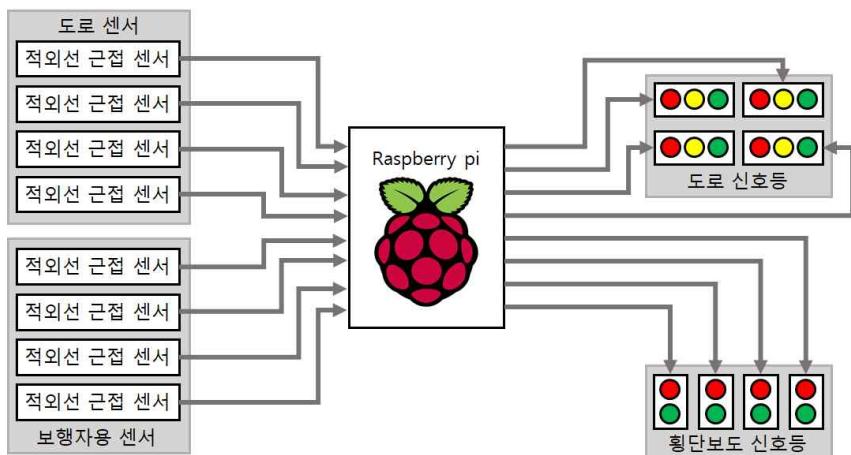


그림 3 주요 시스템 구성

가. 라즈베리파이(Raspberry Pi)

“라즈베리 파이 재단이 학교와 개발도상국에서 기초 컴퓨터 과학의 교육을 증진시키기 위해 개발한 신용카드 크기의 싱글 보드 컴퓨터(그림 4)이다.” [1]



그림 4 라즈베리파이 (Raspberry Pi)

“라즈베리파이를 ‘초소형 컴퓨터’라고 부르는 이유는 컴퓨터의 최소 구성요소와 부품을 가지고 있기 때문이다. 라즈베리파이는 키보드, 모니터 등을 뺀 단일 보드만으로 구성됐다. 다시 말해 컴퓨터 일부 부품인 셈이다. 누군가 보면 미완성으로 그친 제품일 수 있지만, 프로그래머에게는 나만의 컴퓨터를 만들 수 있는 좋은 재료가 된다. 컴퓨터는 이미 보드에 정해진 기능이 있고, 확장할 수 없다. 하지만 라즈베리파이는 사용자가 원하는 대로 기능을 확장하거나 용도를 변경할 수 있다.” [1]

라즈베리파이는 그래픽 성능이 뛰어나면서도 가격이 저렴하다. 현재 라즈베리파이 버전은 1에서 3까지 출시됐으며, 새 버전마다 네트워크 속도나 전력, CPU 기능 등이 개선되고 있다. 기본 모델의 가격은 25-35달러, 한화 3-4만 원으로 기존 컴퓨터에 비해 훨씬 저렴한 수준이다. [1] 본 연구에서 사용한 제품은 라즈베리파이 3 모델 B(\$35)이다.

나. 파이썬(Python) 프로그래밍 언어

“파이썬은 고급 프로그래밍 언어로, 간결한 문법으로 입문자가 이해하기 쉽고 다양한 분야에 활용할 수 있다. 파이썬은 표현 구조가 인간의 사고 체계와 닮아 있고, 외부에 풍부한 라이브러리가 있어 다양한 용도로 확장하기 좋다. 이러한 특징은 유지 보수와 관리도 쉽게 하도록 돋는다. 생산성이 높은 것도 큰 장점이다. 파이썬은 실제로 웹 개발 뿐만 아니라 데이터 분석, 머신러닝, 그래픽, 학술 연구 등 여러 분야에서 활용되고 있다.” [2]

우리는 파이썬을 4개월 정도 배워왔던 덕분에 어느 정도 프로그래밍 할 수 있는 실력을 갖추었다. 라즈베리파이에서 사용되는 프로그래밍 언어가 주로 파이썬이기 때문에 이 연구에서는 라즈베리파이를 이용한다.

III. 연구 방법 및 과정

1. 준비물

센서기반 신호등 개념 증명 시스템 모형 제작에는 라즈베리파이 (그림 5-(a)), 신호등 LED (그림 5-(b)), 근접 센서 (그림 5-(c))가 필요하다. 도로, 신호등 등 모형 제작을 위해 우드락을 사용하였다. 우리가 개념 증명 시스템을 위해 사용한 금액은 총 약 70,000원 정도이며, 표 1에 사용된 금액을 나타내었다. 미국에서 사용하는 센서기반 신호등 시스템 제작보다 훨씬 비용이 적게 든다.

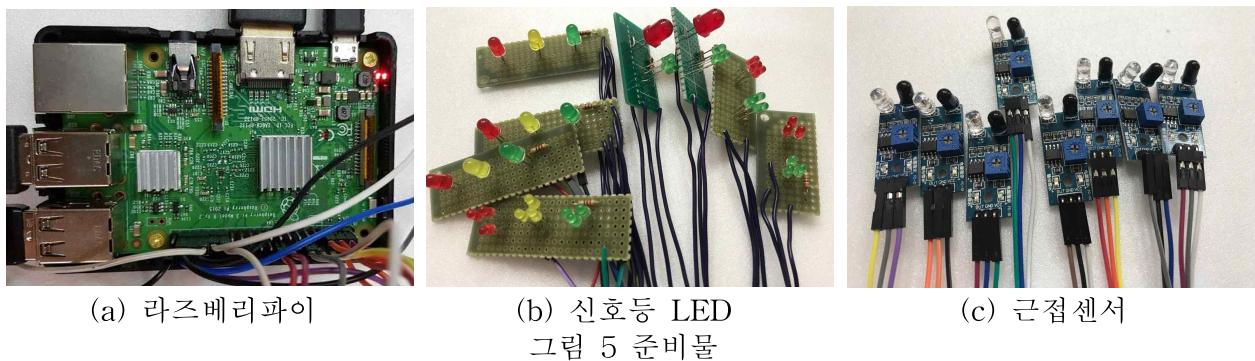


표 1 사용한 금액

품목	가격	수량
라즈베리파이 3 모델 B	40,000원(\$35)	1개
LED	4,000원	38개
센서	20,000원	8개
우드락	6,000원	4개

2. 연구 내용

가. 센서기반 신호등 그림

그림 6은 센서기반 신호등 시스템이 작동되는 예시를 나타내어주는 그림이다. 그림 6과 같이 동서 방향 차로 신호등에 초록불이 켜졌는데도, 지나가는 차가 없어 사람들이 기다리고 있다. 그래서 금속감지센서가 차가 지나다니지 않는 것을 알고 횡단보도 신호등을 켜준다.

나. 시뮬레이션에 필요한 신호등 구성

그림 7의 사거리에서 남북의 길을 NS, 동서의 길을 EW, 횡단보도의 신호등을 Ped(pedestrian : 보행자)라고 하자. NS와 Ped 신호등은 함께 초록불이 켜진다.

