
팀 프로젝트 결과보고서

우회전 신호등 임베디드 시스템

RTL(Right-turn Traffic Light)

팀 명 : A S P

조 원 : 201520883 이혜훈

201823784 신민선

201823782 안지영

제출일 : 2021. 06. 17

분 야 : 스마트 교통

목 차

I. 시스템 개요

- 개발 목표 3

II. 시스템 설계

- 시스템 분석 및 세부설계 3
- 설계 변경사항 5

III. 시스템 구현

- 라즈베리 파이별 구현 내역 및 세부 알고리즘 7

IV. 시험평가

- 개발환경 12
- 테스트결과 12

V. 개발이슈

- 동기화 문제 14
- 제어권 반납 문제 15

VI. 기대효과 및 후속연구 15

VII. 수행경과 16

I. 시스템 개요

■ 개발 목표

본 프로젝트는 우회전 차량, 횡단보도 보행자의 유무에 따라 안내 신호를 출력하는 우회전 신호등 임베디드 시스템에 관한 것이다. 앞선 자료분석에서, 교차로에서 우회전 교통사고 발생 빈도가 높다는 것을 살펴보았다. 또한, 사고의 원인이 운전자의 시야와 밀접한 관련이 있음을 파악할 수 있었다. 따라서 본 프로젝트에서는 원활한 교통흐름을 유지하면서도, 우회전 교통사고를 방지하기 위한 운전자, 보행자 양방향 신호체계를 구축하는 것을 목표로 한다.

II. 시스템 설계

■ 시스템 분석 및 세부설계

3 대의 라즈베리 파이를 사용하여 임베디드 시스템을 구축한다. 2 대의 라즈베리 파이는 Sensor 를 담당하고, 1 대의 라즈베리 파이는 Actuator 역할을 수행한다. 각 라즈베리 파이간 통신을 위해 Socket 통신을 활용하며, 2 개의 서버와 1 개의 클라이언트간 통신이 이루어진다. 센서로부터 전달되는 값과 출력 제어의 기준은 모형에 맞게 설정되며, 실제 교통 상황 중 일부(보행신호)를 추상화하여 표현한다.

① 서버 1

보행자 신호를 감지하고, 우회전 차량의 유무를 판별하여 그 결과를 클라이언트에게 전송한다. 이를 위해 버튼과 초음파 센서를 활용한다. 두 개의 스레드를 선언하며, 각 스레드는 서로 다른 Port 번호를 사용하여 클라이언트와 통신한다.

사용센서	역할 및 제어 내역
초음파 센서	<ul style="list-style-type: none">- 초음파 센서를 활용하여 우회전 차량 유무를 판단한다.- 최초 프로그램 실행시, 보행자 신호라고 판단(추상화)한다.- 일정한 간격으로 클라이언트에게 센싱된 값을 전송한다.
버튼	<ul style="list-style-type: none">- 버튼을 활용하여 보행자 신호를 표현한다.- 버튼 입력시 횡단보도 적색불로 판단(추상화)한다.- 버튼 입력 시 클라이언트에게 값을 전송하며, 클라이언트는 버튼 입력에 따라 GPIO, PWM 제어권 반납을 수행한다.

[표 1] 서버 1 제어 센서 별 역할

② 서버 2

횡단보도를 건너는 보행자의 유무를 판단하여 그 결과를 클라이언트에게 전송한다. 이를 위해 2 개의 압력센서를 활용한다. 각 압력센서가 순차적으로 값을 감지할 때를 포착하여 클라이언트에게 센서가 감지한 값을 전송한다.

사용센서	역할 및 제어 내역
압력센서 1	<ul style="list-style-type: none"> - 첫 번째 압력센서에서 값을 감지한다. 횡단보도 인근의 보행자를 인식한다. - Read/Write 동기화를 위해 센서로부터 감지된 값을 클라이언트에게 미리 전송한다. (Page 10)
압력센서 2	<ul style="list-style-type: none"> - 두 번째 압력센서에서 값을 감지할 경우, 횡단보도를 건너는 보행자로 판단한다. - 만약 첫 번째 압력센서에서 값을 감지하고, 두 번째 압력센서에서 값을 감지하지 않을 경우 Read/Write 동기화를 위해 클라이언트에게 0 을 전송한다. (Page 10)

