임베디드시스템소프트웨어1
팀프로젝트 최종보고서

제목: KU Smart Farm

2019. 06. 25

컴퓨터공학과	201311276	박 형 민
컴퓨터공학과	201311287	엄 현 식
컴퓨터공학과	201311306	이 진 호
컴퓨터공학과	201311320	한 예 훈
멘토		

1. 서 론

이번 학기에 저희는 커널 모듈을 활용하여 다양한 종류의 캐릭터 디바이스를 제어하는 방법들을 배웠습니다. 배웠던 캐릭터 디바이스들로는 외부의 값을 읽어오는 Sensor나 특정 동작을 행하는 Actuator들이 있었는데, 이러한 디바이스들을 모두 적절히 활용할 수 있는 분야로 '농업'을 떠올려 보았습니다.

그 이유는 1995년부터 본격적으로 농업인구가 감소하고 있는 가운데, 해결방안 중 하나로 '농장 관리시스템의 무인화'가 있습니다. 하지만 무인화 시스템을 안정적으로 구축하기 위해서는 다양한 디바이스들이 동시에 작동하고, 해당 디바이스들을 통합적으로 관리할 중앙시스템을 필요로 하게 됩니다. 저희는 그러한 요건들이 이번 팀 프로젝트의 주제와 잘 어울린다고 생각했기에, 최종적으로 'KU Smart Farm(이하 KSF)'이라는 이름의 '자동 농장 관리 시스템'을 만들게 되었습니다.

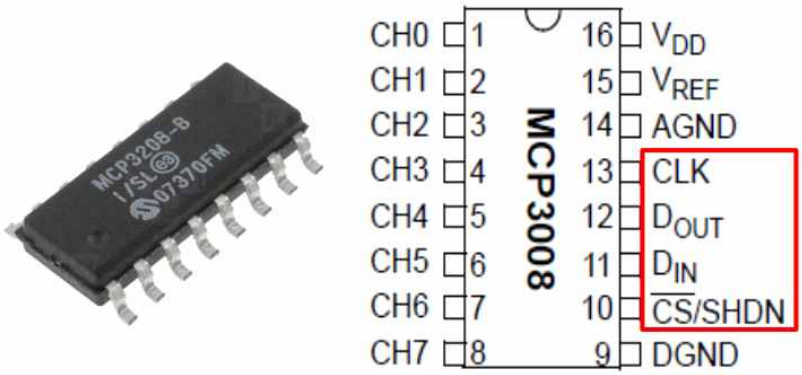
저희 KSF는 이미 활용 중인 기존의 스마트팜 시스템들과 유사한 부분이 많습니다. 가장 먼저 현재 환경 값들을 지속적으로 수집하여 필요한 상황이 되면 자동으로 기계들이 동작하여 항상성을 유지시켜줍니다. 그러한 부분 이외에도, 유저 인터페이스를 활용하여 사용자가 원할 때에도 특정 기계들을 동작시키거나 환경 값을 알아낼 수 있습니다.

KSF가 갖는 다른 스마트팜 시스템과의 차별성으로는 기존의 다른 스마트 팜과 관련한 많은 시스템들은 대부분 값비싼 장비들임에 반해서, 저희 장비들은 비교적 경제적인 부품들을 이용하여 시스템을 구현하는 데에 차별성이 있습니다.

2. 배경지식

MCP 3208

라즈베리 파이의 GPIO는 디지털 입력만 받아들일 수 있습니다. 따라서 아날로그 센서로부터 아날로그 데이터 값을 입력받기 위해 해당 값을 디지털 데이터로 변환해주는 ADC가 필요합니다. ADC 중에서도 MCP 3208은 아날로그 출력 값을 8채널 12bit 디지털 값으로 변환하여 출력합니다. 또한, 저희는 MCP 3208을 SPI 인터페이스를 이용해서 사용하였습니다.

