시스템프로그래밍및실습

팀 프로젝트 최종 보고서

<Sign Translation Machine>

2024년 06월 20일

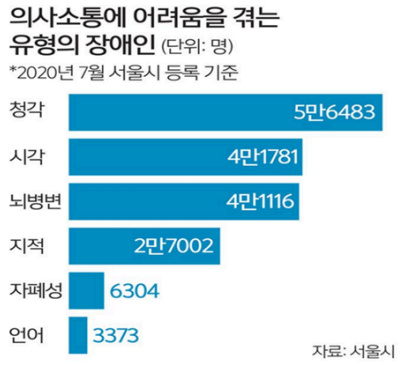
7조	
202020354	정용훈
202020820	남현원
202128683	이지수
201920713	전 진

보고서 목차

I. 주제 및 선정 이유(Motivation)

1. 프로젝트의 필요성 및 차별성

수화를 실시간으로 한글 번역해주는 스마트 장갑을 제작하고자 하였다.



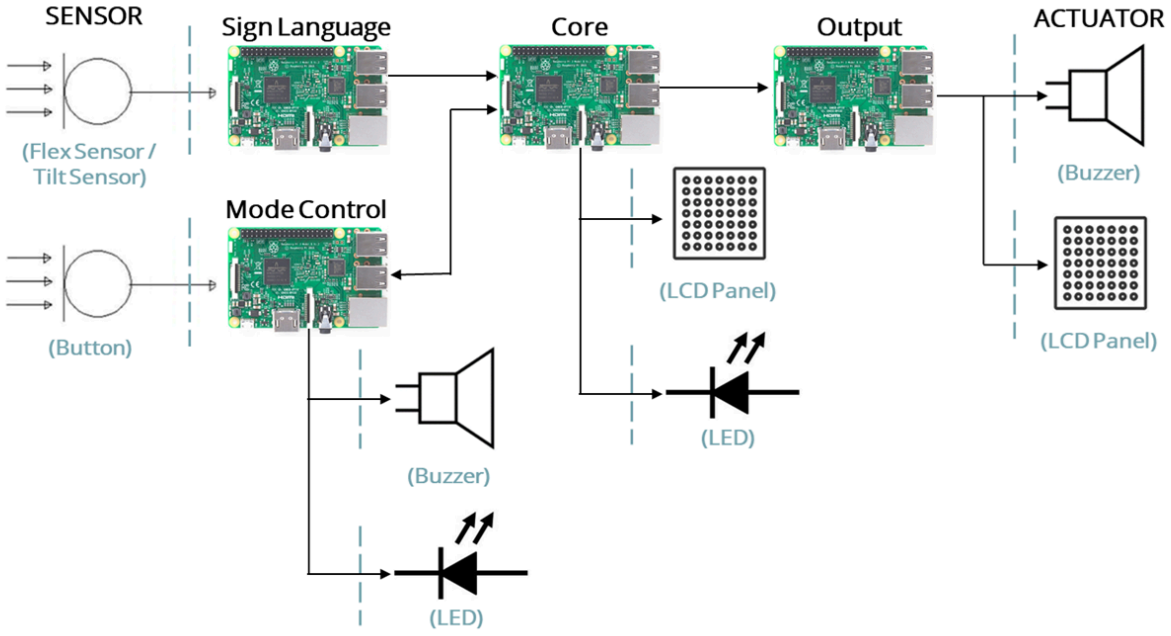
수화가 있음에도 의사소통에 불편함을 겪는 장애인 분들이 많아 청각 장애인과 비장애인 간의 소통을 개선하고, 수화가 필요한 사람들이 다른 사람의 번역이나 많은 학습 시간이 요구되는 수화 배움을 필요로 하지 않고 사람들과 직접 의사소통할 수 있는 쉬운 방법을 제공하고자 하였다. 또한 한국인을 대상으로 한 수화 장갑을 타겟팅하여 한글의 특징적인 초, 중, 종성의 자음과 모음을 조합하여 글자를 나타냄으로써 기존적 영어기반이 아닌 한글 수화를 다루는 차별적 프로젝트를 진행하고자 하였다.