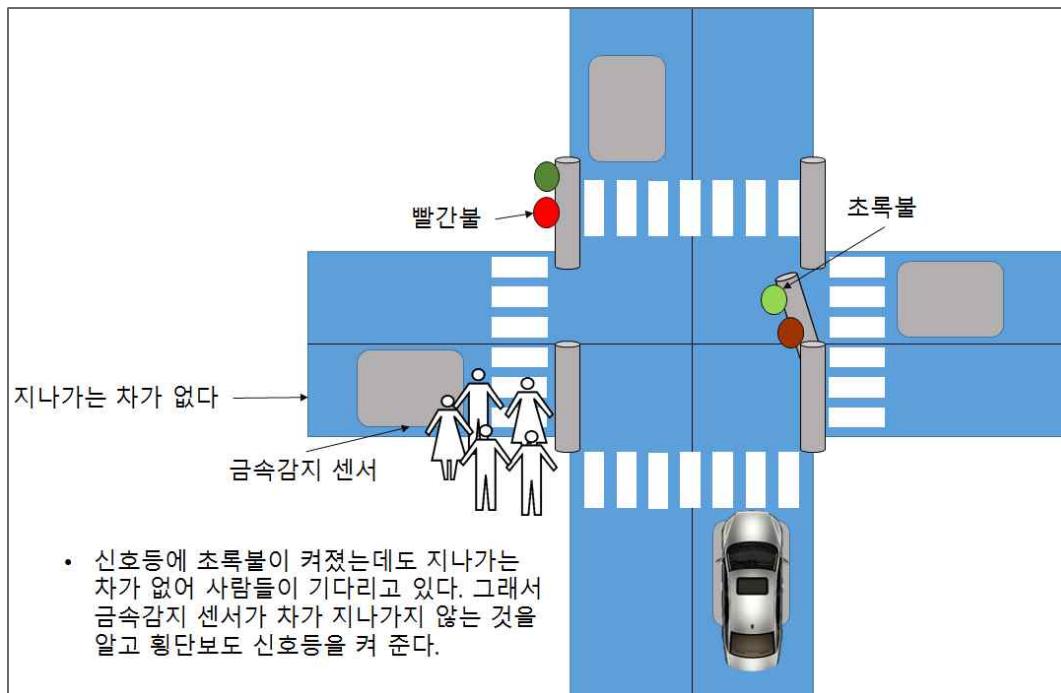


그림 6 센서기반 신호등

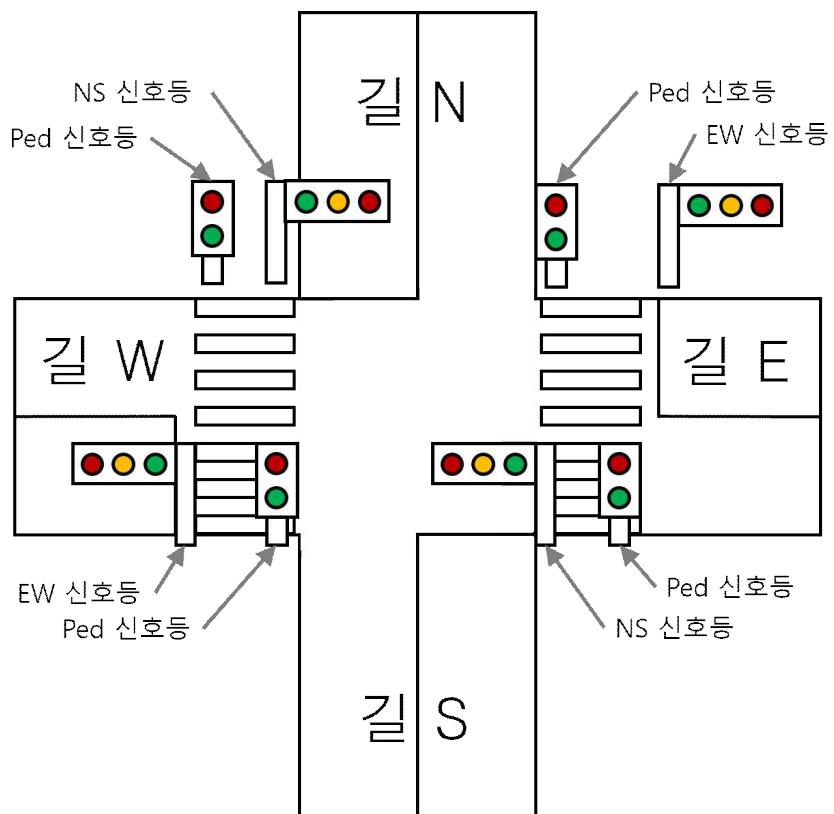


그림 7 시뮬레이션에서 사용된 신호등 구성

다. 시스템 구성도, 순서도

센서기반 신호등 시스템은 그림 8과 그림 9와 같이 작동된다. NS와 EW에 대기하는 차가 없고 Ped에도 대기하는 사람이 없는 상태에서는 기존 신호등 시스템대로 작동된다. 그러나 NS나 EW나 Ped에 차/사람이 하나라도 대기한다면, 원래 초록불이 켜져있던 신호등이 NS와 EW일 때에는 노란불이 켜지게 되고 Ped일 때에는 초록불이 깜박이게 된다. 그리고 중간 과정을 모두 건너뛰고 차/사람이 대기하는 NS나 EW나 Ped에 초록불이 켜지는 순서로 작동된다.