[표 2] 서버 2 제어 센서 별 역할

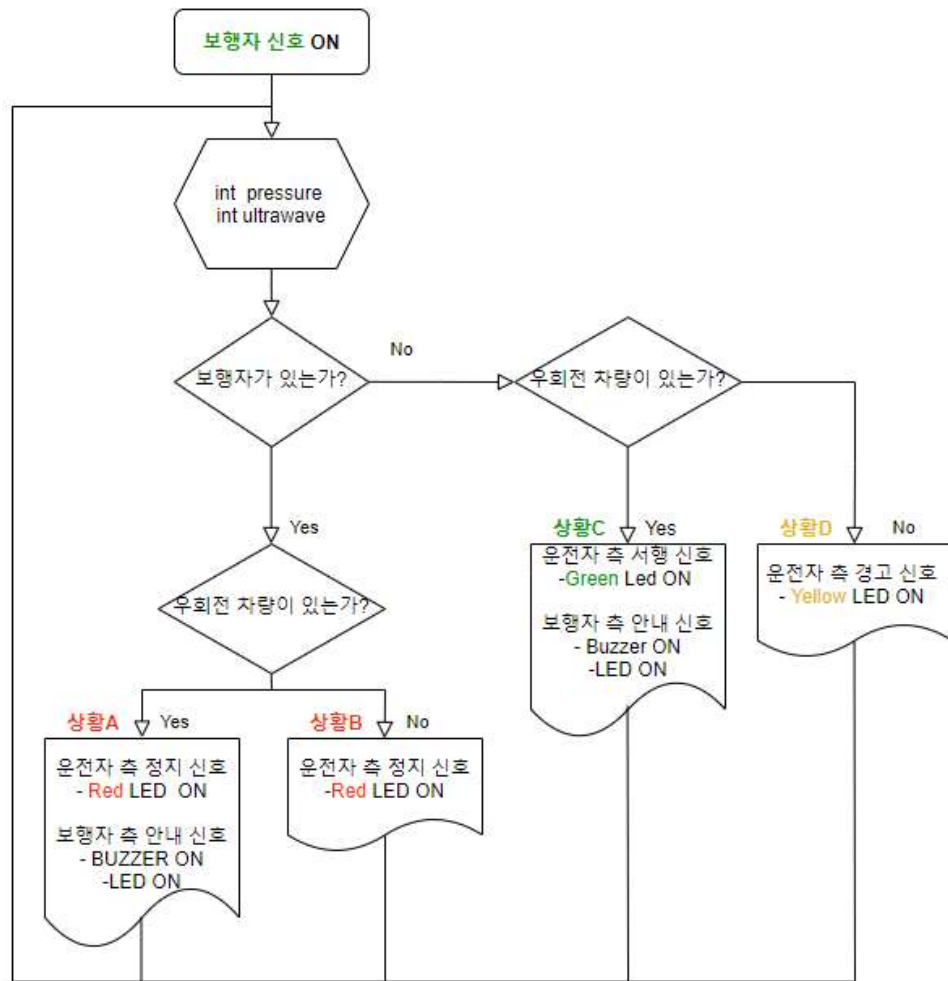
③ 클라이언트

두 개의 서버에게 연결을 요청하고, 센서로부터 감지된 값을 수신한다. 감지한 값에 따라 조건문으로 상황을 분기하여 RGB LED, LED, Buzzer 출력을 제어한다.

사용센서	역할 및 제어 내역
RGB LED	<ul style="list-style-type: none"> - 운전자에게 보행자의 유무를 알리기 위해 사용된다. - Red, Yellow, Green 출력을 제어한다.
LED	<ul style="list-style-type: none"> - 보행자에게 우회전 차량 유무를 알리기 위해 사용된다. - 보행자 신호와 구분하기 위해 일반 LED 출력을 제어한다.
Buzzer	<ul style="list-style-type: none"> - 보행자에게 우회전 차량 유무를 알리기 위해 사용된다. - 시각장애인 보행자를 위해 Buzzer 출력을 제어한다.

[표 3] 클라이언트 제어 센서 별 역할

상황에 따른 출력 제어를 순서도로 나타내면 다음과 같다.



[그림 1] 시스템 출력 신호 알고리즘

■ 설계 변경사항

최초 시스템 설계시 보행자를 인식하기 위해 일체형 보도블럭 밑에 다량의 압력센서를 설치하는 방법을 구상하였다. (표 4) 그러나 제안 발표 이후 Padlet 을 통해 압력센서 위치에 관한 의견을 주고받으며 두 가지 설계 이슈를 파악하였다.

① 압력센서가 내장된 보도블럭을 차도와 가깝게 설치하는 경우

횡단보도 보행자의 보행의도를 미리 파악하기 어렵다. 본 프로젝트의 목적에 부합하지 않으며, 신호 출력을 통한 사고 예방 효과가 크지 않을 것으로 판단하였다.

