초음파를 이용한 주차 안내 시스템

초음파 센서를 통해 주차 구역의 주차 유무를 감지하여 스마트 폰에 표시하고, RED 또는 GREEN LED를 점등.

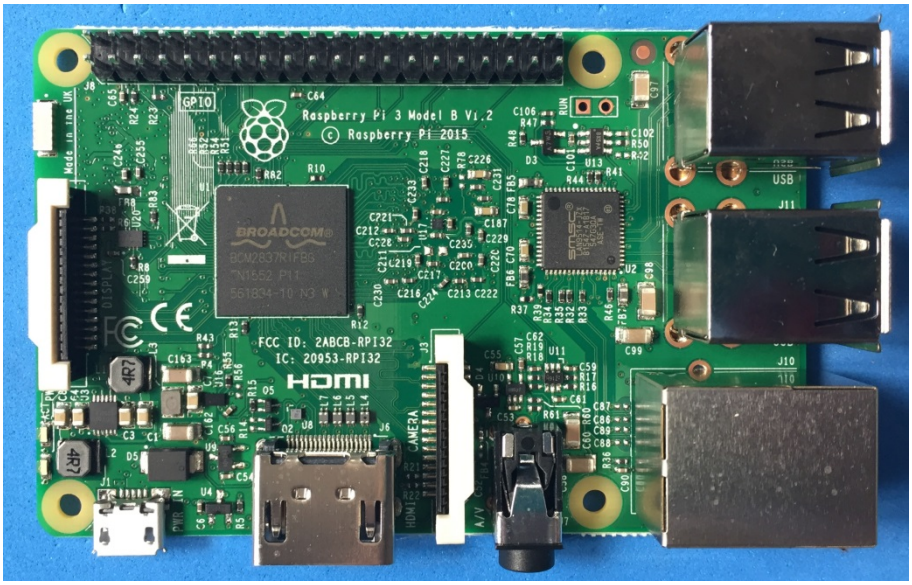
1. User Guide

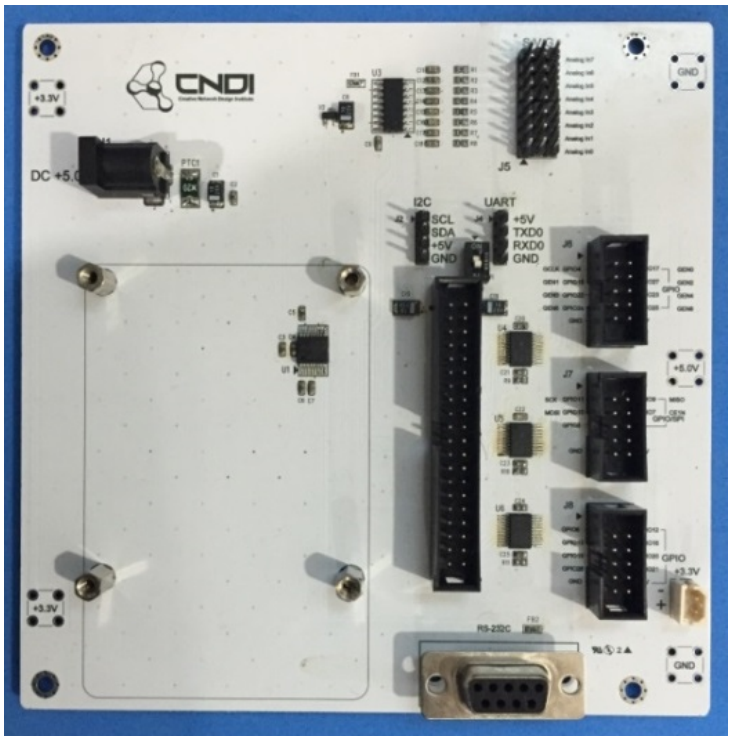
1.1. 사용 모듈 및 결선 방법

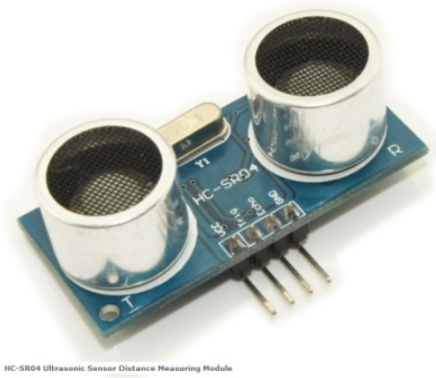
1.1.1. 사용 모듈

이번 캡스톤 디자인에서 사용되는 모듈은 다음과 같다.