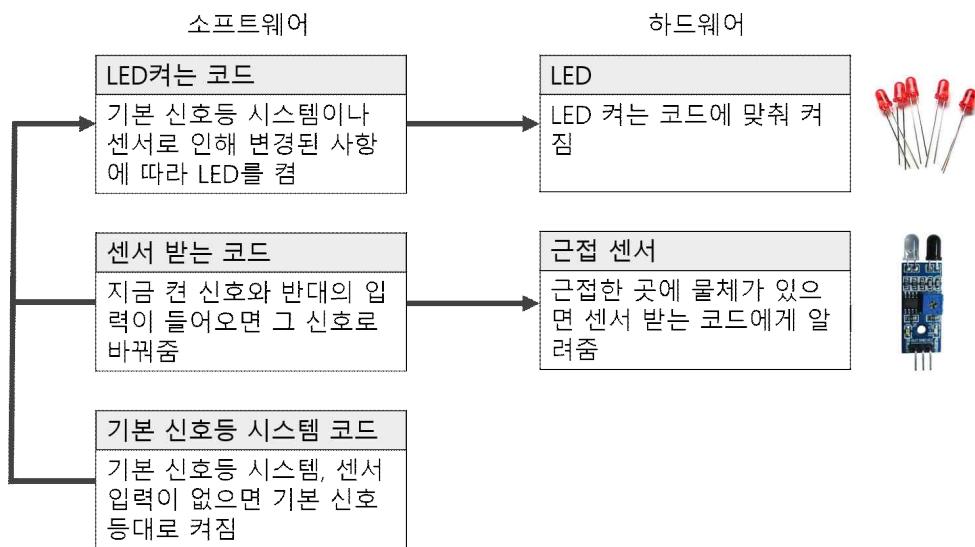


그림 8 센서기반 신호등 시스템 구성도

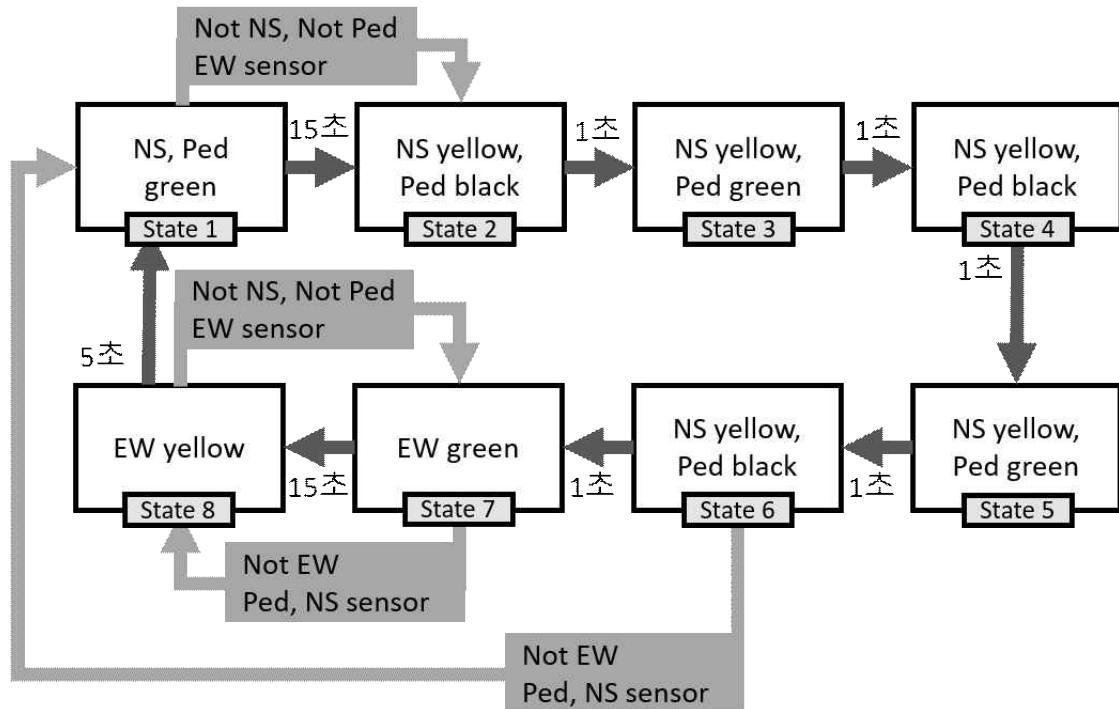
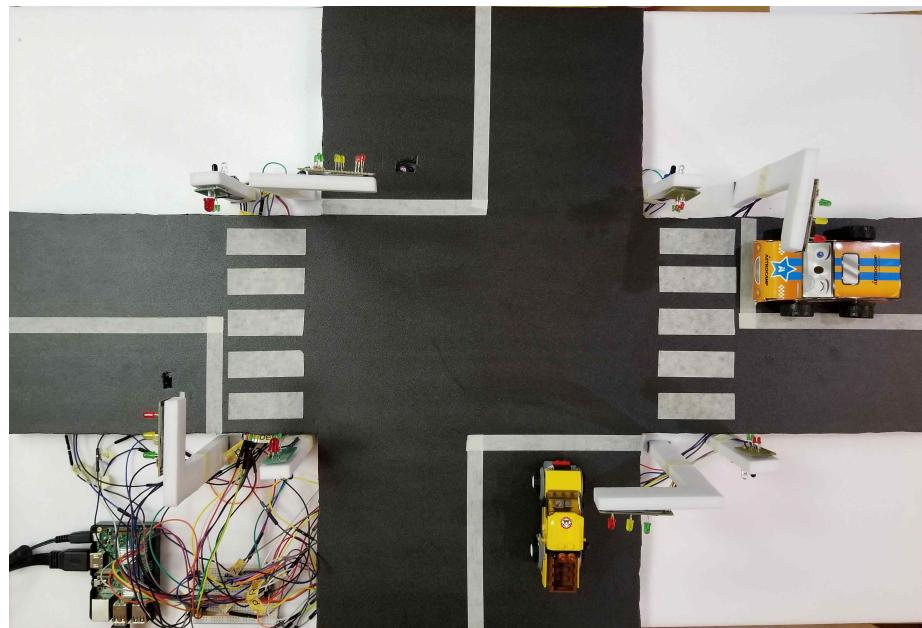


그림 9 센서기반 신호등 시스템 순서도

라. 개념 증명 시스템 모형 제작

그림 10은 실제 제작한 센서기반 신호등 개념 증명 시스템이다. 우드락을 이용하였다. 가로 900mm, 세로 600mm, 높이 180mm 크기로 제작하였다.



(a) 전체 사진



(b) 측면 사진

그림 10 센서기반 신호등 개념 증명 시스템

마. 연구를 위한 가정

사거리의 모든 신호를 사용하여 연구하기에 어려움이 있어 비보호 좌회전을 사용하기로 했다. 단순화를 위해 두 길만 횡단보도가 있다는 가정 하에 연구했다. 다른 두 조건을 한 번에 실험하고 두 결과를 한 번에 도출할 수 있다는 장점이 있다.

바. 기존 신호등 시스템과 센서기반 신호등 시스템 비교

표 2 기존 신호등 시스템과 센서기반 신호등 시스템 비교

기존 신호등 시스템의 동작 순서	센서기반 신호등 시스템의 동작 순서
<ul style="list-style-type: none">정해진 순서대로 불이 켜짐길NS 초록불, 횡단보도 초록불(15초)길NS노란불, 횡단보도 깜빡임(5초)길EW초록불(15초)길EW노란불(5초)	<ul style="list-style-type: none">기존 신호등 시스템 순서대로 가다가 현재 켜진 것과 반대의 센서가 감지되면 그 신호를 켬지나가는 차나 사람이 없으면, 차나 사람이 대기 주인 신호를 켜 줌

사. 기존 계획 대비 변동 사항 및 새롭게 발견한 점

1) 금속감지센서를 적외선 근접센서로 변경

횡단보도에 설치하려던 버튼을 더 편리한 근접 센서로 바꾸기로 했다.

금속 감지센서가 너무 비싼 이유로 그 또한 근접 센서로 교체했다. (실제 도로에 설치한다면 금속 감지 센서를 사용해야 함)

2) 신호등 모형 사이즈 변경

1.5m에 이르는 신호등 모형을 만들려 했으나 시연 등에 어려움이 있어 작은 모형을 만들기로 했다.

3) 응급차량 부분은 향후 계획으로 변경

4) 근접 센서의 한계

미국에서는 금속감지센서를 이용하여 도로에 차가 있는지 알아보지만 센서가 너무 비싸, 금속감지센서 대신 적외선 근접센서를 이용하기로 했다. 그러나 한 문제점이 있었다. 적외선으로 가까운 물체를 인지하는 근접 센서가 계속 반응하고 있는 경우가 있어 고장 난 줄 알았는데 햇빛의 적외선 때문에 가까이에 물체가 있다고 인지되었다. 이를 통해 적외선 근접 센서의 한계를 알게 되었다.

3. 소스코드 구현

가. 개념 증명 시스템을 위한 소스코드: shinho.py

shinho.py는 센서기반 신호등 시스템을 구현한 코드이다. 먼저 기존의 신호등 시스템을 나열하여 코드를 짜고, 거기에 센서를 연결해 센서기반 신호등 시스템을 구현했다. 이 코드는 라즈베리파이 상에서 작동된다. 이 코드에는 함수가 1개가 있는데, 이

함수는 배열에서 주는 값 "R"이나 "Y", "G", "B"에 따라 그에 맞는 색깔의 빨광다이 오드를 켜 준다. 84-89줄에는 배열이 있는데, 각 신호등별 켜지는 색을 state에 따라 보기 쉽게 정리 한 것이다. while문은 무한 반복하도록 하여 무한 반복으로 초를 셀 수 있다. 또한 그 뒤에서는 배열의 순서대로 컴퓨터 모니터 화면에 지금 켜진 색이 나타나도록 해 오류가 없는지 확인 가능하다. while문의 아래 부분에서는 원하는 초 간격에 따라 초를 세서 state를 넘어갈 수 있게 한다. 이 state는 그림 9(순서도)에도 나타나 있다. state 8 까지 모두 다 돌아가면, state는 1, 초를 세는 변수인 a는 0으로 만들어 다시 돌아가게 한다. 여기에 센서를 반영하여 모든 센서에 변수를 부여하였다. 동일한 반응을 보여야 하는 센서 중 하나라도 감지되면 NS나 EW가 1이 된다. 이 값으로 어떤 신호를 켜야 하는지 알려준다. 이는 그림 9의 옆은 회색 화살표와 같다. shinho.py는 센서기반 신호등 시스템 모형에서 사용되는 코드로, 신호등과 근접센서가 작동되게 한다. 주로 배열과 if, elif문을 사용하여 코드를 구현했다.

