2023년도 졸업 논문

음성인식 기술 기반 IoT 물품 수납함

스텝모터를 이용한 물품 수납함 조종 시스템 개발

한국공학대학교

전자공학부

김 지 민

김 지 헌

박 지 건

신 대 혁

2023년도 졸업 논문

음성인식 기술 기반 IoT 물품 수납함

스텝모터를 이용한 물품 수납함 조종 시스템 개발

Design of IoT storage system based on voice recognition technology

2022 年 12 月 02 日

한국공학대학교

전자공학부

김 지 민

김 지 헌

박 지 건

신 대 혁

음성인식 기술 기반 IoT 물품 수납함

指導敎授 김 준 수 

이 論文을 졸업논문 으로 提出함.

2022 年 12 月 02 日

한국공학대학교

전자공학부

김 지 민
김 지 헌
박 지 건
신 대 혁

목차

제 1 장 서론

제 1 절 제품 개발의 배경

음성인식이란 소리를 입력으로 받는 센서 등을 사용해 얻은 신호 중 사람이 말하는 언어를 컴퓨터로 해석하여 해석한 내용을 데이터로 전환하는 처리 과정을 말하며 STT(Speech-to-Text)라고도 한다. 현재 음성 인식 기술의 발전을 통해 AI 면접 솔루션 서비스, 스마트폰과 같은 다양한 분야에서 사용되고 있으며 사용자의 편의성을 향상하고 있다[1].
이처럼 음성인식 기술이 우리 삶에 편의성을 증진하는 방향에 대해 생각해보던 중 장애인이나 거동이 힘든 노약자들이 필요한 물건을 찾는 과정에서 느낄 수 있는 불편함의 해소를 위해 음성인식 기술을 적용해 찾고 싶은 물건을 손쉽게 찾을 수 있도록 한다면 불편해소에 도움이 될 것으로 생각하였고 더 나아가 공장 등 산업에서의 사용에서 긍정적인 영향을 미칠 수 있으리라 생각되었다.

<그림 1>

보건복지부 자료에 따르면 우리나라의 장애 인구는 약 260만을 넘어서고 있고 그중 대다수는 지체장애인이며 장애인의 수는 계속해서 늘어나는 추세다[2]. 또한 사회의 급속한 고령화로 인해 고령인구의 신체기능 손상 등 다양한 선천적, 후천적 요인으로 인한 장애로 일상생활에서 활동에 제약을 갖게 된 경우가 많다. 단편적인 예로 물품을 수납할 때 위치 파악이 어려운 경우, 몸이 불편해서 수납함을 여닫기가 불편한 경우가 존재한다[3].

또 다른 예시로, 영국 데일리 메일에 따르면 영국의 한 민간 보험사가 영국 성인 남녀 3,000명을 대상으로 시행한 조사에서 대부분 사람은 매일 10분 이상 자신이 원하는 물건을 찾는 데 시간을 소비한다고 조사되었다. 이러한 불편함을 해소하기 위해 도움이 될 수 있는 시스템을 개발하고자 한다.

제 2 절 제품의 목표

앞서 살펴본 제품의 개발 배경을 통해 장애인이나 거동이 불편한 노약자들이 수납함에서 원하는 물건을 손쉽게 얻을 수 있도록 접근성 개선을 위해 음성인식 기술을 접목하는 것을 목표로 한다. 사용자의 명령을 음성으로 주고, 입력받은 물건의 좌표에 해당하는 위치로 모터를 이동시켜 수납함을 여는 시스템을 적용한다. Raspberry pi와 Arduino CNC Shield를 이용해 음성인식을 활용하여 사용자의 명령을 인식하고, 인식한 명령어를 MQTT 통신으로 주고받아 입력된 명령어로 저장된 물체의 위치 데이터를 얻어 XY 플로터의 모터를 제어해 접근하는 방식으로 시스템을 구현한다. 추가로 시스템을 발전시키게 된다면 더 나아가 여러 산업현장 및 개인에게 적용하여 효율적인 재고관리를 가능하게 해 사용자의 편의성을 향상하는 목적의 시스템을 제안하는 것을 목표로 한다.

제 2 장 본론

제 1 절 제품 소개

본 제품은 Arduino를 이용한 음성인식을 활용하여 사용자의 명령을 인식하고 명령어를 기반으로 물체의 위치 데이터를 얻어 음성인식 물품 수납함에 적용한 것이다.
물체의 위치에 관한 데이터나 동작에 관한 정보를 Raspberry Pi를 사용해 처리한 후 모터제어를 위한 통신을 ESP8266 보드로 전송하고 Arduino를 사용해 X, Y, Z축에 해당하는 모터를 제어해 XY 플로터를 이동시켜 물품 수납함에 접근해 원하는 동작을 하도록 설계하였다.

제 2 절 주요 기능

1. 물건 등록
물건 등록 기능은 명령어를 통하여 장치를 동작시킨 후 등록하고 싶은 물건을 몇 번째 칸에 등록시켜 달라고 말하면 Raspberry Pi에서 구글 어시스턴트로 음성을 전송하고 데이터를 받아 물건이 등록되는 기능이다.

2. 물건 삭제
물건 삭제 기능은 해당 물건을 사용하지 않는 경우나 다른 물건을 같은 칸에 저장하기 위해 사용한다. 해당 칸에 있는 물건을 삭제해달라는 요청을 하면 해당 물건에 대한 데이터는 삭제된다.

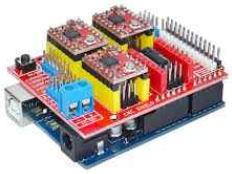
3. 등록된 물건의 위치 변경
등록된 물건의 위치 변경 기능은 해당 칸에 있는 물건을 빈칸으로 위치를 변경할 때 사용한다. 예외적으로 바꾸려고 하는 위치에 다른 물건이 있는 경우에는 변경할 수 없다.

4. 등록된 물건 찾기
등록된 물건 찾기 기능은 수납함에 등록된 물건 중 필요한 물건을 찾을 때 사용한다. 물건 등록 기능을 수행한 후 해당 물건을 찾아달라는 요청을 하게 되면 XY 플로터로 물건의 좌표로 이동하여 물건이 등록된 칸을 열어준다.