② 압력센서가 내장된 보도블럭을 인도의 전 범위에 설치하는 경우

보행자의 보행방향을 파악하기 어렵다. 횡단보도를 건너지 않는 보행자가 센서에 감지될 경우 신호에 혼선을 줄 여지가 있다. 이 방법 또한 프로젝트의 목적에 부합하지 않는다.

압력센서가 내장된 일체형 보도블럭을 이중으로 설치한다. 첫 번째 압력센서에서 보행자로 인식할 수 있는 기준값 이상이 감지되면, 대기 상태로 전환한다. 이후 수 초 이내에 두 번째 압력센서에서도 기준값 이상이 감지될 때, 보행자가 있다고 판단한다. 이를 통해 보행의도를 미리 파악함과 동시에, 기존 설계와 비교했을 때 감지 대상의 보행방향을 정확하게 파악할 수 있다. 아래 표는 이를 비교하기 위해 나타내었다.

일반 보도블럭	기존 설치 방법	개선된 설치 방법
		
많은 개수의 센서 필요 보행자 감지가 제한됨	보행자가 패널 위에 서 있을 때 보행자 판단	보행자가 패널을 순차적으로 밟을 때 보행자 판단

[표 4] 압력센서 설치 방법 비교

Ⅲ. 시스템 구현

■ Rpi 별 구현 사항 및 세부 알고리즘

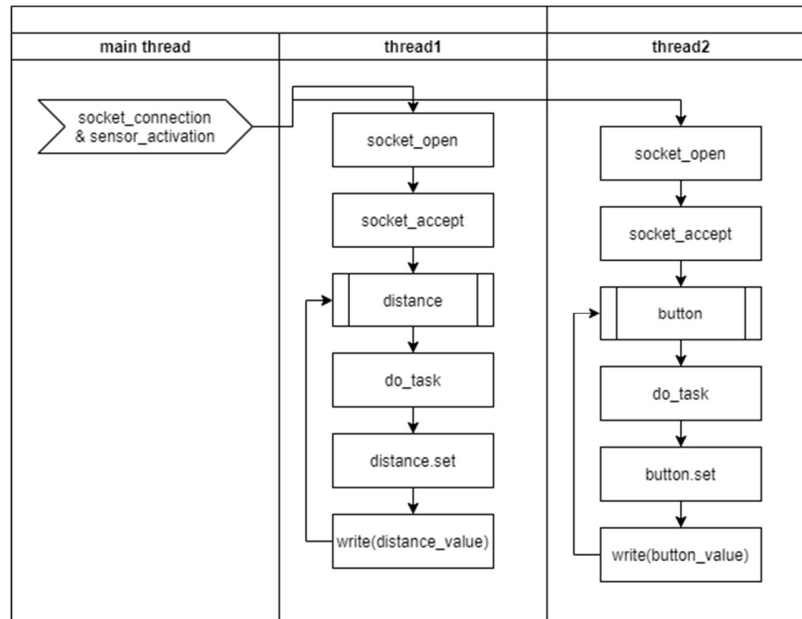


[그림 2] 응용 시스템의 구성요소

본 프로젝트는 사고 예방을 위해 상황에 따라 안정적이고 즉각적인 신호 변화가 중요하다. 그러므로 센서의 값을 Client 에게 보내는 전송 주기를 매우 짧게 설정하였다. 이러한 측면에서 전송 주기, 동기화 등의 이슈를 효율적으로 관리하기 위해 센서의 수에 맞게 스레드를 선언하는 멀티스레드 방식을 채택하였다. 따라서 각 서버와 클라이언트는 다 수개의 스레드를 갖게 되며, 아래 항목은 스레드의 활용을 구체화하여 나타낸 것이다.

① Server1

클라이언트에게 초음파 센서와 버튼 값을 Write 하기 위해 두 개의 스레드를 선언한다. 각 스레드는 서로 다른 포트 번호를 통해 클라이언트와 통신한다.