나. 시뮬레이션을 위한 소스코드: sim.py 와 sim_sensor.py

각 시스템의 평균 대기 시간을 계산하기 위한 시뮬레이션에 필요한 코드이다. 그림 12-(a)는 일반 신호등 시스템일 경우의 평균 대기 시간을 계산하는 시뮬레이션을 구현하였고, 그림 12-(b)는 센서기반 신호등 시스템일 경우의 평균 대기 시간을 계산하는 시뮬레이션을 구현했다. 두 코드에서는 평균 대기 시간을 계산하기 위해 차/사람의 총 수, 차/사람의 총 대기 시간도 구한다. 주로 변수와 if문을 이용해 코드를 구현하였다. 그리고 Poisson(포아송) 분포를 사용하여 평균 도착 간격동안 몇 대의 차/사람이 올지 구현했다.

그림 12-(a)와 그림 12-(b) 모두 가장 먼저 포아송 분포를 사용했다. 그 후 NS, EW, Ped에 일정시간동안 차/사람이 얼마나 오는지의 변수(q_ns, q_ew, q_ped)를 선언했고, 10시간동안 평균 도착 간격마다 차/사람이 하나씩 오는 코드를 구현했다. 그리고 차/사람의 총 수 변수(ns_car, ew_car, ped)와 대기하는 총 시간의 변수(w_ns, w_ew, w_ped)를 선언한 다음, while문을 사용해 시간이 1초마다 하나씩 늘어나도록 했다. 15초가 되기 전까지는 state가 1이고, 15초에는 state 2가 된다. 20초에는 state 3이 되고, 35초에는 state 4가 된다. 그리고 40초가 되면 state는 다시 1이 된다. 여기서 state 1은 NS, Ped가 초록불일 때이고, state 2는 NS, Ped가 노란불일 때를 말한다. state 3은 EW가 초록불일 때이고, state 4는 EW가 노란불일 때를 말한다. 여기에 추가로 그림 12-(b)에서는 차/사람이 있는지 없는지를 구하는 변수(ox_ns, ox_ew, ox_ped)를 선언한다. 차가 없을 때에는 변수의 값이 0이고, 차가 있을 때에는 값이 1이다. 만약 ox_ns와 ox_ped 변수가 1이고, ox_ew 변수는 0이라면, state가 4로 이동해, 먼저 노란불이 깜박인 후 NS, Ped에 초록색불이 켜지게 된다. 마지막으로 두 코드에 모두 총 대기 시간을 총 수로 나눠 차/사람의 평균 대기 시간을 계산하는 코드를 구현했다.

shinho.py

```

import time
import RPi.GPIO as GPIO
import time
import threading
from gpiozero import MotionSensor

def light(b):
    #Fuction to turn on the light according to the array
    GPIO.output(23, True)
    GPIO.output(24, True)
    GPIO.output(2, True)
    GPIO.output(3, True)
    GPIO.output(4, True)
    GPIO.output(14, True)
    GPIO.output(15, True)
    GPIO.output(18, True)
    if shinN[b] == 'G':
        GPIO.output(2, False)
    if shinN[b] == 'Y':
        GPIO.output(3, False)
    if shinN[b] == 'R':
        GPIO.output(4, False)
    if shinS[b] == 'G':
        GPIO.output(2, False)
    if shinS[b] == 'Y':
        GPIO.output(3, False)
    if shinS[b] == 'R':
        GPIO.output(4, False)

    if shinW[b] == 'G':
        GPIO.output(14, False)
    if shinW[b] == 'Y':
        GPIO.output(15, False)
    if shinW[b] == 'R':
        GPIO.output(18, False)
    if shinE[b] == 'G':
        GPIO.output(14, False)
    if shinE[b] == 'Y':
        GPIO.output(15, False)
    if shinE[b] == 'R':
        GPIO.output(18, False)

    if hshinW[b] == 'G':
        GPIO.output(23, False)
    if hshinW[b] == 'R':
        GPIO.output(24, False)
    if hshinE[b] == 'G':
        GPIO.output(23, False)
    if hshinE[b] == 'R':
        GPIO.output(24, False)
    GPIO.cleanup()
    GPIO.setmode(GPIO.BCM)

    GPIO.setup(23, GPIO.OUT)#ped light
    GPIO.setup(24, GPIO.OUT)
    GPIO.setup(2, GPIO.OUT)#trafic light SN
    GPIO.setup(3, GPIO.OUT)
    GPIO.setup(4, GPIO.OUT)
    GPIO.setup(14, GPIO.OUT)#trafic light EW
    GPIO.setup(15, GPIO.OUT)
    GPIO.setup(18, GPIO.OUT)

    sensorn = MotionSensor(16)
    sensors = MotionSensor(20)
    sensore = MotionSensor(21)
    sensorw = MotionSensor(26)
    buttonn = MotionSensor(5)
    buttons = MotionSensor(6)
    buttone = MotionSensor(13)
    buttonw = MotionSensor(19)
    GPIO.output(23, True)
    GPIO.output(24, True)
    GPIO.output(2, True)
    GPIO.output(3, True)
    GPIO.output(4, True)
    GPIO.output(14, True)
    GPIO.output(15, True)
    GPIO.output(18, True)

    i = 1
    ped1 = 0
    ped2 = 0
    ped3 = 0
    ped4 = 0

    shinN = ['G', 'Y', 'Y', 'Y', 'Y', 'R', 'R']#g=green,
    y=yellow, r=red, b=black
    shinS = ['G', 'Y', 'Y', 'Y', 'Y', 'R', 'R']
    shinE = ['R', 'R', 'R', 'R', 'R', 'G', 'Y']
    shinW = ['R', 'R', 'R', 'R', 'R', 'G', 'Y']
    hshinW = ['G', 'B', 'G', 'B', 'G', 'B', 'R', 'R']
    hshinE = ['G', 'B', 'G', 'B', 'G', 'B', 'R', 'R']
    state = 1
    a = 1
    while True:
        time.sleep(1)
        EW = 0
        NS = 0
        nsen = sensorn.motion_detected
        ssen = sensors.motion_detected
        esen = sensore.motion_detected
        wsen = sensorw.motion_detected
        ped1 = buttonn.motion_detected
        ped2 = buttone.motion_detected
        ped3 = buttonw.motion_detected
        ped4 = buttons.motion_detected
        if not nsen or not ssen or not ped1 or not ped2 or
        not ped3 or not ped4:
            NS = 1
            if not esen or not wsen:
                EW = 1
                b = state - 1
                print i, " ", shinN[b], " ", shinS[b], " ",
                shinE[b], " ", shinW[b], " ", hshinW[b], " ",
                hshinE[b], " ", state
                print
                light(b)
                i = i + 1
                if NS == 1 and EW == 0 and state == 6:
                    state = 1
                    a = 0
                if NS == 1 and EW == 0 and state == 7:
                    state = state + 1
                    a = 0
                if EW == 1 and NS == 0 and state == 8:
                    state = 7
                    a = 0
                if EW == 1 and NS == 0 and state == 1:
                    state = state + 1
                    a = 0
                if a == 15:
                    state = state + 1
                    a = 0
                elif a == 1 and state == 2:
                    state = state + 1
                    a = 0
                elif a == 1 and state == 3:
                    state = state + 1
                    a = 0
                elif a == 1 and state == 4:
                    state = state + 1
                    a = 0
                elif a == 1 and state == 5:
                    state = state + 1
                    a = 0
                elif a == 1 and state == 6:
                    state = state + 1
                    a = 0
                elif a == 5 and state == 8:
                    state = 1
                    a = 0
                    a = a + 1