제 3 절 제품 설계

1. 하드웨어 설계 및 구현

가. 주요 구성품

품명	XY 플로터	CNC Shield	ESP 8266	Arduino Uno
사진				
용도	원하는 서랍의 위치로 이동	XY 플로터의 스텝모터 제어	MQTT 통신 목적	CNC Shield 제어
품명	SMPS 5A/12V	RASPBERRY PI	마이크	블루투스 스피커
사진				
용도	CNC Shield 전원공급	음성인식 마이크 스피커 제어	음성인식	기능 응답 출력

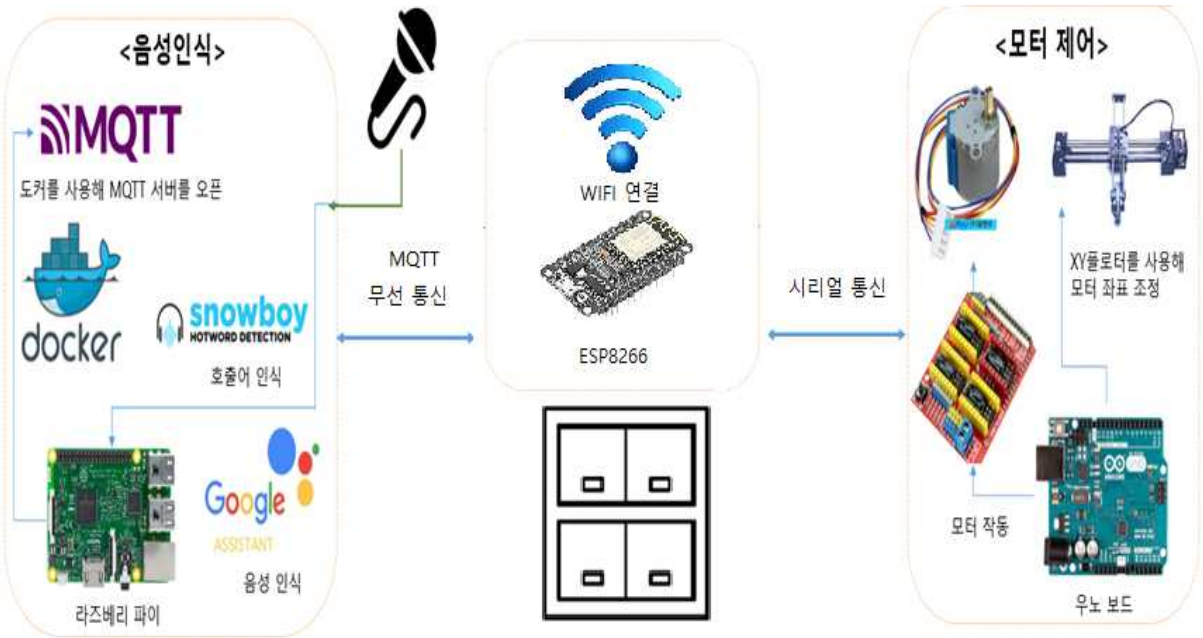
<표 1> 주요 구성품

제품은 크게 2가지로 나뉜다. 음성인식을 담당하는 라즈베리파이 단과 실제 서랍 제어를 담당하는 ESP8266 단으로 구성되며 이 둘은 무선으로 연결된다. 이 이유는 이 둘을 무선으로 연결함으로써 같은 WIFI를 쓸 수 있다면 마이크를 내가 원하는 위치 어디에 위치시켜도 작동할 수 있게 하여 사용자의 편의성을 증가시켜 줄 수 있기 때문이다. 스피커 또한 사용자의 편의성을 증가시켜주기 위해 블루투스 스피커를 선택했고 이 제품에서 이용한 제품은 '구글 홈 미니'이다.

우선 첫 번째 단인 라즈베리파이를 쓴 이유는 음성인식 API인 구글 어시스턴트 API를 쓰기 편할뿐더러 기본적으로 USB 포트와 스피커를 위한 블루투스 모듈을 내장하고 있고 MQTT 통신으로 Docker를 통한 MQTT 서버를 만들기 쉬운 환경이기 때문이다.

두 번째 단인 ESP8266를 쓴 이유는 WIFI 통신이 가능하여 IOT 기술에 자주 쓰이는 MQTT 통신을 할 수 있기 때문이다. 또한 아두이노 우노를 쓴 이유는 CNC Shield가 아두이노 우노와 호환이 쉽게 만들어져 있기 때문이다. 따라서 라즈베리파이에서 ESP8266으로 데이터를 전송하고 ESP8266과 아두이노 우노 사이의 시리얼 통신을 통해 최종적으로 우노에 데이터가 전송된다. 이후 아두이노 우노에서 cnc shield를 제어하여 XY 플로터를 움직일 수 있다.
XY 플로터가 수직 방향으로 움직일 때 중력의 영향을 받으므로 위로 올라가도 다시 떨어지지 않게 스텝모터가 잘 고정되어야 한다. 실험 결과 스텝모터의 자력으로 제품의 무게와 중력을 버틸 수 있는 전압이 12V였고 전원공급을 위해 SMPS 5A/12V 제품을 선택했다.

나. 전체 구조 설계



<그림 2>

2. 소프트웨어 설계 및 구현

가. 상세 동작 알고리즘

1) Docker를 이용해 MQTT 서버를 라즈베리파이에서 열어준 후 ESP8266을 라즈베리파이 MQTT서버에 자동 연결되도록 설정한다.

2) 라즈베리파이에 연결된 마이크를 통해 들어간 음성이 디지털 데이터로 전환한다.

3) 디지털 음성 데이터를 구글 어시스턴트 서버에서 해석 후 라즈베리파이로 돌아온다.

4) 라즈베리파이에서 서랍에 대한 검색, 추가, 삭제에 대한 여러 가지 키워드를 인식한다.

a) 추가 = 라즈베리파이 list.txt 파일에 물건의 이름과 위치가 저장된다.

ex) 가위 두 번째 등록해줘, 가위 세 번째로 바꿔줘(가위가 이미 등록된 경우)

가위 두 번째 추가해줘, 가위 네 번째로 변경해줘(가위가 이미 등록된 경우)

즉, 파일로 저장함으로써 라즈베리파이를 껐다 켜도 위치 정보가 유지되도록 한다.

b) 삭제 = list.txt에서 원하는 물건을 삭제시켜준다.

ex) 가위 삭제해줘, 가위 제거해줘, 가위 없애줘, 가위 지워줘

c) 검색 = list.txt에서 원하는 물건을 찾아준다.

ex) 가위 찾아줘, 가위 어디에 있어.

d) 해당 키워드에 대한 응답을 파이썬 음성 모듈 gTTS를 통해 블루투스로 연결된 스피커로 출력해준다.