2. 프로젝트의 목표

본 프로젝트의 최종 목표는 일상 속 청각 장애인과 비장애인 간의 소통 개선이다. 다양한 센서를 통해 손의 움직임, 손가락의 굽힘정도 등을 감지하여 수화 동작을 정확하게 인식하고, 인식된 수화를 실시간으로 한글 번역하여 한글의 특징적인 자음과 모음이 올바르게 위치하도록 하며, 사용자간의 원활한 의사소통을 가능하게 하고자 한다. 이는 곧 청각장애인분들의 사회적 참여와 독립성을 증진키는데 크게 기여할 수 있다고 생각한다.

II. 시스템 개발 개요

1. 프로젝트 시스템 구조



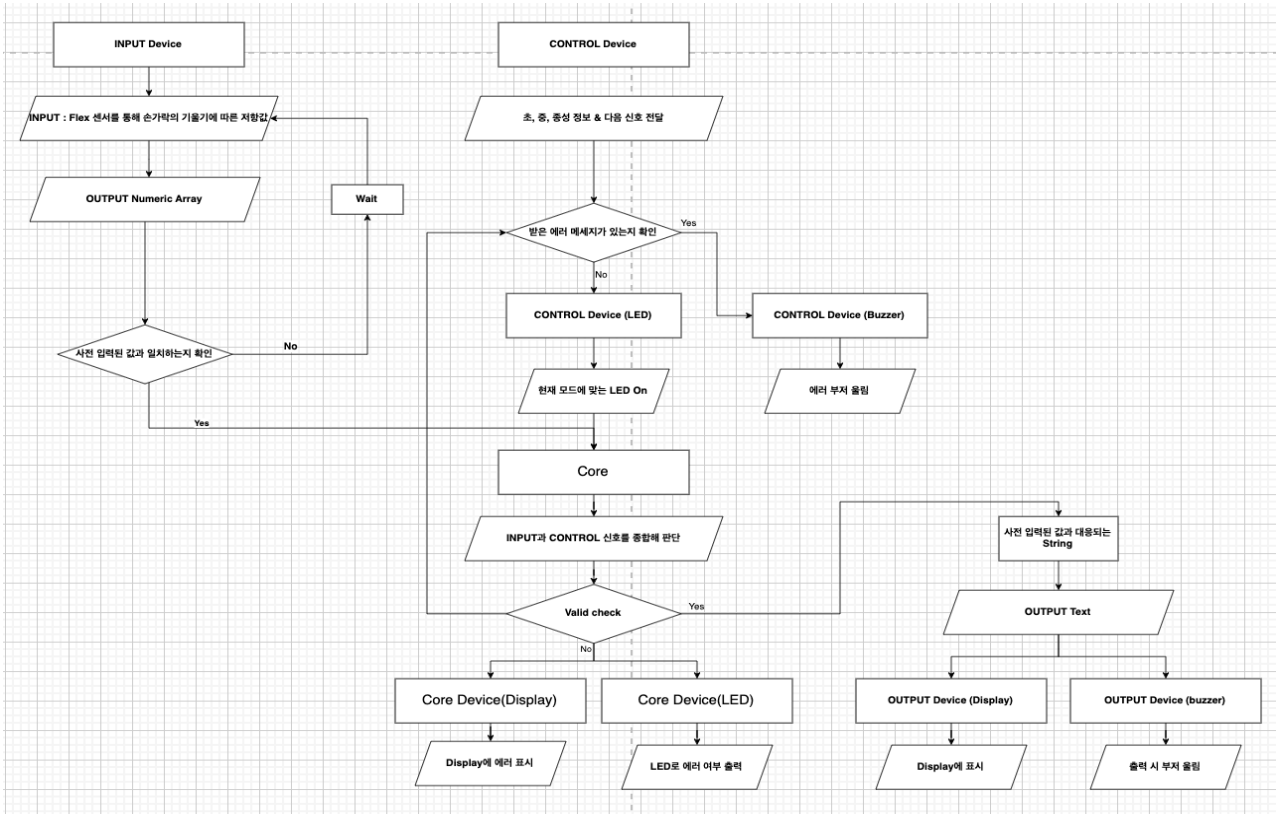
[그림1] 수화 한글 번역 장갑 전체 구조

본 프로젝트는 목표하는 기능을 구현하기 위해 input을 받는 장갑과 그리고 입력을 한글로 번역하는 output과 그리고 사용자가 장갑을 컨트롤 할 수 있게 하는 control 부분과 전체 입력을 한글에 맵핑하고 아웃풋 부분에 보내주는 역할을 하는 core부분에 라즈베리파이를 분배하였다. [그림1]은 수화 한글 번역 장갑의 전체 구조를 보여주고 있다. [표1]는 분배된 라즈베리파이가 구현하는 주요 기능을 알려주고 있다.