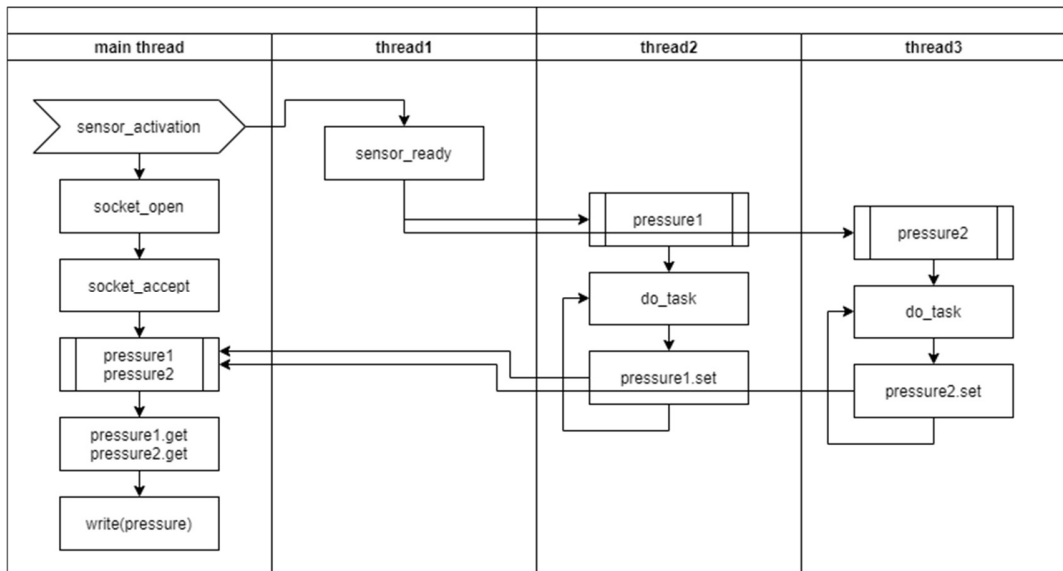
[그림 3] 서버 1 의 멀티 스레드 프로그래밍 구조

코드 중 일부	설명
<pre> while (1) { state =GPIORead(PIN); printf("button state: %d!!!!!!\n", state); if(prev_state ==0 && state==1){ printf("~~~~~button_while~~~~~\n"); snprintf(msg,2,"%d",9); write(clnt_sock, msg, sizeof(msg)); printf("msg=%s\n",msg); sleep(1); } prev_state = state; </pre>	<ul style="list-style-type: none"> - 보행자 신호를 판단한다. - 버튼 trigger 발생 시 클라이언트에게 9 를 보낸다. - 클라이언트는 9 를 Read 할 경우, 보행자 신호가 적색으로 바뀌었다고 판단한다. - GPIO, PWM 제어권을 반납한다.

<pre> usleep(5000); } </pre>	
▲버튼 값을 보내기 위해 선언한 스레드 코드 중 일부	

② Server2

메인 함수에서 보행자 감지 시스템을 가동시킬 1 개의 스레드(thread_1)를 선언한다. 해당 스레드(thread_1)는 2 개의 압력센서를 활성화시키는 pres_sensor_act() 함수를 실행한다. 이때, pres_sensor_act() 함수 내에서도 2 개의 스레드(thread_2, thread_3)를 선언하여, 압력센서의 감지값을 비동기적으로 읽어온다.



[그림 4] 서버 2 의 멀티 스레드 프로그래밍 구조

<보행자 감지 알고리즘>

압력센서①, ②는 각각 횡단보도를 기준으로 바깥쪽 보도블럭과 안쪽 보도블럭에 위치한다. 먼저, 바깥쪽 압력센서(①)가 30 이상의 값을 감지하면 클라이언트에게 전송할 임시 메시지에 해당 값을 미리 write 한다. 이후 안쪽 압력센서(②)에서 30 이상의 값이 감지되지 않으면, 메시지에 0 을 rewrite 함으로써 보행자가 없다는 시그널을 전송한다.

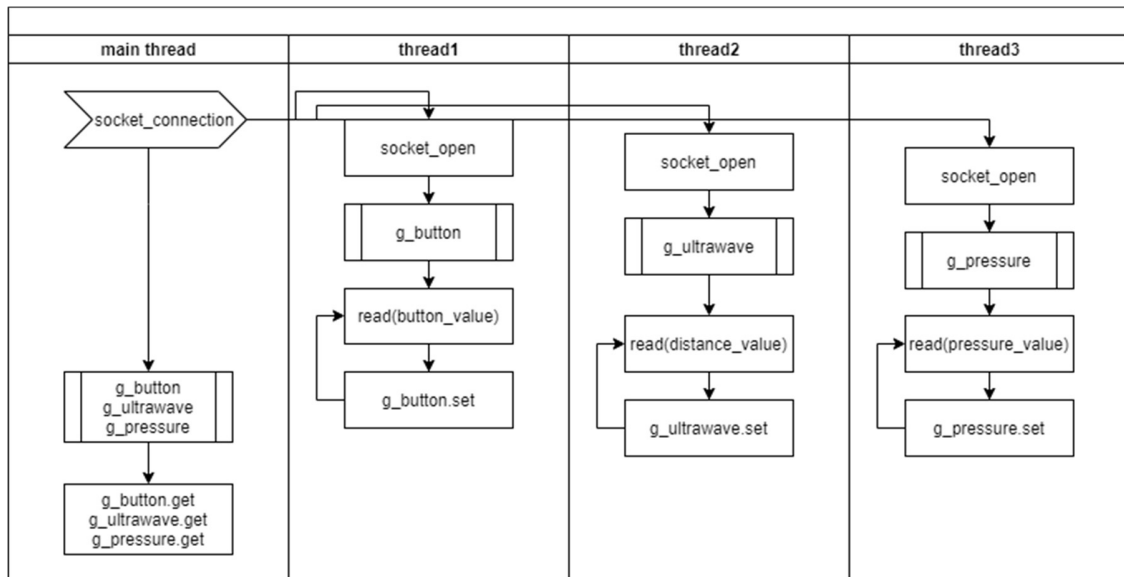
즉, 바깥쪽 보도블럭과 안쪽 보도블럭에 차례로 값이 들어온 경우, 미리 write 해놓은 메시지를 전송함으로써 딜레이를 줄이고, 그렇지 않은 경우에는 이전에 write 한 값을