5) 추가, 삭제에 대한 키워드는 라즈베리파이 내에서만 변화가 일어나지만, 검색 키워드는 데이터 전송이 필요하다.

a) 라즈베리파이 단계

- 검색 키워드 인식 후 list.txt에 해당 물건이 있는지 확인한다.
- 있다면 해당 좌표를 MQTT 서버로 publish 해 준다.

b) ESP8266 단계

- ESP8266에서는 항상 좌표에 대한 토픽을 Subscribe 해놓고 있어야 한다.
- MQTT를 통해 좌푯값을 전달받는다
- ESP8266에서 우노로 시리얼 통신을 통해 좌푯값을 전송한다.

c) 우노 단계

- 시리얼 통신으로 전송받은 좌푯값을 바탕으로 XY 플로터의 스텝모터를 동작시켜 원하는 위치로 이동한다.
- 목표 좌표의 이동을 마치고 또 다른 스텝모터를 이용해 서랍을 밀어줌으로써 서랍이 자동으로 열리게 된다.

제 4 절 개발 결과 및 작동 시현 테스트

1. 하드웨어 개발 결과

하드웨어 개발 결과 전면	하드웨어 개발 결과 후면
	
<그림 3>	<그림 4>

2. 작동 시현 테스트

해석된 데이터를 기반으로 물건 등록, 삭제, 등록된 위치 변경, 찾기 기능을 수행하기 위한 테스트를 진행하였고 결과는 다음과 같다.

가. 물건을 등록하기 위해서 명령어로 동작시킨 후 가위를 다섯 번째 칸에 저장시킨다. 음성을 인식하여 다섯 번째 칸에 가위가 등록됨을 확인할 수 있다.

나. 그다음 가위라는 단어를 인식하여 가위가 있는 칸을 찾는다. XY 플로터로 좌표를 이동해서 가위가 있는 다섯 번째 칸이 열리게 되며 그 좌표를 알려주는 모습은 <그림3>, <그림4>와 같다.

다. 망치를 다섯 번째 칸에 새로 등록하기 위해 명령어를 통하여 진행한다. 그러나 다섯 번째 칸에 가위가 존재하기 때문에 등록할 수가 없다는 음성이 나온다. 망치를 등록하기 위해서는 이미 다섯 번째 칸에 등록된 가위를 삭제하여야 한다.

라. 다섯 번째 칸에 들어있는 가위를 삭제한다. 가위라는 단어가 삭제되기 때문에 다섯 번째 칸에 망치를 등록할 수 있고 이제 가위라는 명령어를 부르면 인식하지 못한다.

가위가 등록된 칸이 열림(전면)	가위가 등록된 칸이 열림(측면)
	
<그림 5>	<그림 6>

제 3 장 결론

본 논문에서는 음성인식 기술을 사용하여 장애인이나 거동이 불편한 노약자들이 수납함에서 원하는 물건을 손쉽게 얻을 수 있도록 접근성 개선을 위한 IoT 기반 물품 수납함 시스템을 설계하였다. 사용자의 명령을 음성으로 입력받아, 입력받은 명령어를 구글 어시스턴트 서버를 통해 낱말 단위로 해석하고, 해석한 내용 중 명령어를 분석하여 등록, 삭제, 위치 변경, 찾기 기능을 수행한다. 명령어에 따라 데이터베이스 안에 저장된 물건에 해당하는 좌표로 모터를 이동시켜 원하는 물건이 들어있는 수납함을 열어주는 기능을 가진다.

Raspberry Pi를 통해 음성을 출력할 스피커와의 블루투스 연결을 항상 확인하고 자동으로 연결을 시도하며, 물건의 위치를 파일로 관리함으로써 Raspberry Pi가 종료되어도 등록된 물건 데이터가 사라지지 않도록 한다. Raspberry Pi에 연결된 마이크와 시스템 사이에서 MQTT 통신함으로써 마이크와 시스템을 무선으로 연결하고, 또한 사용자의 요청에 대한 응답을 스피커로 출력시킴으로써 제품 사용에 대한 직관성과 편의성을 증가시켜준다. 명령의 종류는 등록, 삭제, 위치 변경, 찾기가 있으며 먼저, 등록은 등록하고 싶은 물체의 이름과 번호를 말하면 등록이 된다. 삭제는 삭제하고 싶은 물체의 이름을 말하고 삭제 명령을 말하면 삭제할 수 있다. 위치 변경은 기존에 등록된 물체의 이름과 명령을 말하고 이동시킬 위치를 말하면 데이터베이스의 위치가 수정된다. 마지막으로, 찾기는 찾으려는 물체를 찾아달라 말하면 해당 데이터베이스에 저장된 물체의 위치로 모터를 이동시킨다. 각 명령어는 모두 데이터베이스와 비교하게 되며 해당 명령을 수행할 수 없는 경우에는 불가능하다는 문구가 나오게 된다.

연구를 통해 대용량 물류창고 및 산업현장과 같은 다수의 부품을 사용하는 공간에서 이용하여 작업의 효율성을 증가시키거나 거동이 불편한 노약자나 장애인들이 더욱 편하게 물건을 수납하는 것에 도움을 줄 수 있을 것으로 기대된다. 하지만 음성인식의 정확도에 대한 문제 개선이 필요하다고 생각한다. 한글의 유사 음소 등 특정 음소를 잘못 인식하는 오류처럼 음성인식의 정확도로 인한 사용자의 요구에 맞지 않는 문제[4]가 발생할 수 있다. 이를 개선하기 위한 추가 연구 방향으로 딥러닝 학습 등을 활용하거나 음성인식 알고리즘의 개선을 통해 음성인식 문제를 개선한다면 더욱 사용자 친화적인 모습의 시스템으로 발전할 수 있을 것으로 전망된다.