Glove	Sign Translation Glove를 통한 수화 입력 확인
Mode Control	초성, 중성, 종성 위치 구별 입력 완료 버튼 에러 상황 확인
Core	에러 감지 입력, 한글 맵핑(binary signal -> Integer signal)
Output	한글 출력 지원(Integer signal -> LCD) 입력 감지

[표1] 각 라즈베리파이의 주요 기능

2. 프로젝트 시스템 동작과정



[그림2] 시스템 동작 FLOW

프로젝트의 시스템 동작과정은 위에 있는 [그림2]와 같다. 시스템의 입력부는 Glove Pi와 Control Pi로, Glove Pi 에서는 {Flex sensor 5개 + 기울기 sensor} 를 통해 2진수 6자리 신호를 만들어낸다. 이때 각 자리는 손가락1번~5번 + 기울어짐 여부로 011001과 같이 만들어진다. Glove Pi에서는 해당 신호를 Core Pi로 계속해서 전송한다. 입력부에 해당하는 Control Pi는 한글의 특수성 때문에 장갑 외에 추가한 부분이다. 한글의 경우 'ㄱ' 또는 'ㅏ'와 같이 여러 자음과 모음의 조합으로 한 글자를 표현한다. 따라서 수화로 표현한 한 자음 또는 모음이 초성, 중성, 종성 중 어디에 해당하는지 구분할 필요가 있다. 그래서 Control Pi에서는 3가지 모드(초성,중성,종성)을 버튼으로 구분하여 해당 정보를 Core로 넘겨준다. Glove Pi에서 계속해서 신호를 전송하면, Control에서 초성모드 버튼이 눌리면 Core에서는 지금 Glove상태가 초성을 의미함을 인지할 수 있다. 또한 Control에서 Next 버튼을 누르면 해당 신호가 Core로 전달되고 Core에서는 현재까지 입력받은 정보가 에러가 없다면 Output Pi로 전달하여 한글로 LCD에 출력한다. Core Pi에서는 Control에서 받은 신호의 에러 여부를 판단해 Control Pi로 에러신호를 보내주거나 또는 Core 처리에 에러가 있다면 Core Pi의 LCD에 에러 메세지를 띄운다. Output Pi에서는 Core Pi로부터 6자리 Integer를 전달받는다. 전달받은 값은 2자리씩 끊어, 초성, 중성, 종성에 해당한다. 예를들어 010101과 같이 전달받았다면 01->'ㄱ', 01->'ㅏ', 01->'ㄱ'을 의미하며 LCD에 '각'을 표시한다. LCD는 가로세로 2x2를 사용해 한 글자를 표현하는데 'ㅜ'와 같이 가로로 배치되는 모음의 경우 한글 매핑을 따로 해야하기에 세로모음과 가로모음을 구분하여 정의했다. 가로모음은 자음과 조합되어 한 칸에 표시해야하므로 가로모음은 초성이 00, 중성이 가로모음의 식별자를 의미하게 된다. Core로부터 신호를 전달받으면 LCD에 표시되었음을 알리는 부저가 동작하여 신호음을 낸다

3. 전체 알고리즘

□ RPI 1: Glove

주요 사용 라이브러리는 플렉스 센서의 아날로그 데이터를 읽기 위한 bcm2835 SPI 라이브러리와 스레딩을위한 pthread, 네트워크 통신을 위한 socket 라이브러리를 사용하였다. inclination()함수로 기울기 값을 읽어 저장하였고, read_adc() 함수로 각 플렉스 센서의 ADC값을 읽어 저장하였다. 특히, bcm_spi_transfernb의 tx에서 배열 데이터를 전송하고, rx에서 수신된 데이터를 저장하고자 하였다. void *flex_senor(void*arg) 의 int flexADC0=read_adc(0) 함수에서 연결한 첫번째 flex sensor값(엄지)을 읽고, 굽힘에 따라 0~1024정도까지 다양하게 뜨는 센서 값에서 확실하게 구부릴 경우 1000이상으로 고정 출력하는 것에 따라 if(flexADC0 >=1000)의 if문 함수를 사용하여 구부릴 경우 1, 아닐 경우 0으로 이진화 하여 값을 변수 a에 저장하였다. 모든 손가락에서 데이터를 받음으로 총 5개의 if문을 사용하여 해당 값을 저장하였고, 수화를 2진화 매핑한 테이블에서 저항값, 구부림만으로는 구분이 불가능한 자음, 모음은 기울기 센서의 값을 추가로 받아 구분이 필요하기 때문에 이를 위해 if(a==0 && b==0 && c==1 && d==1 && e==1)와 같이 함수를 추가하여 특정 구부림과 기울기를 만족하였을때 f값을 받아 반환하도록 하였다. 또한, bcm2835_delay(500)함수를 통해 500ms, 0.5초마다 센서 값을 읽어내도록 하였고, while문을 통해 중지할때까지 위 작업을 계속 수행하고자 하였다. 따라서 데이터를 전송하기 위해 생성한 스레드 flex_sensor함수에서 이진화된 a-f 데이터값을 socket를 통해 최종적으로 core에 전송하였다.