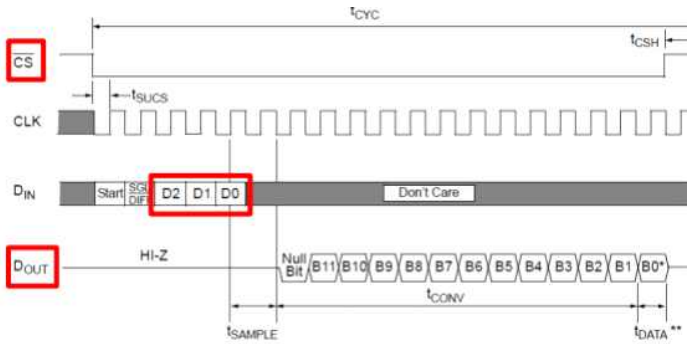


MCP 3208과 라즈베리의 GPIO를 연결할 때 CLK는 SCLK, D_{out}은 SPIMISO, $D_{\in}$은 SPIMOSI, CS/SHDN은 해당하는 CE값에 연결을 하면 됩니다.

SPI 통신

SPI 통신은 직렬 통신을 위한 통신 방법 중 하나로 동기식 전송을 합니다. 또한 Single Master 구조를 가지고, Slave는 Multi-slave 형태로 여러 개를 가질 수 있습니다. I²C와 달리, SPI는 4개의 핀을(CE, SCLK, MOSI, MISO) 사용하여 통신하며 I²C보다 빠르고 Full-Duplex 통신을 지원합니다.

KSF는 조도 센서와 토양 온습도 센서를 사용하기 위해 SPI 통신을 이용하였고 CE값을 (CE0 GPIO 8번, CE1 GPIO 7번)을 사용하여 센서를 구분해서 라즈베리 파이와 통신을 했습니다.