참고 문헌

1. 오효진, “스타트업부터 대기업까지, 실생활에 스며든 AI음성인식 기술”, Venturesquare, 2022, https://www.venturesquare.net/864264
2. 보건복지가족부, 장애인 현황, 2021
3. 황준길, Effective Application of Assistive Technology Service to the Person with Physical Disability by Addition of Speech Recognition Function to the Powered Wheelchair, 대구대학교 대학원, 2008
4. 김아름, A Phonological Study on the Error Types of Korean Automatic Speech Recognition, 서울대학교 대학원, 2022

부록

전체 소스코드

esp8266

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <ConfigPortal8266.h>
#include <PubSubClient.h> // mqtt용 헤더
#include <SoftwareSerial.h>
#include <ssd1306.h>

const int RX = 13;
const int TX = 12;
SoftwareSerial swSer(RX,TX);

SSD1306 display(0x3c, 4, 5, GEOMETRY_128_32); // oled

WiFiClient espClient;
PubSubClient client(espClient);
void callback(char* topic, byte* payload, unsigned int length);
int flag;
int count;
char lodding[3] = {'o','x','l'};

char*                ssid_pfix = (char*)"2017146010";
String               user_config_html = "<p><input type='text' name='mqtt'placeholder='MQTT_IP'>";

void setup() {
    Serial.begin(115200);

    display.init(); // 처음 화면 초기화
    display.flipScreenVertically();
    display.setFont(ArialMT_Plain_16);
    display.drawString(0,0,"Captive Portal");
    display.drawString(0,14,"Start"); // 연결중임을 OLED에 표시
    display.display();

    loadConfig();
    // *** If no "config" is found or "config" is not "done", run configDevice ***
    if(!cfg.containsKey("config") || strcmp((const char*)cfg["config"], "done")) {
        configDevice();
    }
    WiFi.mode(WIFI_STA);
    WiFi.begin((const char*)cfg["ssid"], (const char*)cfg["w_pw"]);

    display.clear();
    display.drawString(0,0,(const char*)cfg["ssid"]);
    display.drawString(0,14,"connecting...l"); // 연결중임을 OLED에 표시
    display.display();

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
        display.clear();
        display.drawString(0,0,(const char*)cfg["ssid"]);
        display.drawString(0,14,"connecting..."+String(lodding[count++%3])); // 로딩글자 출력
        display.display();
    }
    // main setup
    Serial.printf("\nIP address : "); Serial.println(WiFi.localIP());
```

```
    client.setServer((const char*)cfg["mqtt"], 1883); // cfg 데이터는 애초에 char*함수가 아닌가보다 (클래스 타입인듯?)
    client.setCallback(callback);

    display.clear();
    display.drawString(0,10,"MQTT Connect.."+String(lodding[count++%3]));
    display.display();

    while (!client.connected()) {
        Serial.println("Connecting to MQTT...");
        if (client.connect("Board")) {
            Serial.println("connected");
            client.subscribe("xy");
        } else {
            Serial.print("failed with state "); Serial.println(client.state());
            display.clear();
            display.drawString(0,10,"MQTT Connect.."+String(lodding[count++%3]));
            display.display();
            delay(1500);
        }
    }

    display.clear();
    display.drawString(0,0,"MQTT connection");
    display.drawString(0,14,"Success!!");
    display.display();

    swSer.begin(9600);
    delay(500);
    Serial.println("ready to chat Board");
}

void loop() {
    client.loop();
    while (!client.connected()) {
        Serial.println("Connecting to MQTT...");
        if (client.connect("Board")) {
            Serial.println("connected");
            client.subscribe("xy");
            display.clear();
            display.drawString(0,0,"MQTT connection");
            display.drawString(0,14,"Success!!");
            display.display();
        } else {
            Serial.print("failed with state "); Serial.println(client.state());
            display.clear();
            display.drawString(0,10,"MQTT Connect.."+String(lodding[count++%3]));
            display.display();
            delay(1500);
        }
    }
}

void callback(char* topic, byte* payload, unsigned int length) {
    char buf[512];
    byte2buff(buf,payload,length); // 길이대로 안하게되면 그 주솟값에 이전 자료들이 남아있어 이상한값이 나오게되기에 길이를 인자로 넣어준다.\
                                   // byte2buff 헤더파일 수정해줘야함 문자에 /n이 들어가도록
    Serial.print("Message arrived in topic: ");
    Serial.println(topic);
    Serial.print("Message:");
```

```
    Serial.println(buf);
    swSer.print(buf);

    String incomming = String(buf);
    int xpos = incomming.indexOf(“,“);
    int zpos = incomming.indexOf(“,“,xpos+1);

    String strxPos = incomming.substring(0,xpos);
    String strzPos = incomming.substring(xpos+1,zpos);

    xpos = strxPos.toInt();
    zpos = strzPos.toInt();

    Serial.print(“xpos = “);
    Serial.println(xpos);
    Serial.print(“zpos = “);
    Serial.println(zpos);
    Serial.println(“------------------------“);
}
```

아두이노 우노

```
#include <Arduino.h>
#include <AccelStepper.h>
#include <MultiStepper.h>
//**x,y좌표**//
//테스트 기본 좌표
//220,200 / 100,200 / 0,200
//220,150 / 100,150 / 0,140
//220,60  / 100,60  / 0,50
//220,0   / 100,0   / 0,0

//위 테스트 좌표 값 기준으로 움직이면 된다.
//***좌표 값 입력 시 시리얼 모니터에 x,z(100,100)입력 후 엔터**//
//좌표 값 입력 시 좌표 값에 3,333,4,444,132,142..등 3과 4가 포함되지 않도록**//



//3단 서랍장은 책상에서 10cm 높이
//xy 모듈 위치는 3단 서랍장 좌/우 모서리 끝과
//xy 모듈의 판의 x0(서랍장 오른쪽 끝), x1(서랍장 왼쪽 끝)과 일치되게 설치

//스탭모터 동시 구동을 위한 라이브러리 셋팅
AccelStepper stepper1(AccelStepper::FULL2WIRE, 2, 5);
AccelStepper stepper2(AccelStepper::FULL2WIRE, 4, 7);
AccelStepper stepper3(AccelStepper::FULL2WIRE, 3, 6);