□ RPI 2 : Control

서버로부터 에러를 확인하여 부저를 울리게 하는 함수인 get_error_code와 이용자가 버튼을 이용하여 입력 위치를 바꿀 수 있게 하고, 한 글자에 대한 입력 완료에 대한 버튼이 구현하여 서버에게 신호를 보내는 함수인 change_place 함수는 모두 스레드 단위로 실행된다. 부저는 PWM을 사용하여 라이브러리 사용없이 구현하였고, LED와 버튼은 모두 GPIO를 라이브러리 사용없이 직접 구현하였다.

먼저 서버로부터 에러코드를 확인하는 부분에서는 read()함수를 사용하여 메시지를 받는다. 받은 메시지가 에러코드 1을 의미하면 부저가 한 번 울리게 하고 만약 에러코드 2를 의미하면 부저가 2번울리게 하여 에러가 소리를 통해 무슨 상황인지 알 수 있게 한다.

그리고 버튼을 2개를 만들어서 이용자가 지금input 하려는 위치가 초성, 중성, 종성인지 조절할 수 있게하고, RGB LED를 사용하여 사용자가 입력하는 위치를 까먹지 않을 수 있게 한다. 버튼이 눌릴 때마다 사용자가 원하는 위치가 어디인지 write()함수를 사용하여 서버에 메시지를 보낸다.

빨간색 : 초성 입력할 타임 GPIOWrite(RED, 1);

초록색 : 중성 입력할 타임 GPIOWrite(GREEN, 0);

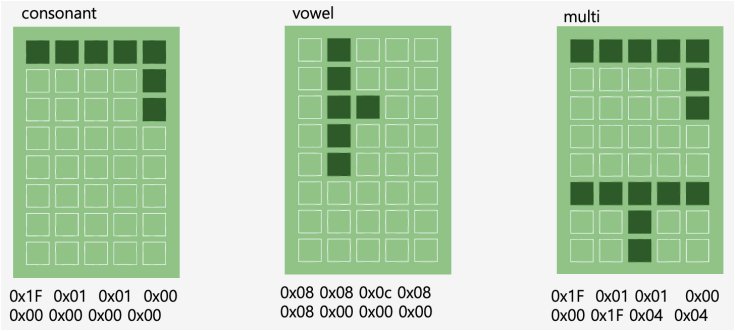
파란색 : 종성 입력할 타임 GPIOWrite(BLUE, 0);

초성 => 중성 => 종성 => 초성 => 중성 => 종성 => … (버튼 한번 누를 때마다 옮겨진다) 이렇게 사용자가 초성, 중성, 종성에 원하는 input을 완료시키면 한 글자가 완성되었다고 server에게 알릴 수 있는 완료 버튼이 있다 이 완료 버튼을 누르게 되면 write()함수를 사용하여 server에게 한 글자 입력이 완료되었다는 신호를 보내게 되고 이용자 또한 버튼이 제대로 눌렸는지 확인할 수 있게 빨간색 단색 LED가 한 번 깜빡 거릴 수 있게 하였다.