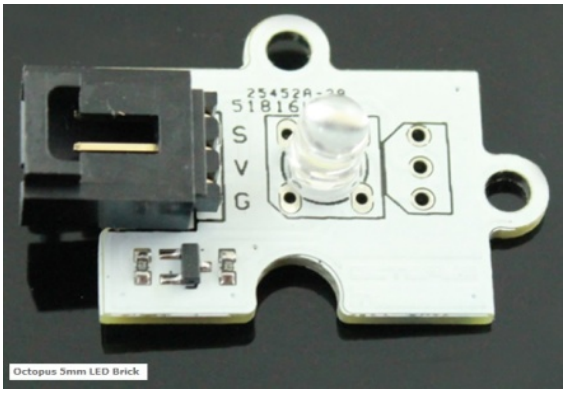
- 라즈베리파이3 모델B
- 라즈베리파이 어댑터 보드
- HC-SR04 Ultrasonic Sensor
- 5mm LED Brick (2개)







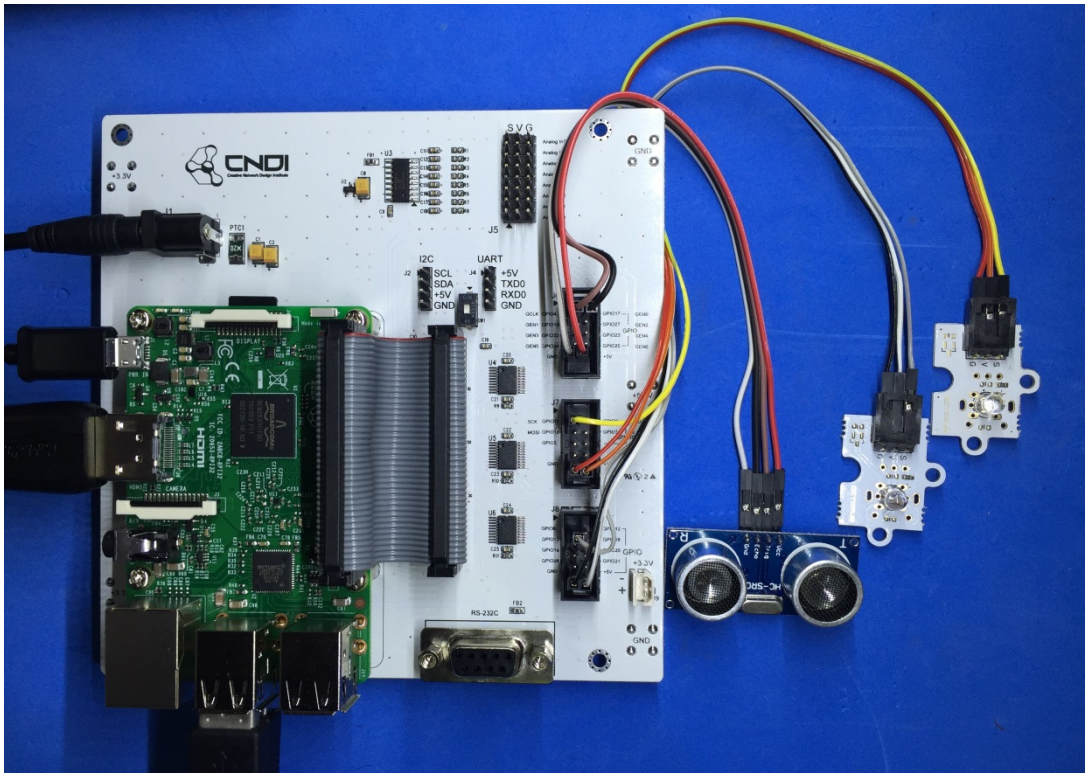
HC-SR04 Ultrasonic Sensor Distance Measuring Module



Octopus 5mm LED Brick

1.1.2. 결선 방법

모듈 별 결선 방법은 다음 그림 및 표와 같다.