MultiStepper steppers;
MultiStepper steppers1;

//스탭모터 위치 변수

int xpos,zpos;
String strxPos,strzPos;
int yPos = 6000;
```

```
String incomming;
void setup()
{
  Serial.begin(9600);

  //스탭모터 속도 및 가감속 설정
  stepper1.setMaxSpeed(2000);
  stepper1.setAcceleration(3000.0);
  stepper2.setMaxSpeed(2000);
  stepper2.setAcceleration(3000.0);
  stepper3.setMaxSpeed(2000);
  stepper3.setAcceleration(3000.0);


  steppers.addStepper(stepper1);
  steppers.addStepper(stepper2);
  steppers1.addStepper(stepper3);

  Serial.println("START...");
  delay(1000);
  Serial.println("READY...");
}
void loop()
{
  long positions[2];        //x,z position
  long positions1[2];       //y position

  while(Serial.available())
  {
    //좌표 값 입력을 위한 프로세스
    incomming = Serial.readStringUntil('\n');
    xpos = incomming.indexOf(",");
    zpos = incomming.indexOf(",",xpos+1);

    strxPos = incomming.substring(0,xpos);
    strzPos = incomming.substring(xpos+1,zpos);

    xpos = strxPos.toInt();
    zpos = strzPos.toInt();

    Serial.println(strxPos);
    Serial.println(strzPos);
    Serial.print("x pos: ");
    Serial.println(xpos);
    Serial.print("z pos: ");
    Serial.println(zpos);
```

```
      Serial.println("y pos: ");
      Serial.println(yPos);
      //move x-axis and z-axis
      Serial.println("GO");
      positions[0] = -xpos * gearRatio;
      positions[1] = -zpos * gearRatio;
      steppers.moveTo(positions);
      steppers.runSpeedToPosition();
      delay(500);

      //forward push axis
      Serial.println("Forward");
      positions1[0] = yPos;
      positions1[1] = 0;
      steppers1.moveTo(positions1);
      steppers1.runSpeedToPosition();
      delay(100);

      //backwrad push axis
      Serial.println("Backward");
      positions1[0] = yPos * 0;
      positions1[1] = 0;
      steppers1.moveTo(positions1);
      steppers1.runSpeedToPosition();
      delay(100);

      //move to home x-axis and z-axis
      Serial.println("Back Home");
      positions[0] = 0 * gearRatio;
      positions[1] = 0 * gearRatio;
      steppers.moveTo(positions);
      steppers.runSpeedToPosition();
      delay(500);
    }
    incomming = "";
}
```

라즈베리파이

```
# Copyright (C) 2017 Google Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
```

```
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Sample that implements a gRPC client for the Google Assistant API."""

import concurrent.futures
import json
import logging
import os
import os.path
import pathlib2 as pathlib
import sys
import time
import uuid

import click
import grpc
import google.auth.transport.grpc
import google.auth.transport.requests
import google.oauth2.credentials
import paho.mqtt.client as mqtt

from google.assistant.embedded.v1alpha2 import (
    embedded_assistant_pb2,
    embedded_assistant_pb2_grpc
)
from tenacity import retry, stop_after_attempt, retry_if_exception

try:
    from . import (
        assistant_helpers,
        audio_helpers,
        browser_helpers,
        device_helpers
    )
except (SystemError, ImportError):
    import assistant_helpers
    import audio_helpers
    import browser_helpers
    import device_helpers


ASSISTANT_API_ENDPOINT = 'embeddedassistant.googleapis.com'
END_OF_UTTERANCE = embedded_assistant_pb2.AssistResponse.END_OF_UTTERANCE
DIALOG_FOLLOW_ON = embedded_assistant_pb2.DialogStateOut.DIALOG_FOLLOW_ON
CLOSE_MICROPHONE = embedded_assistant_pb2.DialogStateOut.CLOSE_MICROPHONE
```

```
PLAYING = embedded_assistant_pb2.ScreenOutConfig.PLAYING
DEFAULT_GRPC_DEADLINE = 60 * 3 + 5
#-------------------------------------------------
import RPi.GPIO as g

g.setwarnings(False)
g.setmode(g.BCM)
g.setup(26, g.OUT)
#-------------------------------------------------
from gtts import gTTS

def talk(speech):
    tts = gTTS(text=speech, lang='ko')
    tts.save("ans.mp3")
    os.system("mpg321 ans.mp3")
    print(speech, end='\n\n')
#-------------------------------------------------
find = ["찾아","어디","어딨"]
add = ["추가","등록","변경","바꿔"]
delete = ["제거","지워","없애","삭제"]
answer = ["네","맞아","맞어","맞다","응","어","맞습","정확"]
recogNum = [["열두 번", "십이 번", "12번", "10 두 번","열두번","열 두 번","10 두번"],["열한 번", "십일 번", "11번","열한번","열 한 번"],["열 번", "십 번", "10번","열번"],
            ["아홉 번", "구 번", "9번","아홉번"],["여덟 번", "팔 번", "8번","여덟번"],["일곱 번", "칠 번", "7번","일곱번"],
            ["여섯 번", "육 번", "6번","여섯번"],["다섯 번", "오 번", "5번","다섯번","오번"],["네 번", "사 번", "4번","네번"],
            ["세 번", "삼 번", "3번","세번"],["두 번", "이 번", "2번","두번"],["첫 번", "일 번", "1번","첫번"]] # 12번째 인식 오류 커버
answer_mod = False
thing = ["name","num"] # 대답모드를 위한 버퍼
#-----------------물건 list 함수----------------------
def list_update():
    global things
    try:
        f = open('list.txt','r',encoding='UTF8')
    except IOError:
        f = open("list.txt", 'w', encoding='UTF8')
        f.close() # list.txt 파일이 없는경우 파일을 생성해준다.

    else:
        buf = f.read().split('\n')
        things = {}

        for n in buf:
            if(len(n)): # null인 배열 요소 방지하기 위한것
                n = n.split()
```