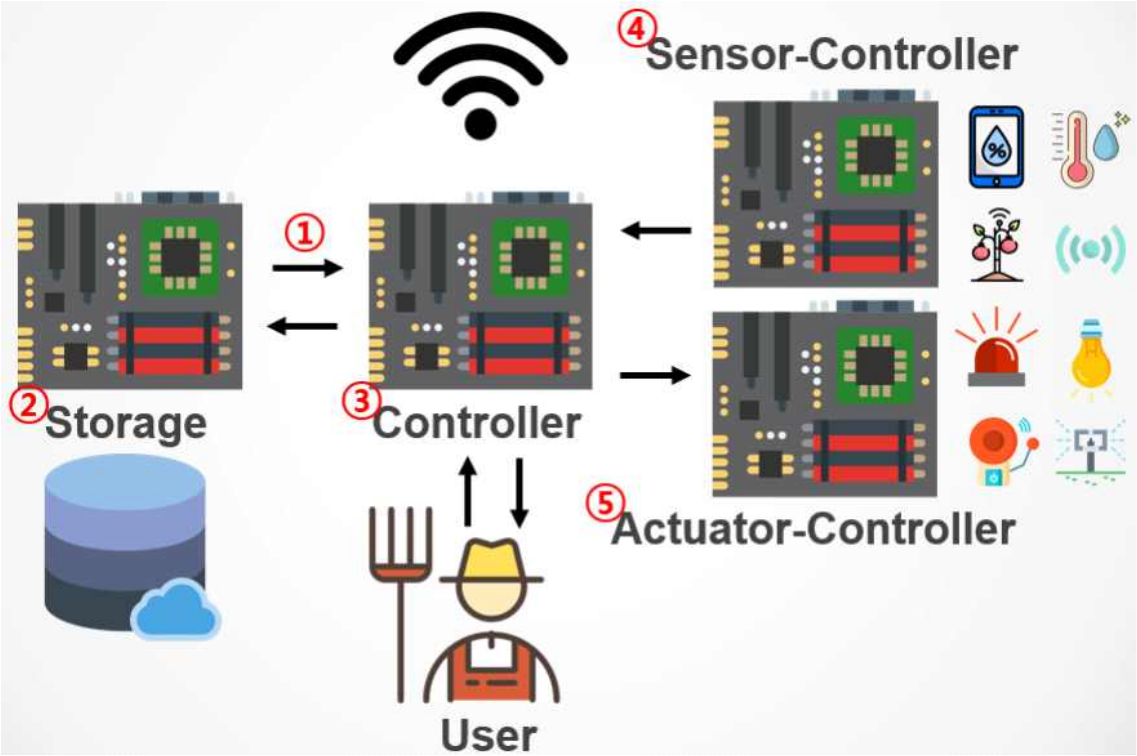


통신을 위해 CE값을 1에서 0으로 주고 SCLK 신호를 주면서 $D_{\in}$에 Start, DIFF 값을 넣고 채널에 따라 D_2, D_1, D_0 값을 각각 넣으면 D_{out}을 통해 데이터를 받을 수 있습니다.

```
void start_mcp(void){
        gpio_direction_output(CE0,0);

        gpio_direction_output(SCLK,1);
        udelay(1);
        gpio_direction_output(SCLK,0);
        gpio_direction_output(MOSI,1);
        udelay(1);
        gpio_direction_output(SCLK,1);
        udelay(1);
        gpio_direction_output(SCLK,0);
        gpio_direction_output(MOSI,1);
        udelay(1);
        gpio_direction_output(SCLK,1);
        udelay(1);
        gpio_direction_output(SCLK,0);
        gpio_direction_output(MOSI,0);
        udelay(1);
        gpio_direction_output(SCLK,1);
        udelay(1);
                gpio_direction_output(SCLK,0);
                gpio_direction_output(MOSI,0);
                udelay(1);
                gpio_direction_output(SCLK,1);
                udelay(1);
                gpio_direction_output(SCLK,0);
                udelay(1);
                gpio_direction_output(SCLK,1);
                udelay(1);
                gpio_direction_output(SCLK,0);
                udelay(1);
                gpio_direction_output(SCLK,1);
                udelay(1);
                gpio_direction_output(SCLK,0);
                udelay(1);
                gpio_direction_output(SCLK,1);
                udelay(1);
                gpio_direction_output(SCLK,0);
                udelay(1);
}
```

3. 본 문



① Communication

저희는 모듈 간 통신 방법으로 소켓 통신 프로그래밍을 사용하였습니다. 위의 그림에서 확인할 수 있듯이 KSF는 활발한 통신이 이루어지는 환경입니다. 이를 효율적으로 관리하고 해결하기 위해서 다음과 같이 KSF_NET 통신 라이브러리를 구현하였습니다.

```
/* Client Function */
int client_open(char *dest_ip, int port, long timeout);
struct response request(int sock, char method, char type, char cmd, unsigned long len, char *data);
int client_close(int sock);
/* Server Function */
int server_open(int port);
int server_close(int sock);
int wait_request(int sock, struct request *req);
int response(int c_sock, char type, unsigned long len, char *data);
```

KSF_NET은 REST 통신의 형태로 “request & response”를 지원합니다. 이에 대해서 KSF는 요청을 보내는 ‘Client’와 요청을 받는 ‘Server’의 역할을 하는 모듈로 구성되어 있습니다.

Request의 Method는 특정 정보를 요청하는 @GET, 특정 값의 변경을 요청하는 @POST, 특정 값의 저장을 요청하는 @PUT 총 3가지를 지원합니다. Client는 Server에게 Request를 보내고, Server는 이에 응답하여 (method, type, cm, data)를 해석하여 적절한 기능을 수행합니다. 또한 수행한 결과에 대해서 Client에게 Response를 보내면서 부분적인 통신을 마칩니다.

② Storage(R3)

Storage의 경우에는 Controller에서 보내주는 데이터 값들을 저장하고, Controller가 특정 데이터를 요청했을 때 저장되어 있는 값들을 계산하여 전송해줍니다. 이때 저희는 File System을 이용하여 저장소를 만들었고, soil, light, alert_log(EMG_T, EMG_P)로 구분하여 저장 파일을 구성하였습니다.

데이터를 저장할 때는 char 문자열 형태로 저장을 하고, atoi() 함수를 이용하여 문자열 데이터를 숫자로 변경시켜줍니다. 이후 sprintf() 함수를 이용하여 결과 값을 다시 문자열로 변경시켜주고, 이를 ①에서 미리 선언해둔 구조체의 인자 값에 포함시켜 전송하게 됩니다.

③ Controller(R2)

R2의 Data Controller의 역할은 R1의 센서에서 전송한 데이터 값들을 가공하여 R3의 Storage Controller에 전송하는 역할입니다. Command Controller는 KSF에서 중앙 제어의 역할을 하며, R3의 Refine Controller에게 필요한 데이터를 요청하고, 받은 값에 따라 R4의 Actuator Controller에 명령을 내립니다.