□ RPI3: Output

LCD 한칸은 16진수 8자리로 표현됨.(8 x 5)로 한 행당 16진수 한 문자로 표현. (bitmapping에 다음 사이트를 이용해 해당하는 16진수 8자리를 확인 : https://maxpromer.github.io/LCD-Character-Creator/)

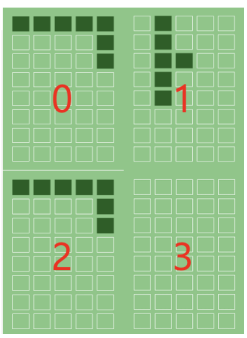
structure korean : 자음 또는 모음에 해당하는 customchar를 정의하기 위해 생성한 구조체, 식별자 key와 mapping 값 value가 있으나, 처음에는 식별자로 식별하려 했으나 해당 구조체 인덱스로 식별가능해, key의 경우 정의는 했지만 사용하진 않았다.



korean consonant[14] : 자음 14자를 매핑. 순서대로 ㄱ,ㄴ,ㄷ,...
korean vowel[9] : 모음 중 세로로 배치되는 모음 ㅏ, ㅓ, ㅣ ...
korean multi[70] : 모음 중 가로로 배치되는 모음과 자음의 조합. 14 * 5 = 70개 , 고, 노, 두, 루, ...
unsigned char empty[8] : 빈칸 정의
word_cnt : thread1, 2에서 공통으로 사용되며 글자수가 2를 넘지 않게 하기 위함. (CGRAM issue)

thread는 2개가 사용되며 thread 1 은 core로부터 input과 부저 처리 thread 2는 LCD에 글자 표현을 담당. + thread1,2 에 모두 word_cnt 전 후에 mutex lock & unlock을 걸어 overwrite방지했다.

thread 1(input_thread) : 미리 상수로 정의된 IP주소와, port number로 socket을 형성. socket으로 받는 데이터의 길이는 6으로 msg는 7로 선언.(null 고려) 실제 통신 시, char * 6 만큼 읽어옴. 연결되었다면 while로 계속해서 read함. read로 받은 값이 정상적이라면(return read > 0) word_cnt를 증가시키고, int인 front, middle, back에 저장. (string을 int로 변환) 변환된 수에 해당하는 문자를 loadCharacter로 메모리에 로드했습니다. 이후 부저 세팅 & 출력(pwm)



loadCharacter : front, middle, back, word_cnt를 parameter로 가짐. word_cnt -1을 전달하는데 전달하는 시점에 1을 더한 후라, 1을 뺀 값을 전달. loadCharacter에서는 front를 확인해 multi인지 여부를 확인하고 구분해 처리. 두 경우 모두 createCustomChar로 미리 지정된 문자를 lcd_byte함수로 전달. 화면에 표시되는 위치는 위 사진과 같이 0,1,2,3,4,5,6,7 까지 있고 cnt_idx(=word_cnt)를 이용해 2글자를 표현했다.

createCustomChar : location, charmap을 인자로 가진다. location은 화면에 표시할 위치 인덱스, charmap은 미리 지정해놓은 consonant, vowel, multi가 저장된 char 배열. cgram_location은 location과 같은데 40으로 나눈 나머지를 쓰는 이유는, LCD 테스트 중, 혹시 2글자 이상 표시가 되는지 확인해볼때 사용한 것으로 cgram_location과 location은 사실 동일한 값. lcd_byte함수에서 실제 CGRAM에 로드되는데, 먼저 lcd_byte로 해당 location로 주소를 설정하고, 이후 lcy_byte로 charmap을 전달해 CGRAM에 쓴다.

lcd_byte : LCD가 8bit 데이터가 아니라 4bit 데이터 전송을 사용하기 때문에 8bit 데이터를 4/4로(high,low)로 나눠 전송함. wiringPiI2CReadReg8을 이용해 나눈 high와 low를 전송. 화면에 표시. high 전송 후, 그리고 low 전송 후 lcd_toggle_enable을 불러 LCD가 수신할 수 있도록 한다.

thread 2 (display_thread) : ClrLcd함수로 LCD를 초기화한다. 이후 첫 line부터 typeChar를 이용해 0,1위치에 (createCustomChar에서 해당 위치에 로드한) 글자를 표시한다. 두 번째 line도 동일. (start_idx는 테스트할때 썼던 변수. - 아무역할 안함)

Not Used : typeInt, typeFloat, typeln : 숫자 입력이 있을때 사용하는데, 숫자 입력이 없어서 사용하지 않는다.

main : lcd_init()으로 lcd에 초기값 세팅. {init_korean(), init_vowel_type1(), init_multi_type1()} 에서 미리 지정해놓은 customChar를 전역의 변수에 복사. (가독성을 위해 init함수 따로 빼 놓음) 나머지 thread 생성 및 join함수.