```

그림 11 shinho.py 코드

sim.py

```

import random
import math

def nextTime(rateParameter): # functions that cause
    random values
    return -math.log(1.0 - random.random()) /
rateParameter # Poisson distribution
random.seed(21) # [7]

q_ns = [] # how many cars are on the ns road for a
certain period of time
q_ew = [] # how many cars are on the ew road for a
certain period of time
q_ped = [] # how many people are on the crosswalk
for a certain period of time
for i in range(36000): # 10 hours simulation
    q_ns.append(0)
    q_ew.append(0)
    q_ped.append(0)
t1 = 0 # t = time
t2 = 0
t3 = 0
while t1 < 36000:
    q_ns[int(t1)] = 1
    t1 = t1 + math.ceil(nextTime(1/10.0)) # increased
by 10 seconds
while t2 < 36000:
    q_ew[int(t2)] = 1
    t2 = t2 + math.ceil(nextTime(1/10.0))
while t3 < 36000:
    q_ped[int(t3)] = 1
    t3 = t3 + math.ceil(nextTime(1/10.0))

state = 1
t = 0
i = 0
ns_car = 0 # total cars on ns road
ew_car = 0 # total cars on ew road
ped = 0 # total people on crosswalk
w_ns = 0 # total time cars wait on ns road
w_ew = 0 # total time cars wait on ew road
w_ped = 0 # total time people wait on crosswalk
while t < 36000:
    print "time:", t
    if i < 15:
        state = 1
        print state
    if i >= 15 and i < 20:
        state = 2
        print state
    if i >= 20 and i < 35:
        state = 3
        print state
    if i >= 35 and i < 40:
        state = 4
        print state
    if i == 40:
        state = 1
        print state
        i = 0
    if state == 1: #ns, ped
        ns_car = ns_car + q_ns[t]
        ew_car = ew_car + q_ew[t]
        if q_ew[t] == 1:
            w_ew = w_ew + 20 - i
    if state == 2: #yellow
        ns_car = ns_car + q_ns[t]
        ew_car = ew_car + q_ew[t]
        if q_ns[t] == 1:
            w_ns = w_ns + 40 - i
        if q_ew[t] == 1:
            w_ew = w_ew + 20 - i
    if state == 3: #ew
        ns_car = ns_car + q_ns[t]
        ew_car = ew_car + q_ew[t]
        ped = ped + q_ped[t]
        if q_ns[t] == 1:
            w_ns = w_ns + 40 - i
        if q_ped[t] == 1:
            w_ped = w_ped + 40 - i
    if state == 4: #yellow
        ns_car = ns_car + q_ns[t]
        ew_car = ew_car + q_ew[t]
        ped = ped + q_ped[t]
        if q_ns[t] == 1:
            w_ns = w_ns + 40 - i
        if q_ew[t] == 1:
            w_ew = w_ew + 60 - i
        if q_ped[t] == 1:
            w_ped = w_ped + 40 - i
    if state == 1 or state == 2:
        ped = ped + q_ped[t]
    i = i + 1
    t = t + 1
print "ns - cars:", ns_car
print "ew - cars:", ew_car
print "ped - people:", ped
print "ns - cars waiting signal, average time:", w_ns, float(w_ns)/ns_car # average waiting time on
ns road
print "ew - cars waiting signal, average time:", w_ew, float(w_ew)/ew_car # average waiting time on
ew road
print "ped - people waiting signal, average time:", w_ped, float(w_ped)/ped # average waiting time on
crosswalk

```

(a) sim.py 코드

sim_sensor.py

```

import random
import math

def nextTime(rateParameter): # functions that cause
    random values
    return -math.log(1.0 - random.random()) /
rateParameter # Poisson distribution
random.seed(21) # [7]

q_ns = [] # how many cars are on the ns road for a
certain period of time
q_ew = [] # how many cars are on the ew road for a
certain period of time
q_ped = [] # how many people are on the crosswalk
for a certain period of time
for i in range(36000): # 10 hours simulation
    q_ns.append(0)
    q_ew.append(0)
    q_ped.append(0)
t1 = 0 # t = time
t2 = 0
t3 = 0
while t1 < 36000:
    q_ns[int(t1)] = 1
    t1 = t1 + math.ceil(nextTime(1/10.0)) # increased by
10 seconds
while t2 < 36000:
    q_ew[int(t2)] = 1
    t2 = t2 + math.ceil(nextTime(1/10.0))
while t3 < 36000:
    q_ped[int(t3)] = 1
    t3 = t3 + math.ceil(nextTime(1/10.0))

state = 1
t = 0
i = 0
ns_car = 0 # total cars on ns road
ew_car = 0 # total cars on ew road
ped = 0 # total people on crosswalk
w_ns = 0 # total time cars wait on ns road
w_ew = 0 # total time cars wait on ew road
w_ped = 0 # total time people wait on crosswalk
ox_ns = 0 # whether there is car on ns road
ox_ew = 0 # whether there is car on ew road
ox_ped = 0 # whether there is car on crosswalk
while t < 36000:
    print "time:", t
    if q_ns[t] == 1:
        ox_ns = 1
    if q_ew[t] == 1:
        ox_ew = 1
    if q_ped[t] == 1:
        ox_ped = 1
    if i < 15:
        state = 1
        ox_ns = 0
        ox_ped = 0
        print state
        if q_ns[t] == 0 and ox_ew == 1 and q_ped[t] ==
0: # if there is no car on ns road, there is a car on
ew road, there is no car on crosswalk
            i = 15
    if i >= 15 and i < 20:
        state = 2
        print state

```

```

if i >= 20 and i < 35:
    state = 3
    ox_ew = 0
    print state
    if ox_ns == 1 and q_ew[t] == 0 and ox_ped == 0:
# if there is a car on ns road, there is no car on ew
road, there is no car on crosswalk
        i = 35
    if ox_ns == 1 and q_ew[t] == 0 and ox_ped == 1:
# if there is a car on ns road, there is no car on ew
road, there is a car on crosswalk
        i = 35
    if ox_ns == 0 and q_ew[t] == 0 and ox_ped == 1:
# if there is no car on ns road, there is no car on ew
road, there is a car on crosswalk
        i = 35
    if i >= 35 and i < 40:
        state = 4
        print state
    if i == 40:
        state = 1
        print state
        i = 0
    if state == 1: # ns, ped
        ns_car = ns_car + q_ns[t]
        ew_car = ew_car + q_ew[t]
        if q_ew[t] == 1:
            w_ew = w_ew + 20 - i
    if state == 2: # yellow
        ns_car = ns_car + q_ns[t]
        ew_car = ew_car + q_ew[t]
        if q_ns[t] == 1:
            w_ns = w_ns + 40 - i
        if q_ew[t] == 1:
            w_ew = w_ew + 20 - i
    if state == 3: # ew
        ns_car = ns_car + q_ns[t]
        ew_car = ew_car + q_ew[t]
        ped = ped + q_ped[t]
        if q_ns[t] == 1:
            w_ns = w_ns + 40 - i
        if q_ped[t] == 1:
            w_ped = w_ped + 40 - i
    if state == 4: # yellow
        ns_car = ns_car + q_ns[t]
        ew_car = ew_car + q_ew[t]
        ped = ped + q_ped[t]
        if q_ns[t] == 1:
            w_ns = w_ns + 40 - i
        if q_ew[t] == 1:
            w_ew = w_ew + 60 - i
        if q_ped[t] == 1:
            w_ped = w_ped + 40 - i
    if state == 1 or state == 2:
        ped = ped + q_ped[t]
        i = i + 1
        t = t + 1
print "ns - cars:", ns_car
print "ew - cars:", ew_car
print "ped - people:", ped
print "ns - cars waiting signal, average time:", w_ns,
float(w_ns)/ns_car # average waiting time on ns road
print "ew - cars waiting signal, average time:", w_ew,
float(w_ew)/ew_car # average waiting time on ew road
print "ped - people waiting signal, average time:",
w_ped, float(w_ped)/ped # average waiting time on
crosswalk