④ Sensor-Controller(R1)

Sensor-Controller의 경우 fork 함수를 통해 세 개의 프로세스들을 만들었습니다. 한 프로세스에서는 주기적으로 조도 센서와 토양 온습도 센서로부터 각각 조도 값과 토양 습도 값을 Controller(R2)에게 값을 전송합니다. 또 다른 프로세스에서는 PIR 값에 인터럽트가 발생하기 전까지 Sleep하고 있다가 인터럽트가 발생하면 Ultra_Sonic 센서를 통해 거리를 읽어 미상의 물체가 일정 거리 안으로 들어오면 바로 Actuator-Controller(R4)에게 비상 신호를 전달하고 Controller(R2)에게도 이 사실을 알립니다. 마지막 남은 프로세스는 주기적으로 대기 중 온도 값을 측정해서 일정 온도 이상으로 올라가면 마찬가지로 Actuator-Controller(R4)에게 비상 신호를 전달하고 Controller(R2)에게도 이 사실을 알립니다.

기본적으로 Sensor-Controller는 Controller(R2)와 통신을 하지만 비상 상황의 경우 신속한 대처를 위해 Actuator-Controller와 직접 통신을 하며 비상 신호를 전달합니다.

⑤ Actuator-Controller(R4)

Actuator Device를 제어하는 모듈입니다. 제어하는 장치는 “LED(광합성용), LED(비상 알림용), Buzzer(비상 알림용), Motor(Sprinkler), Water pump(Sprinkler)” 총 6개입니다.
해당 모듈은 R2, R1으로부터 요청을 대기하는 Server의 형태를 가지기에, 각 장치에 대한 제어 신호를 받아 적절하게 기능을 수행합니다.

4. 소프트웨어 사용 방법

- 소프트웨어 설치

PC에서 R1, R2, R3, R4 각각

```
sh compile.sh
```

커맨드를 입력 후 라즈베리파이 패스워드를 입력하면, 컴파일하고 라즈베리파이에 scp로 실행파일이 전송됩니다. 또한 장치를 제어하는 R1, R4의 경우 KERNEL/ 에 존재하는 compile.sh 스크립트를 실행하면 각 모듈을 컴파일하여 전송합니다.

- 소프트웨어 실행

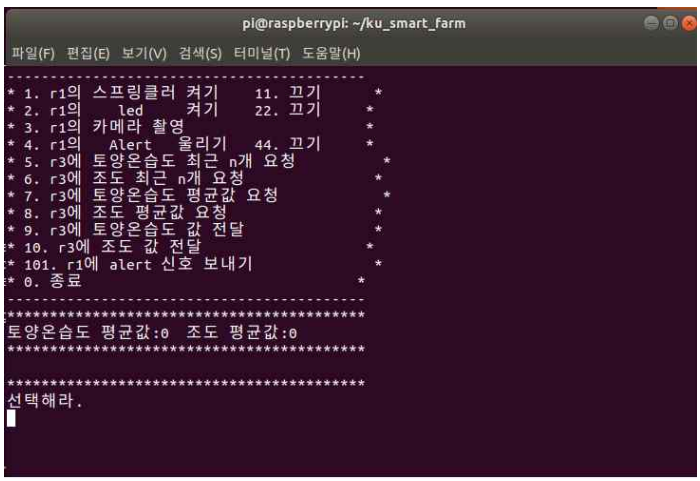
Module

1. R1에서 module_insert.sh & sensor_mknod.sh 실행
2. R4에서 module_insert.sh & actuator_mknod.sh 실행

Controller

1. R3에서 Storage Controller, Refine Controller 실행
2. R2에서 Data Controller, Command Controller 실행
3. R1에서 Sensor Controller, Actuator Controller 실행
4. R2에서 User Interface 실행(선택)

- 사용을 위한 매뉴얼



위의 인터페이스 구성처럼 R1의 스프링클러, LED, 카메라, 버저를 끄고 켜는 신호를 보내는 것과 R3에게 데이터를 요청하고 출력하거나, R3에 조도 값과 Alert 신호를 보내는 명령을 수행할 수 있습니다.