Module	HC-SR04	Raspberry Pi Adapter
Pin	VCC	+5V
Pin	Trig	GPIO4
Pin	Echo	GPIO17
Pin	GND	GND

Module	LED Brick(RED)	Raspberry Pi Adapter
Pin	G(GND)	GND
Pin	V(VCC)	+5V
Pin	S(Signal)	GPIO6

Module	LED Brick(GREEN)	Raspberry Pi Adapter
Pin	G(GND)	GND
Pin	V(VCC)	+5V
Pin	S(Signal)	GPIO11

라즈베리파이를 20 x 2 케이블을 이용하여 어댑터 보드와 연결한다.
HC-SR04와 5mm LED Brick은 5V 전원과 연결한다.

모듈의 장치 및 회로에 대한 상세한 내용은 데이터시트 및 회로도를 참고한다.

1.2. 예제프로그램

1.2.1. 라즈베리파이 예제프로그램

본 예제 프로그램은 센서를 이용해 측정된 거리를 실시간으로 확인하여 RED/GREEN LED로 물체(차량) 감지 여부를 확인한다.

초음파 센서의 물체 감지 기준 거리는 100cm 이다. 기준 거리 내에서 물체가 감지되는 경우, Red LED를 점등하고, 밖에서 감지되는 경우, Green LED를 점등한다.

측정된 거리 값은 Wi-Fi를 사용한 TCP/IP 소켓통신을 사용하여 스마트(안드로이드) 폰으로 전송하고 확인한다.

- rasp_server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#include "parking.h"
#include "led.h"

#define BUF_SIZE                100
#define INIT_VAL_OF_MEM         0
#define NUM_OF_ARGC             2
#define PROTOCOL_TYPE           0
```

```
#define NUM_OF_QUEUE		5
#define DATA_SIZE		1

#define PKT_STX 0x01
#define PKT_ETX 0x05

#define CMD_SENSOR_REQ		0x10
#define CMD_SENSOR_RES		0x90

int getData(void);
void error_handling(char *message);

int main(int argc, char *argv[])
{
	int serv_sock, clnt_sock;
	FILE * fp;
	char snd_buf[BUF_SIZE];
	char rcv_buf[BUF_SIZE];
	int read_cnt;
	int readBufSize;

	struct sockaddr_in serv_adr, clnt_adr;
	socklen_t clnt_adr_sz;

	int getData;
	char getData_msb = 0;
	char getData_lsb = 0;

	printf("=====================================₩n");
	printf("TCP/IP Data Transmission Program - Server₩n");
	printf("=====================================₩n₩n");

	// check port info
	if(argc != NUM_OF_ARGC) {
```

```
    printf("Usage: %s <port>₩n", argv[0]);
    exit(1);
}

serv_sock =   socket(PF_INET, SOCK_STREAM, PROTOCOL_TYPE);
if (serv_sock == -1)
        error_handling("socket() error");

memset(&serv_adr, INIT_VAL_OF_MEM, sizeof(serv_adr));

serv_adr.sin_family = AF_INET;
serv_adr.sin_addr.s_addr = htonl(INADDR_ANY);          // get ip address
serv_adr.sin_port = htons(atoi(argv[1]));              // get port num

if (bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr)) == -1)
        error_handling("bind() error");

if (listen(serv_sock, NUM_OF_QUEUE) == -1)
        error_handling("listen() error");

clnt_adr_sz = sizeof(clnt_adr);
clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_adr, &clnt_adr_sz);
printf("Connected IP : %s₩n₩n", inet_ntoa(clnt_adr.sin_addr));

setupWiringPiGpio();
initLed();
initSR04();

while(1)
{
    if(readBufSize = read(clnt_sock, rcv_buf, BUF_SIZE) != -1) {
        if(rcv_buf[0] == PKT_STX) {
            if(rcv_buf[1] == CMD_SENSOR_REQ) {
                if(rcv_buf[4] == PKT_ETX) {
```

```
                                        getData = getSensorData();
                                        controlLed(getData);

                                        getData_lsb = (getData & 0xff);
                                        getData_msb = ((getData>>8) & 0xff);