0 으로 무효화시킨다. 단, 서버 2 에서 임시 메시지를 작성하는 시점에 클라이언트가 값을 읽으면 시스템이 오작동할 수 있으므로, 서버와 클라이언트 사이의 동기화 타이밍을 조정하여 클라이언트가 정상적인 시점에 메시지를 읽을 수 있도록 처리한다.

코드 중 일부	설명
<pre> while (1) { out_state = pressure1; printf("*** out_state: %d ***\n", out_state); if (out_state > 30) { // by person; 100 snprintf(msg, 4, "%d", out_state); res = write(clnt_sock, msg, sizeof(msg)); sleep(1); in_state = pressure2; printf("*** in_state: %d ***\n", in_state); if (in_state < 30) { snprintf(msg, 4, "%d", 0); res = write(clnt_sock, msg, sizeof(msg)); sleep(1); printf("res = %d, msg = %s\n", res, msg); } } else { snprintf(msg, 4, "%d", 0); res = write(clnt_sock, msg, 4); sleep(1); printf("res = %d, msg = %s\n", res, msg); } usleep(50000); } </pre>	<ul style="list-style-type: none"> - 횡단보도를 건너는 보행자를 감지한다. - 빠른 전송을 통한 동기화를 위해 첫 번째 압력센서에서 30 이상의 값을 감지하면 write 한다. - 만약 두 번째 압력센서에서 기준 값 이상이 인식되지 않으면, 0 을 write 함으로써 보행자가 없다고 판단한다.
▲ 횡단보도 보행자 판단을 위한 압력센서 값 제어	

③ Client

서버 1 과 통신하기 위한 스레드 2 개, 서버 2 와 통신하기 위한 스레드 1 개를 선언한다. 각 스레드는 초음파 센서, 버튼 센서, 압력 센서의 값을 read 한다. 이 값을 전역변수에 대입하여 메인 함수에서 출력을 제어한다.



[그림 5] 클라이언트의 멀티 스레드 프로그래밍 구조

코드 중 일부	설명
<pre> while(1) { pressure =read(sock,msg,sizeof(msg)); if (pressure == -1) error_handling("read() error"); g_pressure= atoi(msg); printf("Receive pressure data : %s\n", msg); sleep(1); } </pre> <p>▲ 압력 센서의 값을 Read 하는 스레드 코드 중 일부</p>	<ul style="list-style-type: none"> - 서버 2 로부터 압력센서 값을 읽는다. - 읽은 값을 정수형으로 변환하여 전역 변수(g_pressure)에 대입한다.