- 소스 코드 링크 (github)

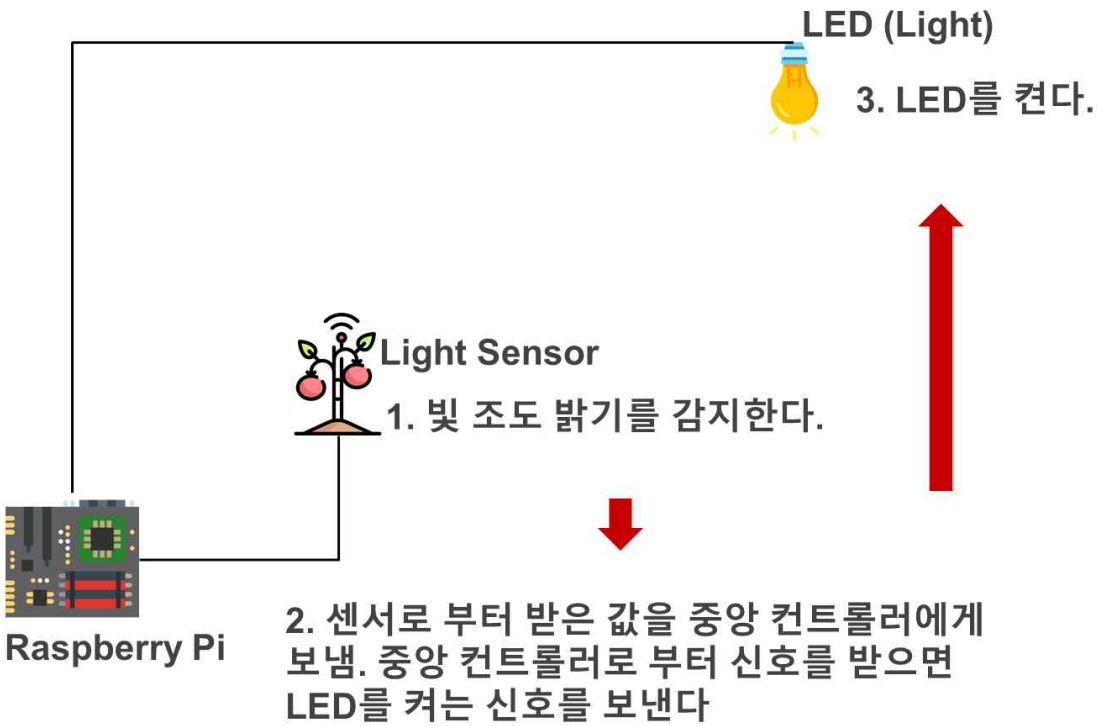
https://github.com/aeomhs/EMSO_TERM

5. 시연 시나리오

중간발표 때 발표 했던 시나리오1,2,3,4 그대로 시연 시나리오를 진행했습니다.