                                        snd_buf[0] = PKT_STX;
                                        snd_buf[1] = CMD_SENSOR_RES;
                                        snd_buf[2] = getData_lsb;
                                        snd_buf[3] = getData_msb;
                                        snd_buf[4] = PKT_ETX;

                                        write(clnt_sock, snd_buf, BUF_SIZE);
                                }
                        }
                }
          }
          else {
                write(clnt_sock, snd_buf, BUF_SIZE);
                break;
          }
        }
        close(clnt_sock);
        close(serv_sock);
        printf("Transmission terminated.₩n");
        printf("Disconnected(IP : %s)₩n₩n", inet_ntoa(clnt_adr.sin_addr));
        return 0;
}
void error_handling(char *message)
{
        fputs(message, stderr);
        fputc('₩n', stderr);
        exit(1);
}
```

- Parking.h

```
#ifndef __PARKING_H__
#define __PARKING_H__

int setupWiringPiGpio(void);
int getSensorData(void);
int controlLed(int distance);

#endif   /* __PARKING_H__ */
```

- Parking.c

```
#include "parking.h"
#include "hc-sr04.h"
#include "led.h"

int setupWiringPiGpio(void)
{
        if(wiringPiSetupGpio() == -1)
        return 1;
}

int getSensorData(void)
{
        float fDistance;

        initSR04();

        fDistance = controlSR04();
        printf("Distance:%.2fcm₩n", fDistance);

        return fDistance;
}

int g_nLedOnValue = 100;
```

```
int controlLed(int distance)
{
        if(distance < g_nLedOnValue) {
           LedOn(RED);                          //Red On
           LedOff(GREEN);
        }
        else {
           LedOn(GREEN);    //Green On
           LedOff(RED);
        }
}
```

- hc-sr04.h

```
#ifndef __HCSR04_H__
#define __HCSR04_H__

#include <stdio.h>
#include <wiringPi.h>

#define TP          4
#define EP          17

float TpInit(void);
int initSR04(void);
float controlSR04(void);

#endif   /* __HCSR04_H__ */
```

- hc-sr04.c

```
#include "hc-sr04.h"

float TpInit(void)
{
        float fDistance;
```

```
	int nStartTime, nEndTime;

	digitalWrite(TP, LOW);
	delayMicroseconds(2);
	// pull the Trig pin to high level for more than 10us impulse
	digitalWrite(TP, HIGH);
	delayMicroseconds(10);
	digitalWrite(TP, LOW);

	while(digitalRead(EP) == LOW);
	nStartTime = micros();

	while(digitalRead(EP) == HIGH);
	nEndTime = micros();

	fDistance = (nEndTime - nStartTime) / 29. / 2.;

	return fDistance;
}

int initSR04(void)
{
	pinMode(TP,OUTPUT);
	pinMode(EP,INPUT);
	return 0;
}

float controlSR04(void)
{
	float fDistance = TpInit();

	delay(1000);
	return fDistance;
}
```

- led.h

```
#ifndef __LED_H__
#define __LED_H__

#include <wiringPi.h>
#include <stdio.h>

#define LED_RED_PIN		6
#define LED_GREEN_PIN		11

#define RED		1
#define GREEN		0

int initLed(void);
void LedOn(int color);
void LedOff(int color);

#endif	/* __LED_H__ */
```

- led.c

```
#include "led.h"

int initLed(void)
{
	pinMode(LED_RED_PIN, OUTPUT);
	pinMode(LED_GREEN_PIN, OUTPUT);

	digitalWrite(LED_RED_PIN, LOW);
	digitalWrite(LED_GREEN_PIN, LOW);

	return 0;
}
```

```
void LedOn(int color)
{
        if(color == 1) {                // RED
          digitalWrite(LED_RED_PIN, HIGH);
        }
        else{
          digitalWrite(LED_GREEN_PIN, HIGH);
        }
}