```

(b) sim_sensor.py 코드

그림 12 sim.py, sim_sensor.py 코드

4. 연구 진행 과정

그림 13과 그림 14는 센서기반 신호등 시스템 모형을 제작할 때 찍은 사진이다. 그림 13-(a)는 신호등에 LED를 연결하고, 각 전선이 어떤 것과 연결되었는지 알 수 있도록 스티커를 부착하는 모습이다. 그림 13-(b)는 LED를 연결한 신호등에 직접 납땜 작업을 하는 모습이다. 그리고 그림 14-(a)는 여러 신호등 전선과 근접 센서 전선을 라즈베리파이에 연결하는 모습이다. 그림 14-(b)는 최종적으로 완성된 센서기반 신호등 시스템 모형을 들고 지도교수님과 함께 찍은 사진이다.

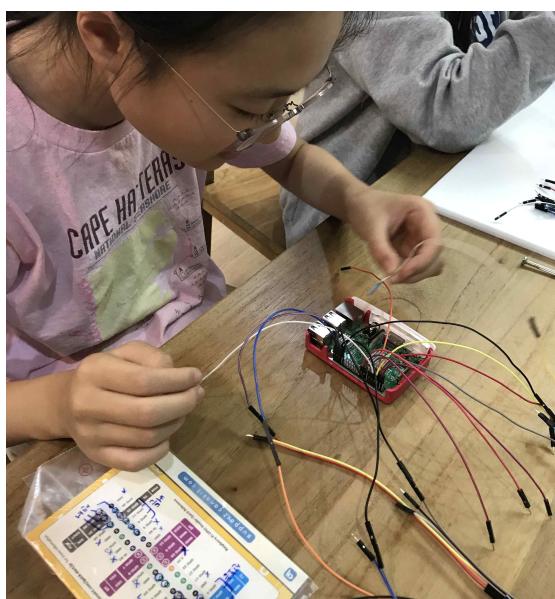


(a) LED 연결

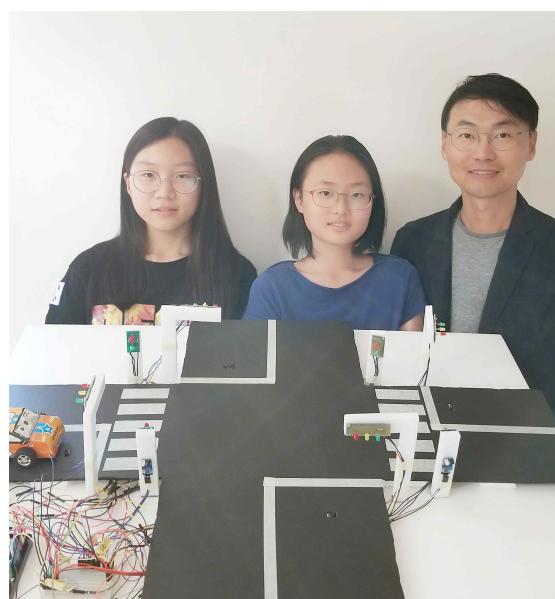


(b) 납땜 작업

그림 13 연구 진행 과정



(a) 라즈베리파이에 전선들을 연결



(b) 최종 모형

그림 14 연구 진행 과정

IV. 연구 결과

1. 제안하는 시스템의 대기 시간 변동 그래프

이 시뮬레이션은 sim.py와 sim_sensor.py 코드 실행을 통해 결과가 나왔고, 얻은 결과를 꺾은선 그래프로 나타내었다. sim.py와 sim_sensor.py에서는 10시간동안 평균 도착 간격(10초, 20초, 30초, 40초, 50초, 60초)마다 차량이나 보행자가 하나씩 올 것으로 가정하여 구현했고, Poisson(포아송) 분포를 이용하여 무작위로 차량이나 보행자를 생성하였다. 그래서 10초마다 차량이나 보행자가 하나씩 올 때, NS, EW, Ped의 총 수는 약 3,600명/대 정도이다. 여기서 Poisson(포아송) 분포란, “단위 시간 안에 어떤 사건이 몇 번 발생할 것인지를 표현하는 이산 확률 분포이다.” [3]

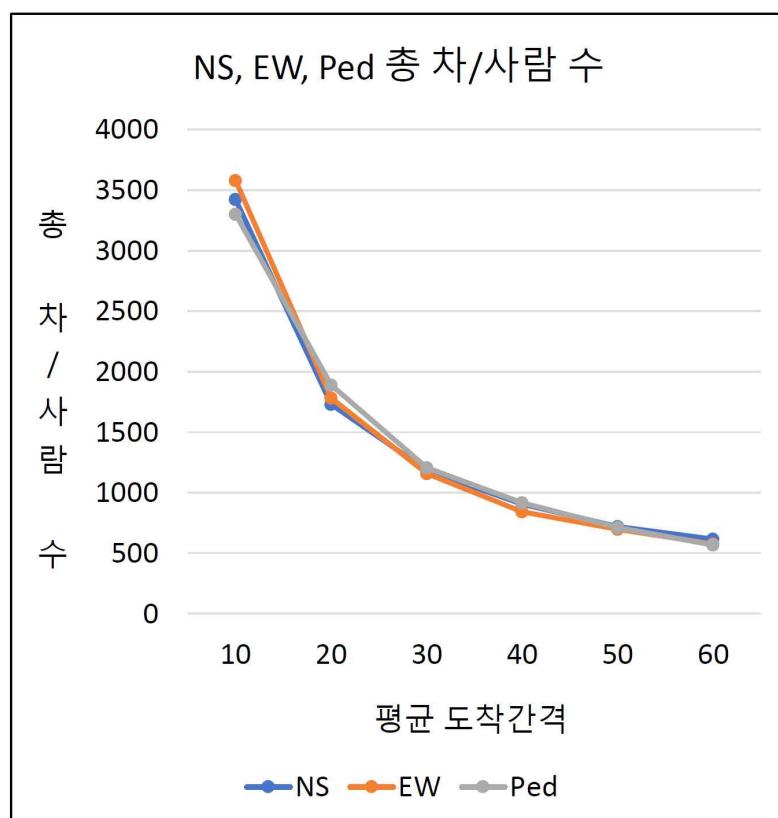
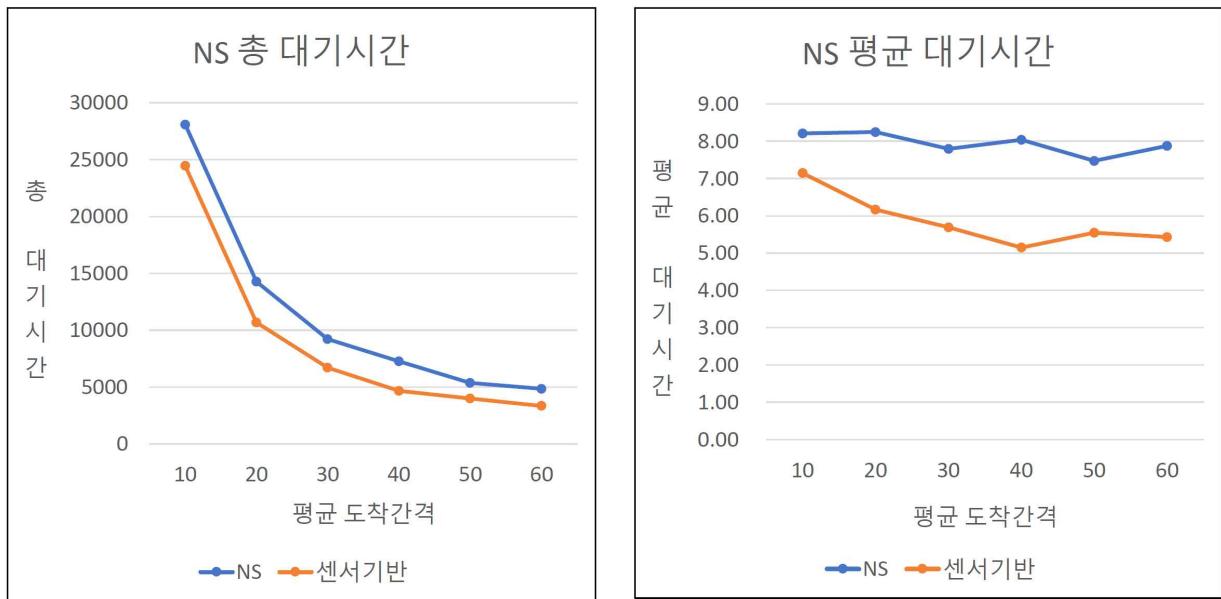