```
                things[n[0]] = n[1]
    print(things)
    return

def list_add(text): # 키가 같다면 내용만 바뀌게 된다.
    global answer_mod
    global things
    if answer_mod == False:
        name = text.split()[0]
        flag = 0
        for n in range(len(recogNum)):
            for i in recogNum[n]:
                if i in text:
                    num = len(recogNum) - n
                    flag = 1 # 숫자 인식이 됐다는 의미
                    speech = "물건은 {}이며 {} 번째 서랍 등록 맞나요?".format(name,num)
                    talk(speech)
                    answer_mod = True
                    thing[0] = name
                    thing[1] = num
                    return

        if flag == 0:
            speech = "서랍 용량보다 크거나 원하시는 위치를 인식할 수 없습니다."
            talk(speech)
            return
    else:
        answer_mod = False
        for n in answer:
            if n in text:
                name = thing[0]
                num = thing[1]
                result = []
                result.append(str(x[(num-1)%3]))
                result.append(str(y[(num-1)//3]))
                f = open("list.txt",'w',encoding='UTF8')
                xy = ','.join(result) # xy값을 join을 통해서 문자열로 변환
                for n in things:
                    if things[n] == xy: # 만약 같은 좌표에 물건이 존재한다면
                        speech = "해당 위치에는 "+n+" 물건이 존재합니다."
                        talk(speech)
                        return
                things[name] = xy
                for n in things:
                    f.write(n + ' '+things[n] + '\n')
                f.close()
                list_update()
```

```
                speech = str(num)+"번째 "+name+" 추가 완료."
                talk(speech)
                return
        speech = "긍정적 대답이 아니므로 등록을 취소합니다."
        talk(speech)
        return

def list_delete(name):
    global things
    #name = input("제거 물건 이름 입력 : ")
    if name in things:
        del things[name]
        f = open('list.txt','w',encoding='UTF8')
        for n in things:
            f.write(n + ' '+things[n] + '\n')
        f.close()
        list_update()
        speech = name+" 제거 완료."
        talk(speech)
    else:
       speech=name + " 물건은 없습니다."
       talk(speech)

def list_find(name): # text 전체를 볼것인지 맨 처음 단어만 볼것인지 고민
    global things
    if name in things:
        print('물건 :', name, '\n좌표 :', things[name])
        print(client.publish("xy", things[name]))
        locate = things[name].split(',')
        speech = "{} 좌표는 {}, {}입니다.".format(name, locate[0], locate[1])
        talk(speech)
    else:
        speech = name + " 물건은 없습니다."
        talk(speech)


x = [220, 100, 0]
y = [210, 140, 60, 0]

things = {}
list_update()
#------------------ mqtt통신-----------------------
server = "127.0.0.1"

def on_connect(client, userdata, flags, rc):
    print("Connected with RC : " + str(rc))
```

```
client = mqtt.Client()
client.connect(server, 1883, 60)
client.on_connect = on_connect

client.loop_start()
#-------------------------------------------------
import snowboydecoder
import signal

interrupted = False


def signal_handler(signal, frame):
    global interrupted
    interrupted = True


def interrupt_callback():
    global interrupted
    return interrupted

signal.signal(signal.SIGINT, signal_handler)
detector = snowboydecoder.HotwordDetector("resources/models/jarvis.umdl",sensitivity=[0.8,0.80],
audio_gain=5,apply_frontend=True)
#-------------------------------------------------



class SampleAssistant(object):
    """Sample Assistant that supports conversations and device actions.
    Args:
      device_model_id: identifier of the device model.
      device_id: identifier of the registered device instance.
      conversation_stream(ConversationStream): audio stream
        for recording query and playing back assistant answer.
      channel: authorized gRPC channel for connection to the
        Google Assistant API.
      deadline_sec: gRPC deadline in seconds for Google Assistant API call.
      device_handler: callback for device actions.
    """

    def __init__(self, language_code, device_model_id, device_id,
                 conversation_stream, display,
                 channel, deadline_sec, device_handler):
        self.language_code = language_code
        self.device_model_id = device_model_id
        self.device_id = device_id
```

```
        self.conversation_stream = conversation_stream
        self.display = display

        # Opaque blob provided in AssistResponse that,
        # when provided in a follow-up AssistRequest,
        # gives the Assistant a context marker within the current state
        # of the multi-Assist()-RPC "conversation".
        # This value, along with MicrophoneMode, supports a more natural
        # "conversation" with the Assistant.
        self.conversation_state = None
        # Force reset of first conversation.
        self.is_new_conversation = True

        # Create Google Assistant API gRPC client.
        self.assistant = embedded_assistant_pb2_grpc.EmbeddedAssistantStub(
            channel
        )
        self.deadline = deadline_sec

        self.device_handler = device_handler

    def __enter__(self):
        return self

    def __exit__(self, etype, e, traceback):
        if e:
            return False
        self.conversation_stream.close()

    def is_grpc_error_unavailable(e):
        is_grpc_error = isinstance(e, grpc.RpcError)
        if is_grpc_error and (e.code() == grpc.StatusCode.UNAVAILABLE):
            logging.error('grpc unavailable error: %s', e)
            return True
        return False

    @retry(reraise=True, stop=stop_after_attempt(3),
           retry=retry_if_exception(is_grpc_error_unavailable))
    def assist(self):
        """Send a voice request to the Assistant and playback the response.
        Returns: True if conversation should continue.
        """
        continue_conversation = False
        device_actions_futures = []

        self.conversation_stream.start_recording()
        logging.info('Recording audio request.')
```

```
def iter_log_assist_requests():
    for c in self.gen_assist_requests():
        assistant_helpers.log_assist_request_without_audio(c)
        yield c
    logging.debug('Reached end of AssistRequest iteration.')