IV. 시험평가

■ 개발환경

운영체제 : Windows 10 , Raspbian OS

사용 에디터 : Visual Studio Code, Geany

사용 언어 : C

소스코드 목록: client.c, server1.c, pressure.h, pressure.c, server2.h, server2.c

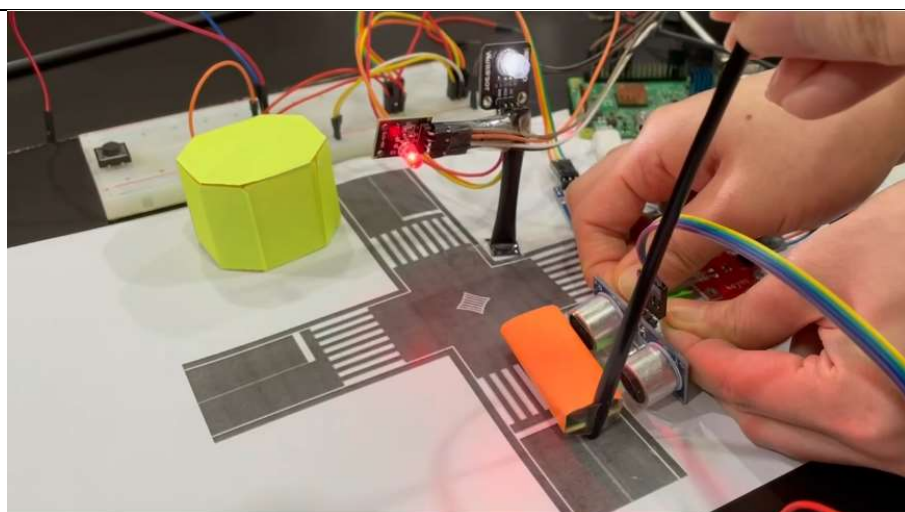
컴파일 방법:

```
gcc -o server1 server1.c -lpthread
gcc -c -o pressure.o pressure.c (*)
gcc -c -o server2.o server2.c (*)
gcc -o server2.out pressure.o server2.o -lpthread (*)
gcc -o client client.c -lpthread
```

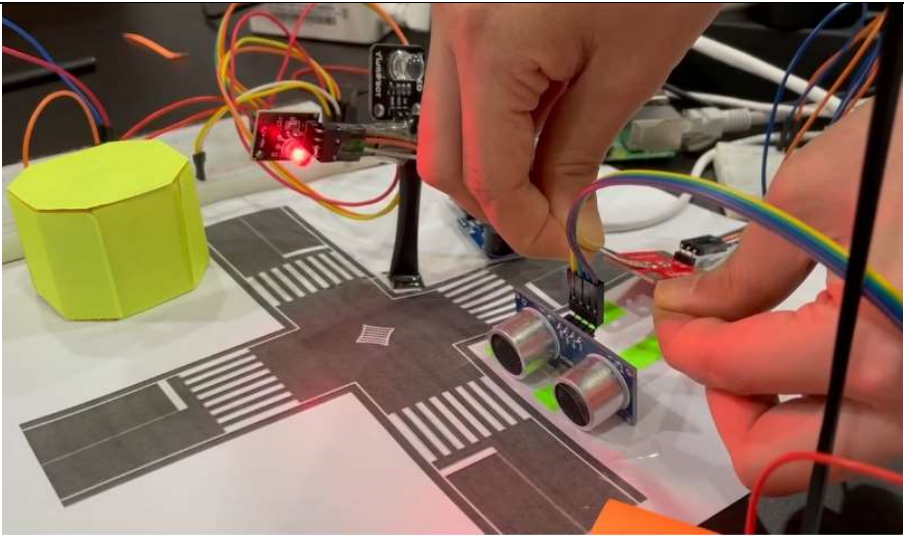
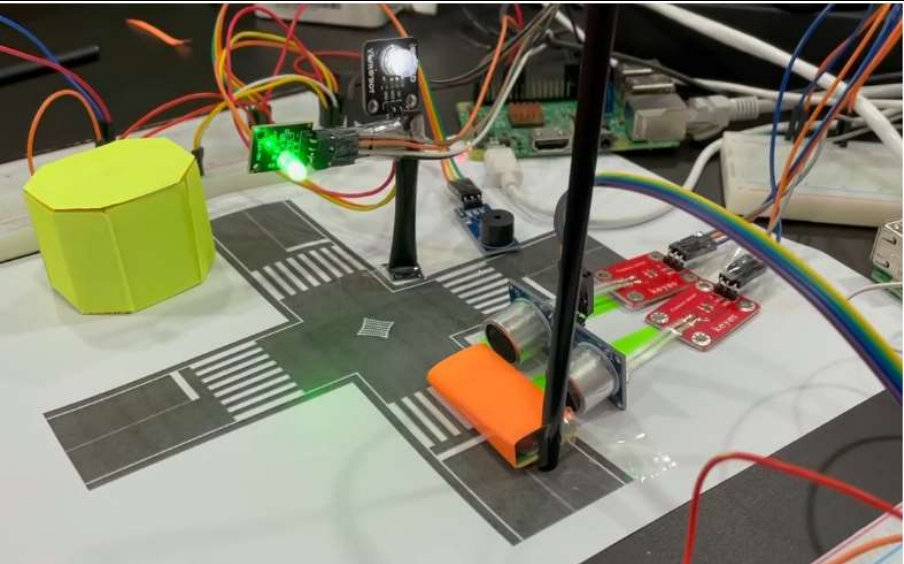
* server2 관련 헤더/소스 파일(pressure.h, pressure.c, server2.h, server2.c) Makefile 작성 → make 커맨드를 이용하여 컴파일 가능