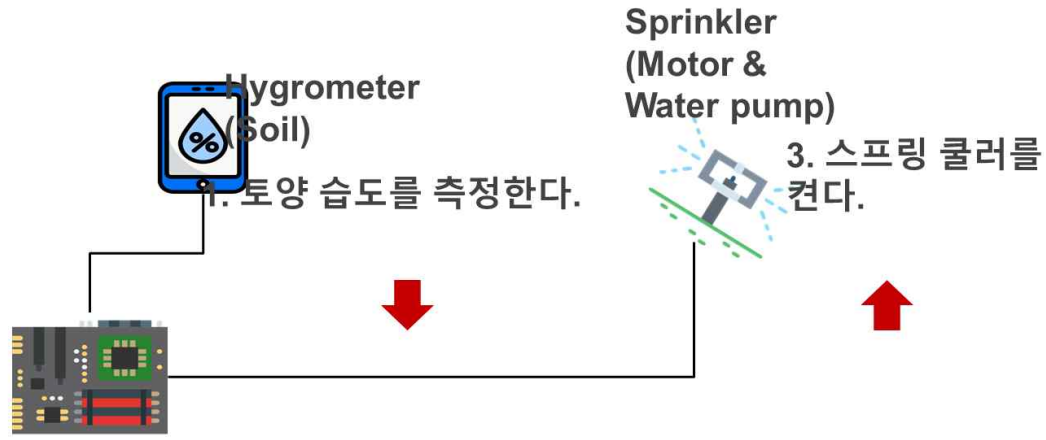
- 시나리오1

1. 빛 조도를 주기적으로 측정합니다.
2. 센서로 부터 받은 값을 R2의 Data Controller에게 전송하고, R2의 Data Controller는 R3의 Storage Controller에게 데이터를 전송합니다.
3. R2의 Command Controller는 R3의 Refine Controller에게 주기적으로 조도 평균값을 요청합니다.
4. R2가 받은 조도 평균값이 일정 수준 이하가 되면, R2의 Command Controller가 R4의 Actuator Controller에게 LED를 On하라는 신호를 보냅니다.
5. LED가 켜집니다.



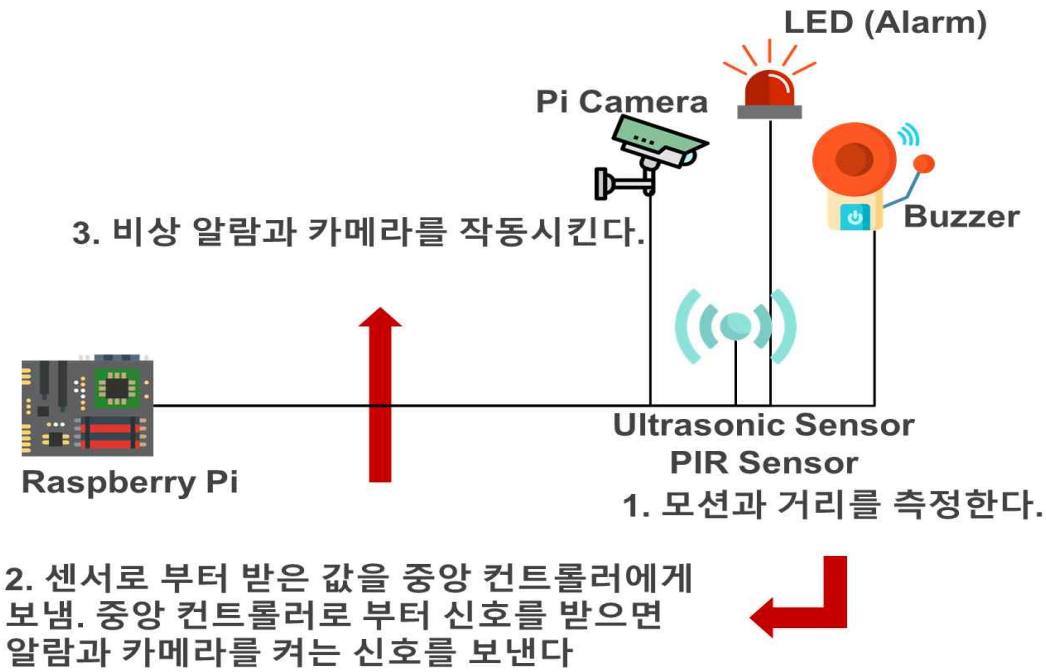
- 시나리오2

1. 토양 습도를 주기적으로 측정합니다.
2. 센서로 부터 받은 값을 R2의 Data Controller에게 전송하고, R2의 Data Controller는 R3의 Storage Controller에게 데이터를 전송합니다.
3. R2의 Command Controller는 R3의 Refine Controller에게 주기적으로 토양 습도 평균값을 요청합니다.
4. R2가 받은 토양 습도 평균값이 일정 수준 이하가 되면, R2의 Command Controller가 R4의 Actuator Controller에게 Sprinkler를 On하라는 신호를 보냅니다.
5. Sprinkler가 켜집니다.