# This generator yields AssistResponse proto messages
# received from the gRPC Google Assistant API.
for resp in self.assistant.Assist(iter_log_assist_requests(),
                                  self.deadline):
    assistant_helpers.log_assist_response_without_audio(resp)
    if resp.event_type == END_OF_UTTERANCE:
        logging.info('End of audio request detected.')
        logging.info('Stopping recording.')
        self.conversation_stream.stop_recording()
        text = ' '.join(r.transcript for r in resp.speech_results)
        print(text)
    if resp.speech_results:
        logging.info('Transcript of user request: "%s".',
                     ' '.join(r.transcript
                              for r in resp.speech_results))
        text = ' '.join(r.transcript for r in resp.speech_results)
    if len(resp.audio_out.audio_data) > 0:
        if not self.conversation_stream.playing:
            self.conversation_stream.stop_recording()
            self.conversation_stream.start_playback()
            logging.info('Playing assistant response.')
            print("결과 :",text)
            if "종료" in text:
                print('프로그램 종료')
                sys.exit()

            if answer_mod: # 추가에 대한 응답모드
                list_add(text)
                return

            for keyword in find:
                if keyword in text:
                    list_find(text.split()[0])
                    return

            for keyword in add:
                if keyword in text:
                    list_add(text)
                    return
```

```
                    for keyword in delete:
                        if keyword in text:
                            list_delete(text.split()[0])
                            return
                self.conversation_stream.write(resp.audio_out.audio_data)
            if resp.dialog_state_out.conversation_state:
                conversation_state = resp.dialog_state_out.conversation_state
                logging.debug('Updating conversation state.')
                self.conversation_state = conversation_state
            if resp.dialog_state_out.volume_percentage != 0:
                volume_percentage = resp.dialog_state_out.volume_percentage
                logging.info('Setting volume to %s%%', volume_percentage)
                self.conversation_stream.volume_percentage = volume_percentage
            if resp.dialog_state_out.microphone_mode == DIALOG_FOLLOW_ON:
                continue_conversation = True
                logging.info('Expecting follow-on query from user.')
            elif resp.dialog_state_out.microphone_mode == CLOSE_MICROPHONE:
                continue_conversation = False
            if resp.device_action.device_request_json:
                device_request = json.loads(
                    resp.device_action.device_request_json
                )
                fs = self.device_handler(device_request)
                if fs:
                    device_actions_futures.extend(fs)
            if self.display and resp.screen_out.data:
                system_browser = browser_helpers.system_browser
                system_browser.display(resp.screen_out.data)

        if len(device_actions_futures):
            logging.info('Waiting for device executions to complete.')
            concurrent.futures.wait(device_actions_futures)

        logging.info('Finished playing assistant response.')
        self.conversation_stream.stop_playback()
        return text

    def gen_assist_requests(self):
        """Yields: AssistRequest messages to send to the API."""

        config = embedded_assistant_pb2.AssistConfig(
            audio_in_config=embedded_assistant_pb2.AudioInConfig(
                encoding='LINEAR16',
                sample_rate_hertz=self.conversation_stream.sample_rate,
            ),
            audio_out_config=embedded_assistant_pb2.AudioOutConfig(
                encoding='LINEAR16',
```

```
                    sample_rate_hertz=self.conversation_stream.sample_rate,
                    volume_percentage=self.conversation_stream.volume_percentage,
                ),
                dialog_state_in=embedded_assistant_pb2.DialogStateIn(
                    language_code=self.language_code,
                    conversation_state=self.conversation_state,
                    is_new_conversation=self.is_new_conversation,
                ),
                device_config=embedded_assistant_pb2.DeviceConfig(
                    device_id=self.device_id,
                    device_model_id=self.device_model_id,
                )
            )
            if self.display:
                config.screen_out_config.screen_mode = PLAYING
            # Continue current conversation with later requests.
            self.is_new_conversation = False
            # The first AssistRequest must contain the AssistConfig
            # and no audio data.
            yield embedded_assistant_pb2.AssistRequest(config=config)
            for data in self.conversation_stream:
                # Subsequent requests need audio data, but not config.
                yield embedded_assistant_pb2.AssistRequest(audio_in=data)


@click.command()
@click.option('--api-endpoint', default=ASSISTANT_API_ENDPOINT,
              metavar='<api endpoint>', show_default=True,
              help='Address of Google Assistant API service.')
@click.option('--credentials',
              metavar='<credentials>', show_default=True,
              default=os.path.join(click.get_app_dir('google-oauthlib-tool'),
                                   'credentials.json'),
              help='Path to read OAuth2 credentials.')
@click.option('--project-id',
              metavar='<project id>',
              help=('Google Developer Project ID used for registration '
                    'if --device-id is not specified'))
@click.option('--device-model-id',
              metavar='<device model id>',
              help=(('Unique device model identifier, '
                     'if not specifed, it is read from --device-config')))
@click.option('--device-id',
              metavar='<device id>',
              help=(('Unique registered device instance identifier, '
                     'if not specified, it is read from --device-config, '
                     'if no device_config found: a new device is registered '
```

```
                    'using a unique id and a new device config is saved')))
@click.option('--device-config', show_default=True,
              metavar='<device config>',
              default=os.path.join(
                  click.get_app_dir('googlesamples-assistant'),
                  'device_config.json'),
              help='Path to save and restore the device configuration')
@click.option('--lang', show_default=True,
              metavar='<language code>',
              default='ko-KR',
              help='Language code of the Assistant')
@click.option('--display', is_flag=True, default=False,
              help='Enable visual display of Assistant responses in HTML.')
@click.option('--verbose', '-v', is_flag=True, default=False,
              help='Verbose logging.')
@click.option('--input-audio-file', '-i',
              metavar='<input file>',
              help='Path to input audio file. '
              'If missing, uses audio capture')
@click.option('--output-audio-file', '-o',
              metavar='<output file>',
              help='Path to output audio file. '
              'If missing, uses audio playback')
@click.option('--audio-sample-rate',
              default=audio_helpers.DEFAULT_AUDIO_SAMPLE_RATE,
              metavar='<audio sample rate>', show_default=True,
              help='Audio sample rate in hertz.')
@click.option('--audio-sample-width',
              default=audio_helpers.DEFAULT_AUDIO_SAMPLE_WIDTH,
              metavar='<audio sample width>', show_default=True,
              help='Audio sample width in bytes.')
@click.option('--audio-iter-size',
              default=audio_helpers.DEFAULT_AUDIO_ITER_SIZE,
              metavar='<audio iter size>', show_default=True,
              help='Size of each read during audio stream iteration in bytes.')
@click.option('--audio-block-size',
              default=audio_helpers.DEFAULT_AUDIO_DEVICE_BLOCK_SIZE,
              metavar='<audio block size>', show_default=True,
              help=('Block size in bytes for each audio device '
                    'read and write operation.'))
@click.option('--audio-flush-size',
              default=audio_helpers.DEFAULT_AUDIO_DEVICE_FLUSH_SIZE,
              metavar='<audio flush size>', show_default=True,
              help=('Size of silence data in bytes written '
                    'during flush operation'))
@click.option('--grpc-deadline', default=DEFAULT_GRPC_DEADLINE,
              metavar='<grpc deadline>', show_default=True,
```