■ 테스트 결과

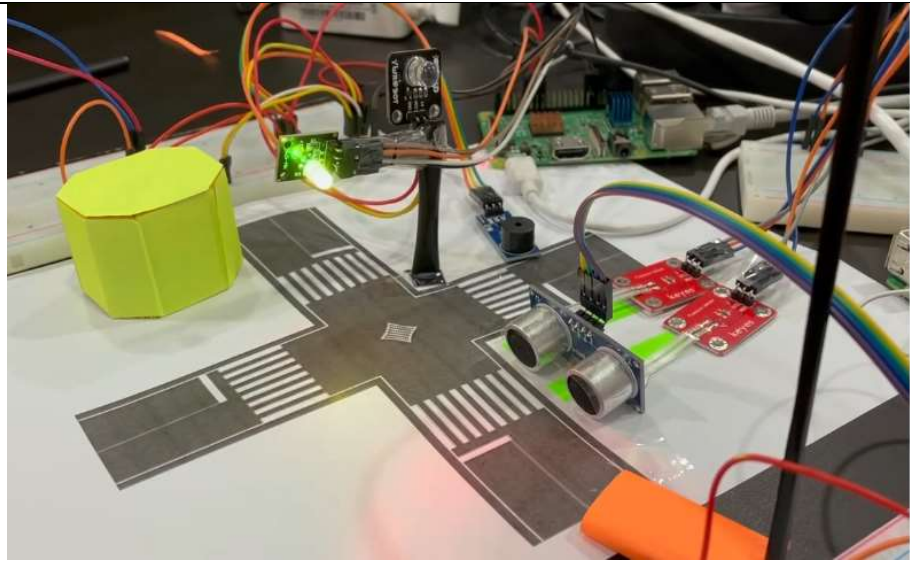
상황 A



▲ 보행자, 우회전 차량 모두 감지된 경우 (Buzzer 출력 확인됨)

<p>상황 B</p>	 <p>▲보행자만 감지된 경우</p>
<p>상황 C</p>	 <p>▲ 우회전 차량만 감지된 경우 (Buzzer 출력 확인됨)</p>

상황 D

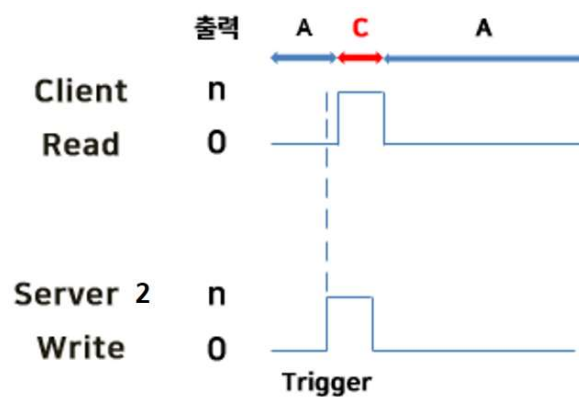


▲ 우회전 차량, 보행자 모두 없는 경우

V. 개발이슈

■ 동기화 문제

클라이언트가 서버로부터 센서의 값을 Read Operation 을 수행하는 과정에서 동기화 문제가 발생하였다. 서버 2 는 보행자가 감지된 경우(trigger) n 값을 write 하고, 보행자가 없으면 0 을 write 한다. 특정 상황에서 클라이언트는 n 값을 Read 하고, 다시 0 을 Read 하기 전에 상황에 맞는 출력을 제어해야 하는데, 이 간격이 너무 짧아서 출력 제어가 원활하지 않았다.



[그림 6]

이를 해결하기 위해 서버 2 의 Write 알고리즘을 수정하였다. 먼저, 바깥쪽 압력센서가 n 값을 감지하면 대기 상태로 변경되며, 감지한 값을 클라이언트에게 전송한다. 이후, 안쪽 압력센서가 기준 값 미만을 감지하면 보행자가 없다고 판단하여 다시 0 을 클라이언트에게 전송한다. 만약 보행자가 바깥쪽 압력센서에만 위치한 경우, n 값이 감지되지만, 수 ms 또는 그보다 적은 시간 이내에 0 을 rewrite 하기 때문에 횡단보도 보행자를 정확히 구분할 수 있으며, 센서의 값을 기존 코드보다 빠르게 보내므로 효율적이다. 이러한 알고리즘을 통해 동기화 문제를 적절히 해결하였고, 통신 및 출력제어가 원활히 수행되는 것을 확인하였다.