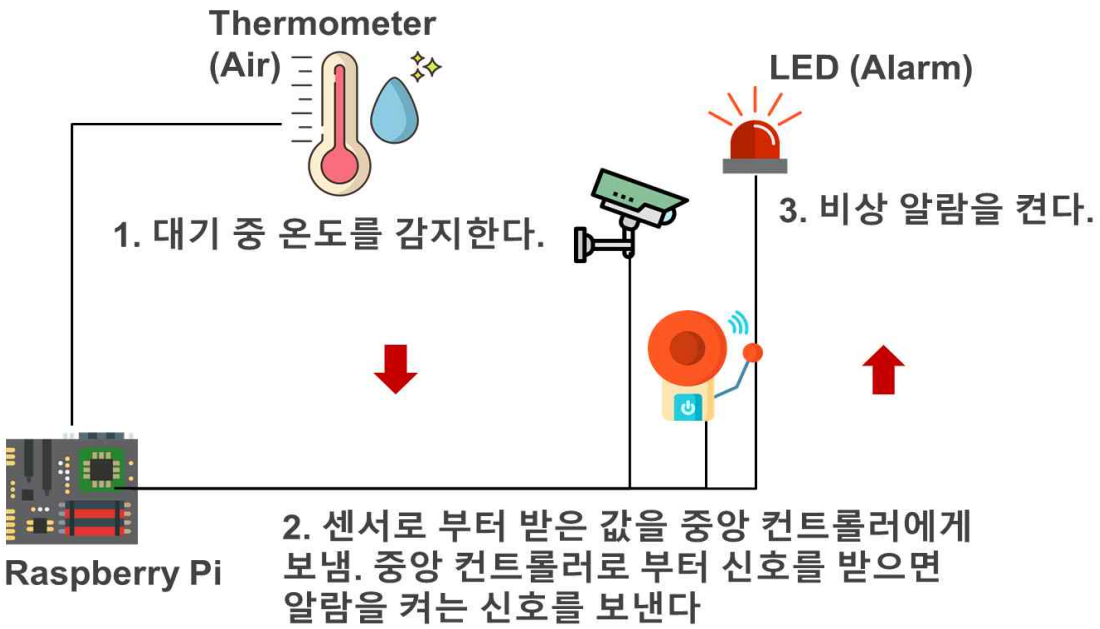
- 시나리오3

1. PIR 센서가 외부 물체의 움직임을 지속적으로 측정합니다.
2. 센서가 Rising 되면, 10초의 타이머가 실행됩니다. 해당 타이머 동안 UltraSonic 센서가 계속 물체와의 거리를 잽니다.
3. R1이 측정한 물체와의 거리 값이 비정상적으로 낮으면 물체가 가까이 접근하였다고 판단을 하게 되어 비상 알람을 켜고, LED가 켜지며, 카메라로 농장을 촬영합니다.
4. 이후 해당 거리 값을 기록하기 위하여 R3로 해당 거리 값을 R3의 Storage Controller로 전송해 줍니다.