```
                help='gRPC deadline in seconds')
@click.option('--once', default=False, is_flag=True,
                help='Force termination after a single conversation.')
def main(api_endpoint, credentials, project_id,
         device_model_id, device_id, device_config,
         lang, display, verbose,
         input_audio_file, output_audio_file,
         audio_sample_rate, audio_sample_width,
         audio_iter_size, audio_block_size, audio_flush_size,
         grpc_deadline, once, *args, **kwargs):
    """Samples for the Google Assistant API.
    Examples:
      Run the sample with microphone input and speaker output:
        $ python -m googlesamples.assistant
      Run the sample with file input and speaker output:
        $ python -m googlesamples.assistant -i <input file>
      Run the sample with file input and output:
        $ python -m googlesamples.assistant -i <input file> -o <output file>
    """
    # Setup logging.
    logging.basicConfig(level=logging.DEBUG if verbose else logging.INFO)

    # Load OAuth 2.0 credentials.
    try:
        with open(credentials, 'r') as f:
            credentials = google.oauth2.credentials.Credentials(token=None,
                                                                **json.load(f))
            http_request = google.auth.transport.requests.Request()
            credentials.refresh(http_request)
    except Exception as e:
        logging.error('Error loading credentials: %s', e)
        logging.error('Run google-oauthlib-tool to initialize '
                      'new OAuth 2.0 credentials.')
        sys.exit(-1)

    # Create an authorized gRPC channel.
    grpc_channel = google.auth.transport.grpc.secure_authorized_channel(
        credentials, http_request, api_endpoint)
    logging.info('Connecting to %s', api_endpoint)

    # Configure audio source and sink.
    while 1:
        if answer_mod == False:
            print('Listening... Press Ctrl+C to exit')
            g.output(26, g.HIGH)

            # main loop
```

```
        detector.start(detected_callback=snowboydecoder.play_audio_file,
                       interrupt_check=interrupt_callback,
                       sleep_time=0.03)

        detector.terminate()

    audio_device = None
    if input_audio_file:
        audio_source = audio_helpers.WaveSource(
            open(input_audio_file, 'rb'),
            sample_rate=audio_sample_rate,
            sample_width=audio_sample_width
        )
    else:
        audio_source = audio_device = (
            audio_device or audio_helpers.SoundDeviceStream(
                sample_rate=audio_sample_rate,
                sample_width=audio_sample_width,
                block_size=audio_block_size,
                flush_size=audio_flush_size
            )
        )
    if output_audio_file:
        audio_sink = audio_helpers.WaveSink(
            open(output_audio_file, 'wb'),
            sample_rate=audio_sample_rate,
            sample_width=audio_sample_width
        )
    else:
        audio_sink = audio_device = (
            audio_device or audio_helpers.SoundDeviceStream(
                sample_rate=audio_sample_rate,
                sample_width=audio_sample_width,
                block_size=audio_block_size,
                flush_size=audio_flush_size
            )
        )
    # Create conversation stream with the given audio source and sink.
    conversation_stream = audio_helpers.ConversationStream(
        source=audio_source,
        sink=audio_sink,
        iter_size=audio_iter_size,
        sample_width=audio_sample_width,
    )

    if not device_id or not device_model_id:
        try:
```

```
            with open(device_config) as f:
                device = json.load(f)
                device_id = device['id']
                device_model_id = device['model_id']
                logging.info("Using device model %s and device id %s",
                             device_model_id,
                             device_id)
        except Exception as e:
            logging.warning('Device config not found: %s' % e)
            logging.info('Registering device')
            if not device_model_id:
                logging.error('Option --device-model-id required '
                              'when registering a device instance.')
                sys.exit(-1)
            if not project_id:
                logging.error('Option --project-id required '
                              'when registering a device instance.')
                sys.exit(-1)
            device_base_url = (
                'https://%s/v1alpha2/projects/%s/devices' % (api_endpoint,
                                                             project_id)
            )
            device_id = str(uuid.uuid1())
            payload = {
                'id': device_id,
                'model_id': device_model_id,
                'client_type': 'SDK_SERVICE'
            }
            session = google.auth.transport.requests.AuthorizedSession(
                credentials
            )
            r = session.post(device_base_url, data=json.dumps(payload))
            if r.status_code != 200:
                logging.error('Failed to register device: %s', r.text)
                sys.exit(-1)
            logging.info('Device registered: %s', device_id)
            pathlib.Path(os.path.dirname(device_config)).mkdir(exist_ok=True)
            with open(device_config, 'w') as f:
                json.dump(payload, f)

    device_handler = device_helpers.DeviceRequestHandler(device_id)

    @device_handler.command('action.devices.commands.OnOff')
    def onoff(on):
        if on:
            logging.info('Turning device on')
        else:
```

```
                logging.info('Turning device off')

        @device_handler.command('com.example.commands.BlinkLight')
        def blink(speed, number):
            logging.info('Blinking device %s times.' % number)
            delay = 1
            if speed == "SLOWLY":
                delay = 2
            elif speed == "QUICKLY":
                delay = 0.5
            for i in range(int(number)):
                logging.info('Device is blinking.')
                time.sleep(delay)

        with SampleAssistant(lang, device_model_id, device_id,
                             conversation_stream, display,
                             grpc_channel, grpc_deadline,
                             device_handler) as assistant:
            # If file arguments are supplied:
            # exit after the first turn of the conversation.
            if input_audio_file or output_audio_file:
                assistant.assist()
                return

            # If no file arguments supplied:
            # keep recording voice requests using the microphone
            # and playing back assistant response using the speaker.
            # When the once flag is set, don't wait for a trigger. Otherwise, wait.
            wait_for_user_trigger = not once
            os.system("aplay resources/ding.wav &") # 딩동 소리 시점 바꿈 (스노우보이 디코더 수정)
            print("음성인식 시작")
            g.output(26, g.LOW) # LED가 꺼짐으로써 음성인식 시작
            text = assistant.assist()
            #conversation_stream.close()

if __name__ == '__main__':
    main()
```