그림 15 NS, EW, Ped 총 차/사람 수

그림 15는 평균 도착 간격(10초, 20초, 30초, 40초, 50초, 60초)동안 차가 한 대씩 올 때, NS 도로와 EW 도로, 횡단보도에서 기다리는 차량과 사람들의 수를 나타낸 그래프이다. 이는 일반 신호등 시스템일 경우와 센서기반 신호등 시스템일 경우에 나타나는 결과가 같기 때문에 그래프를 한 개로 나타내었다. NS 도로, EW 도로, 횡단보도의 총 수가 서로 거의 비슷하다.

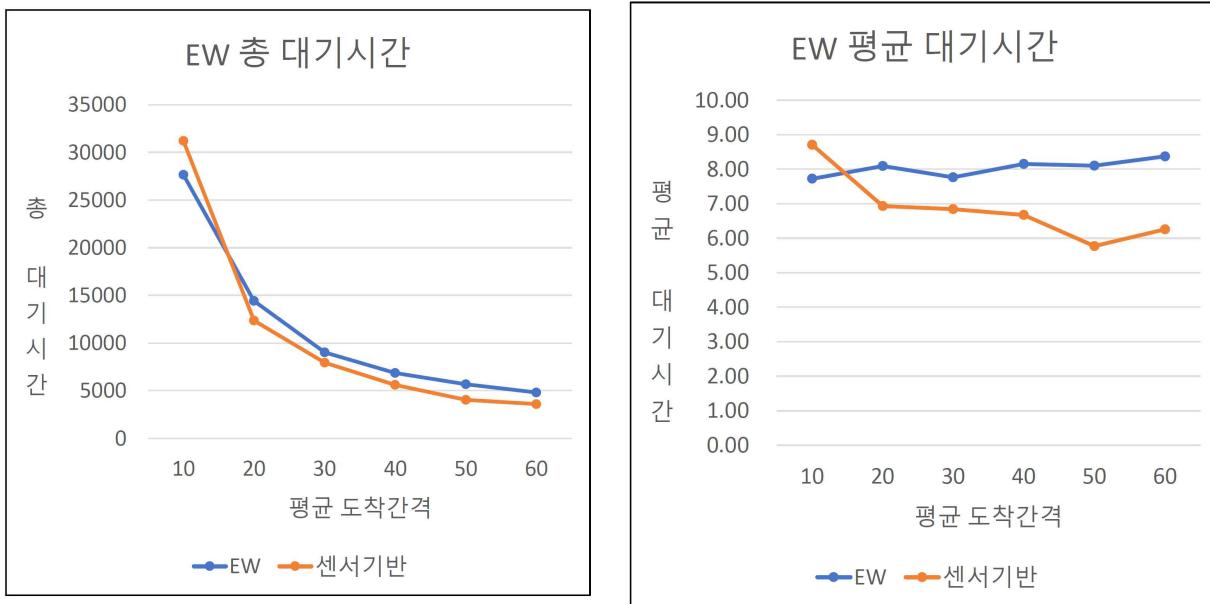


(a) NS 도로 총 대기 시간

(b) NS 도로 평균 대기 시간

그림 16 NS 도로 총 대기 시간과 평균 대기 시간

그림 16-(a)는 NS 도로에서 일정시간동안 한 대씩 온 차들이 지나가기 위해 신호등이 켜지길 기다릴 때, 일반 신호등 시스템일 경우와 센서기반 신호등 시스템일 경우에 차가 총 대기하는 시간을 나타내었다. 이는 일반 신호등 시스템일 경우와 센서기반 신호등 시스템일 경우에 나타나는 결과가 다르기 때문에 그래프 안에서 값을 두 개로 나타내었다. 그림 16-(b)는 차들이 NS 도로에서 신호등이 켜지길 기다리는 평균 대기 시간을 일반 신호등 시스템일 경우와 센서기반 신호등 시스템일 경우로 나타낸 것이다. 그림 16-(b)는 그림 16-(a)의 값을 NS 도로에 온 차들의 수로 나눠 평균 대기 시간을 구한 것으로, 센서기반 신호등 시스템일 경우의 평균 대기 시간이 일반 신호등 시스템일 경우보다 훨씬 짧은 것을 볼 수 있다. 이는 일반 신호등 시스템일 경우와 센서기반 신호등 시스템일 경우에 나타나는 결과가 다르기 때문에 그래프 안에서 값을 두 개로 나타내었다.