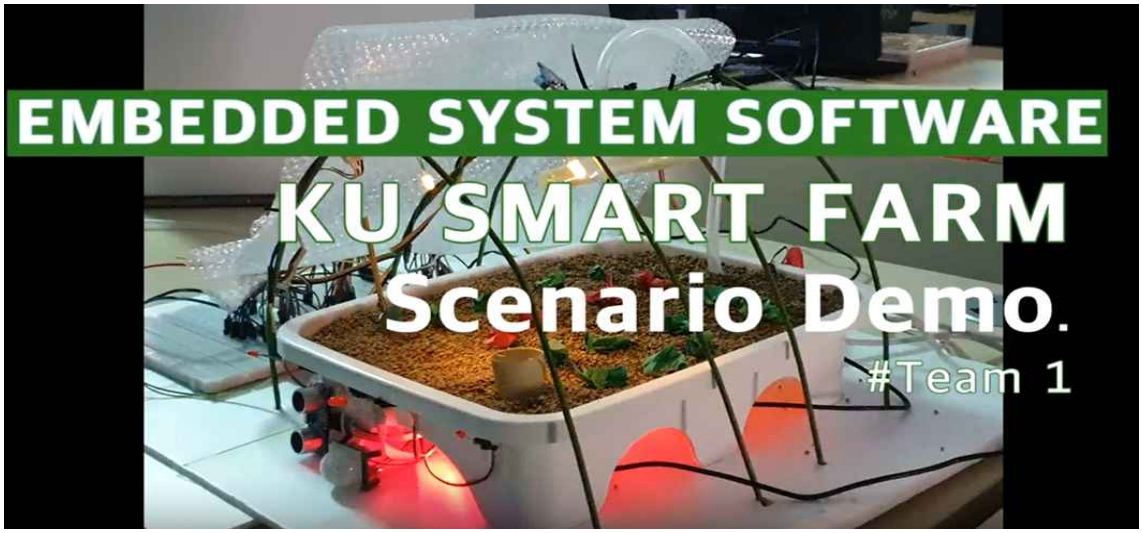


- 시나리오4

1. 대기 중 온도를 주기적으로 측정합니다.
2. 센서로 부터 받은 값을 R2의 Data Controller에게 전송하고, R2의 Data Controller는 R3의 Storage Controller에게 데이터를 전송합니다.
3. R1이 측정한 대기 중 온도 값이 급격한 변화가 있으면 비상 알람을 켜고, LED가 켜지며, 카메라로 농장을 촬영합니다.



- 데모 동영상 링크 (youtube)



-> https://youtu.be/uXOjjKhMu4A

6. 결론 및 향후 계획

결론

저희는 HTTP의 Rest API의 형식을 본 따서, Get/Post/Put의 형태로 라즈베리파이 간 Request를 날려주는 모델을 만들었습니다. 또한 10가지 종류의 Sensor나 Actuator들을 모두 유기적으로 연결시켜 각자의 역할을 올바르게 수행하도록 만들었습니다.

결과적으로 저희는 파이 간 통신이나 캐릭터 디바이스들이 의도한대로 잘 동작하는 최종 모델을 만들어내었고, 이는 별다른 소프트웨어의 도움 없이 수업시간에 배웠던 이론들을 응용하여 구현해내었습니다.

하지만, 다음과 같은 2가지 측면에서 처음에 의도했던 것과 다른 방식으로 구현이 되었습니다.

① 라즈베리파이마다 넣는 디바이스들을 시나리오에 맞춰서 구분하여 넣었어야 했는데, 처음에 생각을 잘못하여 Sensor와 Actuator 별로 구분하였습니다. 예를 들어 화재가 발생하는 등의 온도가 급격히 상승하는 긴급한 상황이 발생하였을 때, 중앙 시스템을 거치지 않고 해당 파이에서 빠르게 조치를 취해야 합니다. 그러기 위해서는 하나의 파이에 시나리오에 맞추어 디바이스를 묶어놔야 했는데, 단순히 Sensor와 Actuator 별로 파이를 구분하여 위급 상황에서도 파이 간 통신이 이루어지는 비현실적인 상황이 발생하였습니다.

② Storage 기능을 담당하는 R3에서, mysql과 같은 Database를 설치하여 이를 활용해보려고 하였습니다. 라즈베리파이의 IP와 기타 설정들을 변경해가면서 mysql의 설치는 성공하였지만, Linux에서 arm 명령어로 크로스 컴파일을 진행할 때 mysql 옵션을 같이 넣어주는 방법을 찾지 못하여 어쩔 수 없이 File System을 이용하여 데이터를 저장하게 되었습니다.

제언

저희 팀은 본 프로젝트를 진행하면서 많은 밤샘을 통해 팀원 각자의 역할과 책임감, 협동의 중요성 등을 재확인 할 수 있었습니다. 협업을 보다 효율적으로 하기 위해서 저희는 **'Git'**이라는 툴을 사용하여 쉽게 코드를 관리하였고, **'Slack'**이라는 커뮤니케이션 툴을 통하여 현재 진행 상황을 지속적으로 게시하고 참고할 수 있었습니다.

그 외에도 개별적인 Sensor의 동작들을 연결시켜 여러 Sensor들이 유기적으로 동작하도록 구현하는 방법을 고민함으로써, 프로그래밍 역량 또한 기를 수 있었습니다.

향후 계획

저희가 진행했던 프로젝트는 앞으로의 농업 분야에서 많이 응용될 수 있는 기본적인 작업을 수행합니다. 저희 프로젝트가 수행했던 자동화를 차후 농업 분야에서 응용해 사용하게 된다면, 국내 농업 일손 부족과 같은 문제를 해결하는데 많은 부분 기여 할 수 있을 것으로 기대하고 있습니다.

팀원 개개인적인 측면에서는, 임베디드 소프트웨어 수업에서 터득한 지식과 경험들을 기반으로 더 넓고 더 깊게 알고 있는 IT 인재가 될 수 있도록 노력하겠습니다.