■ 제어권 반납 문제

라즈베리 파이 환경에서 센서를 제어하기 위해 GPIO, PWM, SPI 인터페이스를 활용하였고, 이들의 제어권을 조작하여 센서의 동작을 제어하였다. 또한, 통신을 위해 다수개의 Socket 프로그래밍을 하였다. 그러나 초기 테스트 과정에서 프로그램을 중간에 종료할 때마다 제어권을 반납하지 않아서 프로그램 재실행이 안되는 문제가 있었다. 이를 해결하기 위해 횡단보도 보행자 신호를 추상화한 버튼 센서를 활용하였다. 버튼 입력시 클라이언트에게 msg 를 전송하며, 이 값을 읽은 클라이언트는 모든 출력을 반납하고, 프로그램을 종료하도록 구현하였다. 클라이언트가 종료되면, 서버의 Socket 이 닫히면서 모든 제어권을 반납하게 된다.

VI. 기대효과 및 후속연구

본 프로젝트는 기존 우회전 전용 보조 신호등에 비해 우회전 교통사고를 방지하는 데 효과적이다. 현행 도로교통법상, 교차로 우회전 상황에서 운전자는 보행자가 없을 때 횡단보도 신호와 상관없이 서행으로 지나갈 수 있다. 이는 원활한 교통흐름을 위한 것이다. 그러나 기존의 우회전 전용 보조 신호등은 보행자 유무와 상관없이 차량의 흐름을 정지시킨다. 이는 사고예방을 위한 것이나, 도로교통법의 취지와 맞지 않는다. 또한, 우회전 전용 신호등은 표지판, 가로수 등에 의해 신호 가시성이 매우 떨어진다. 이러한 실정은 운전자들의 우회전 전용 신호 인식이 떨어트린다.

본 프로젝트는 이러한 문제를 효과적으로 해결하였다. 보행자 유무와 우회전 차량 유무를 판단하여 상황에 따라 유동적으로 정지, 서행 신호를 제공한다. 이를 통해 원활한 교통흐름을 도모할 수 있다. 또한, 본 프로젝트는 운전자와 보행자 모두에게 신호를

제공함으로써 운전자, 보행자 모두에게 위험한 상황을 사전에 식별하도록 돕는다. 가령, 운전자가 보행자를 미처 보지 못했을 경우, 보행자가 뒤늦게 뛰어들어가는 경우, 자전거나 공유 킥보드가 빠른 속도로 보행하는 경우 등 다양한 상황에서의 사고를 미연에 방지할 수 있다.

향후 보행자, 운전자 데이터를 활용하여 개선시 다양한 사고 예방 기능을 수행할 수 있을 것으로 판단한다. 비단 우회전 사고뿐만 아니라, 야간시에 차량이나 보행자 유무에 따라 신호를 조절할 수 있을 것이다. 이 밖에도, 우회전 차량이 많을 때와 적을 때, 보행자가 많을 때와 적을 때 등의 시간별, 날씨별 데이터를 수집하여 조금 더 세분화된 신호체계를 구성할 수 있을 것이다.

VII. 수행경과

날짜	진행 내용
5 월	<p>3 일 : 아이디어 회의, 사용 가능한 센서 목록 체크</p> <p>5 일 : 6 가지 아이디어 중 우회전 전용 신호등 시스템 채택</p> <p>6~10 일 : 자료 조사 및 PPT, 제안서 작성, 피드백 반영</p> <p>14~18 일 : 인터페이스의 이해 및 제어 함수 구현 및 학습 공유</p> <p>19 일 : 부저 (PWM), 초음파센서(GPIO), 압력센서(SPI) 동작제어 및 연계 구현 및 코드 리뷰</p> <p>21 일 : SPI 인터페이스 API 중 adc converter 에 의한 bit-operation 과 디지털 신호 변환 매커니즘 학습, 스레드 활용</p> <p>26 일 : 센서의 비동기적 동작 테스트, 조원 간 테스트 내역 공유</p> <p>28 일 : 2 개의 서버와 1 개의 클라이언트 구조로 통신 설계</p> <p>29 일 : 보행의도 감지를 위해 서버 2 알고리즘 변경, 스레드를 활용한 통신 구현</p> <p>30 일 : Bind, Connection 오류 식별</p>
6 월	<p>1 일 : Bind, Connection 오류 해결, 서버 2 와 클라이언트 간 동기화 및 제어권 반납 문제 식별, 서버 1 라즈베리 파이 고장으로 교환, Bread Board 문제로 인하여 초음파 센서가 제대로 동작하지 않는 문제 식별</p>

	~6 일 : 오류 해결 및 정상 동작 확인 ~8 일 : 발표자료 준비
--	---

- Gitlab 링크 : <https://git.ajou.ac.kr/leehyehoon/rtl/-/tree/master>