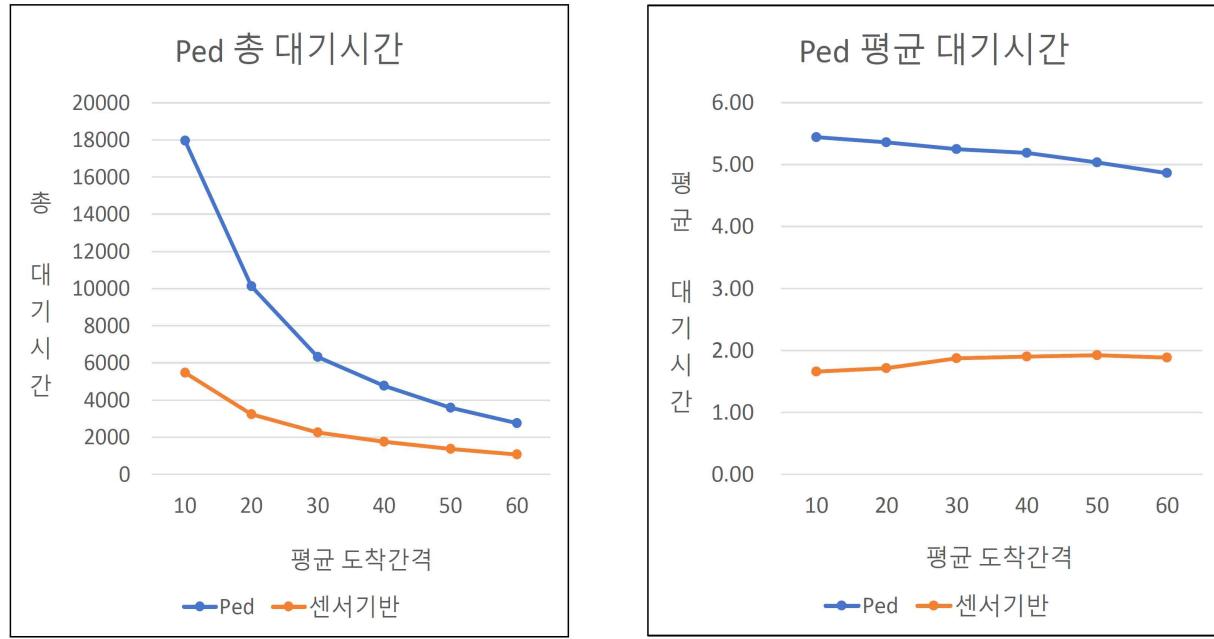


(a) EW 도로 총 대기 시간

(b) EW 도로 평균 대기 시간

그림 17 EW 도로 총 대기 시간과 평균 대기 시간

그림 17-(a)는 EW 도로에서 일정시간동안 한 대씩 온 차들이 신호등 켜지길 기다릴 때, 차량이 총 대기하는 시간을 나타낸 그래프이다. 그림 17-(b)는 그림 17-(a)의 값을 NS 도로에 온 차들의 수로 나눠 평균 대기 시간을 구해 나타낸 그래프이다. 그림 17-(a), 그림 17-(b)의 그 외 설명은 NS와 동일하다. 그러나 그림 17-(a)와 그림 17-(b)에서 모두 도착 간격이 10초일 때에는 이상하게 센서기반 신호등 시스템일 경우의 대기 시간이 일반 신호등 시스템일 경우의 대기 시간보다 더 길다. 그 이유는 센서기반 신호등 시스템일 경우, NS 도로와 횡단보도는 같은 방향에 있어 신호등이 같이 켜지게 되고 NS 도로와 횡단보도에도 10초가 될 때마다 대기하는 차/사람 수가 하나씩 늘어나게 된다. 그러면 NS 도로와 횡단보도에서 대기하는 차/사람을 위해 EW보다 신호등이 더 많이 켜지게 되어, 대기 시간이 일반 신호등 시스템일 경우보다 약간 길어진다. 그래도 센서기반 신호등 시스템일 경우의 평균 대기 시간이 일반 신호등 시스템일 경우보다 훨씬 짧은 것을 볼 수 있다.



(a) Ped 횡단보도 총 대기 시간

(b) Ped 횡단보도 평균 대기 시간

그림 18 횡단보도 총 대기 시간과 평균 대기 시간

그림 18-(a)는 횡단보도에서 일정시간동안 한 명씩 온 사람들이 신호등 켜지길 기다릴 때, 사람들이 총 대기하는 시간을 나타낸 그래프이다. 그림 18-(b)는 그림 18-(a)의 값을 횡단보도에 온 사람들의 수로 나눠 평균 대기 시간을 구해 나타낸 그래프이다. 그림 18-(a), 그림 18-(b)의 그 외 설명은 NS, EW와 동일하다. 그림 18-(a)와 그림 18-(b)에서는 NS와 EW에 비해, 센서기반 신호등 시스템일 경우의 평균 대기 시간과 일반 신호등 시스템일 경우의 평균 대기 시간의 차이가 훨씬 큰 것을 볼 수 있다.

표 3 보행자와 운전자의 평균 대기 시간 비교

	일반 신호등	센서기반 신호등	개선 폭
보행자	5.19초	1.83초	약 65% 감소
운전자	7.98초	6.36초	약 20% 감소

표 3은 보행자와 운전자의 평균 대기 시간을 일반 신호등 시스템과 센서기반 신호등 시스템으로 나누어 나타낸 표이다. 센서기반 신호등 시스템이 사용될 때, 일반 신호등 시스템에 비해 평균 대기 시간이 몇 %의 시간 감소 효과가 나타났는지 표로 제작했다. 그 결과, 센서기반 신호등 시스템일 경우에 일반 신호등 시스템일 경우보다 보행자는 평균 대기 시간이 약 65% 감소하였고, 운전자는 평균 대기 시간이 약 20% 감소하는 큰 개선 폭을 볼 수 있었다.

V. 결론 및 소감

1. 결론

NS 도로, EW 도로, 횡단보도의 총 대기 시간과 평균 대기 시간에 대한 그래프를 만들었을 때 결과를 더 확실하게 볼 수 있었는데, 모두 센서기반 신호등 시스템일 경우의 평균 대기 시간이 일반 신호등 시스템일 경우의 평균 대기 시간보다 훨씬 짧았다. 또한 결과를 퍼센트로 나타내었을 때, 센서기반 시스템일 경우에 일반 신호등 시스템일 경우보다 보행자는 약 65%의 대기 시간이 줄어들었고, 운전자는 약 20%의 대기 시간이 줄어드는 큰 개선 폭을 볼 수 있었다. 이를 통해 일반 신호등 시스템일 경우와 센서기반 신호등 시스템일 경우의 차이를 크게 볼 수 있다.

이렇게 평균 대기 시간을 줄일 수 있는 센서기반 신호등 시스템을 고비용의 기존 Loop sensor가 아닌 저비용의 적외선 근접센서로 실현 가능함을 라즈베리파이를 이용한 모형으로 제작하는 목표가 달성되었다.

2. 소감

센서기반 신호등 시스템이 우리나라에서도 사용되길 바라는 마음으로, 센서기반 신호등 시스템을 시뮬레이션해보고 직접 모형도 제작해보는 과정에서 많은 어려움이 있었는데, 그중에서도 많은 양의 코드를 구현하는 점이 가장 힘들었던 것 같다. 그리고 매번 늦은 시간까지 같이 모형을 제작하고 논문을 쓰는 점 또한 어려웠다. 그러나 논문을 처음 써보면서, 논문을 쓰는 방법과 그동안 잘 알지 못했던 우리나라와 미국의 신호등 체계에 대해 자세히 알게 되었다. 힘들고 어려운 점도 있었지만, 이 연구를 통해 많은 것을 배우고 모형도 제작해보는 것이 정말 좋은 기회였다고 생각된다.

◆ 참고 문헌

[1] 위키 백과 (라즈베리파이)

https://ko.wikipedia.org/wiki/라즈베리_파이

[2] 위키 백과 (파이썬)

<https://ko.wikipedia.org/wiki/파이썬>

[3] 위키 백과 (Poisson 분포)

https://ko.wikipedia.org/wiki/%EB%8B%A8%EC%9A%9C_%EB%8E%A5%ED%95%91

[4] 한국 신호등 체계

https://ko.wikipedia.org/wiki/%ED%85%84%ED%8A%B8%EC%9D%BC%EC%84%9C%EC%84%9C%ED%8A%8C_%ED%8A%9C%ED%8A%9C%ED%8A%9C

[5] A. Albagul, M. Hrari, Wahyudi and M. F. Hidayathullah, Design and Development of Sensor Based Traffic Light System, American Journal of Applied Sciences 3(3): 1745–1749, 2006.

[6] Abdul-Yasser Abd-Fatah, Rosnah Mohd. Yusuff, Faieza Abdul Aziz , and Norzima Zulkifli, Simulation of “Time-based” Versus “Sensor-based” Traffic light System: 252042463, 2011.

[7] Poisson 분포 코드

<http://preshing.com/20111007/how-to-generate-random-timings-for-a-poisson-process/>

[8] 코드 구현할 때, 참고한 Python 책

Zed A. Shaw, Learn Python the Hard Way, Addison-Wesley Professional, 2013.