void LedOff(int color)
{
        if(color == 1) {                // RED
          digitalWrite(LED_RED_PIN, LOW);
        }
        else {
          digitalWrite(LED_GREEN_PIN, LOW);
        }
}
```

1.2.2. 안드로이드 응용프로그램

- 첨부된 안드로이드 소스 참조

안드로이드 응용프로그램은 라즈베리파이에 장착된 Wi-Fi 무선랜 어댑터를 통하여 라즈베리파이 서버와 연결하고 연결이 성공하면 1초 주기로 센서 값을 요청하는 패킷을 전달하고, 수신한 센서 값을 스마트 폰 UI에 표시한다. 이 때, 안드로이드 앱은 Network Client로 동작한다.(TCP/IP 사용)
첨부된 소스에는 Project Full Source(RaspNetworkClient)와
설치 파일 'ParkingManagement.apk'가 들어있다. apk 파일 설치는 사용하는 스마트 폰에 따라 방법이 다르므로 인터넷에서 자료를 찾아 참고한다.

1.2.3. 실행

- 네트워크 구성

  다음과 같이 네트워크를 구성한다.

현재 프로그램 소스 상의 IP 주소는 임의로 정해져 있으므로 사용하고 있는 AP의 네트워크 구성을 살펴 알맞은 IP 주소를 확인하고, 스마트 폰 프로그램에서 IP 주소를 변경하여 연결한다.

라즈베리 파이에 무선 어댑터를 연결하여 네트워크에 연결한 경우, 다음 명령어를 사용하여 IP 주소를 확인할 수 있다.

```
pi@raspberrypi ~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:b7:87:65
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:165 errors:0 dropped:0 overruns:0 frame:0
          TX packets:165 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:11136 (10.8 KiB)  TX bytes:11136 (10.8 KiB)

wlan0     Link encap:Ethernet  HWaddr 90:9f:33:ea:8a:ff
          inet addr:192.168.10.39  Bcast:192.168.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:54121 errors:0 dropped:25293 overruns:0 frame:0
          TX packets:17035 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8955589 (8.5 MiB)  TX bytes:2917107 (2.7 MiB)

pi@raspberrypi ~ $
```

- 라즈베리파이 프로그램 실행

  라즈베리파이를 실행하여, 무선 네트워크 구성이 완료되면, LXTerminal을 열어 앞서 작성한 예제 프로그램을 실행한다. 실행은 해당 폴더의 실행파일이 있는 위치 다음 명령어를 사용한다.

```
pi@raspberrypi ~ $ cd proj/applications/ParkingManagement/
pi@raspberrypi ~/proj/applications/ParkingManagement $ sudo ./rasp_server 8888
========================================
TCP/IP Data Transmission Program - Server
========================================
```

이렇게 서버가 준비되면, 클라이언트의 연결을 기다리며 대기한다. 안드로이드 프로그램이 실행되고 Connection 버튼을 눌러 Connected 메시지가 표시되며 클라이언트가 연결된다. 네트워크 접속이 원활하지 않은 경우, 연결이 정상적으로 되지 않을 수 있다.

클라이언트가 연결되면 다음과 같이 프로그램이 진행되면서 초음파 센서로 측정한 거리 값을 스마트 폰으로 전송한다.

```
pi@raspberrypi ~/proj/applications/ParkingManagement $ sudo ./rasp_server 8888
========================================
TCP/IP Data Transmission Program - Server
========================================

Connected IP : 192.168.10.33

Distance: 178.50cm
Distance: 86.57cm
Distance: 178.98cm
Distance: 178.95cm
Distance: 178.93cm
Distance: 178.55cm
Distance: 178.93cm
Distance: 178.93cm
Distance: 178.95cm
Distance: 142.45cm
```

- 안드로이드 프로그램 실행

라즈베리파이 프로그램이 실행되고 서버가 연결 대기 상태에 놓이면, 안드로이드 프로그램을 실행한다. 안드로이드 앱은 프로그램과 함께 제공한 "Parking-Management.apk"를 실행하여 스마트 폰에 설치한다.

앱을 실행하면 다음과 같은 화면이 생성되며, Server Setting 버튼을 눌러 사용하고 있는 AP와 연결되는 IP주소를 입력한다.

기존에 설정된 IP 주소를 수정하고 Enter 버튼을 누른다.

IP 주소가 설정되면, Connection 버튼을 눌러 서버와 연결한다.

서버와 연결이 완료되면 Connection 버튼이 비활성화 되면서 Connected 문구가 표시된다. 그리고 Sensing Value에 센서가 측정한 값을 주기적으로(1초) 표시한다.

정상적으로 Connection이 이루어 지지 않는 경우, 네트워크 상태나 프로그램의 정상 작동 여부를 확인한다.