□ RPI4: Core

코어는 크게 두 파트로 구성되어 있다.

먼저 서버를 담당하는 main 코드에서는 서버 소켓을 생성 및 주소 할당을 통해 서버를 열었다. 이후 쓰레드 함수를 통해 연결되는 클라이언트들과 동시에 통신할 수 있는 환경을 제작하였다. 클라이언트는 서버에 연결된 순서에 따라 모드를 할당 받았으며, 이후 해당 모드에 따라 역할을 수행할 수 있도록 구분하는 과정을 추가하였다. 차후에 이 부분을 개선한다면, 모든 클라이언트 쓰레드 함수를 통합하여 모드를 통해 다른 행동을 하는 것이 아닌, 각각 모드에 알맞은 쓰레드 함수를 별도로 설정하는 것이 프로그램의 성능 측면에서 더욱 발전이 있을 것이라 생각한다.

두 번째는 글자 정보를 조합하는 파트이다. glove로부터는 1초 간격으로 손가락의 굽힘 펴짐 및 기울기 센서의 정보를 6자리의 이진수로 전송받고 있다. control로부터는 초성 종성 중성에 대한 위치 정보와 글자 완성 정보를 한자리 정수로 전송받고 있다. glove로부터는 지속적인 입력을, control로부터는 필요한 시점에만 입력을 받고 있고, 이 두 정보가 한 세트로 글자 배열에 저장되어야 하므로 두 배열을 임시로 저장하는 tmp 배열을 만들었다. 이후 해당 배열에 두 정보가 다 들어갔다면 6자리의 이진수를 알맞은 자음 및 모음으로 변환하는 mapping 작업을 수행 후 글자 배열에 저장하였다. 마지막으로 control에서 글자 완성 정보를 전송하였다면 미리 설정한 두가지 오류 형식인 미완성 글자 여부와 잘못된 입력 여부를 확인한다. 이 때 오류를 확인하였다면 해당 정보를 LCD패널에 띄우고 LED를 점등하는 작업을 수행한 후 오류 정보를 control에게 전송한다. 오류가 없음이 확인되었다면 해당 글자 정보를 output에 적합한 형식으로 변환하여 output에게 전송한다.

III. 프로젝트 분석

1. 요구 사항 분석

1-1) 정확한 수화 인식: 본 프로젝트의 핵심은 정확한 수화 인식이다. 이를 위해 5개의 flex sensor와 기울기 센서를 활용하여 손가락의 구부러짐과 손의 기울기를 측정하였다. 이러한 데이터를 종합하여 각 손가락의 위치와 손의 자세를 정확하게 파악하였다. 이를 통해 쌍자음과 합성모음을 제외한 모든 한글 자음을 표현할 수 있었다. 또한, 각 센서의 데이터는 실시간으로 수집되어 입력 신호의 정확성을 높였으며, 다양한 손 모양을 정확하게 인식할 수 있도록 하였다.

1-2) 실시간 번역: 수화 인식 장갑을 통해 표현된 수화를 실시간으로 번역하는 기능을 구현하였다. 제어 버튼을 통해 초성, 중성, 종성을 선택하여 한글 문자를 구성할 수 있도록 하였다. 이를 통해 사용자는 실시간으로 수화 입력을 번역하여 텍스트로 출력할 수 있었다. 한글의 낱말 규칙에 따라 초성에 모음이 들어가는 등의 오류가 발생할 경우, Core 부분의 LCD에 오류 메시지를 출력하도록 하여 사용자가 쉽게 오류를 인식하고 수정할 수 있도록 하였다. 이를 통해 한글 수화 번역의 정확성과 사용자 편의성을 동시에 향상시켰다. 실제 테스트결과 수화를 장갑으로 표현한 후 LCD에 출력값이 표시 되는데 까지 시간은 0.5초 이내로 빠르게 동작함을 확인했다.

1-3) 한글화: 자음과 모음을 비트 매핑하여 LiquidCrystal LCD에서 한글 수화에 대한 입력과 출력을 지원하였다. 비트 매핑을 통해 각 자음과 모음을 효율적으로 처리할 수 있었으며, 이를 통해 다양한 한글 문자를 LCD에 정확하게 출력할 수 있었다. 또한, 비트 매핑을 통해 각 자음과 모음의 조합을 쉽게 처리할 수 있었으며, 이를 통해 사용자가 손쉽게 한글 문장을 구성할 수 있도록 하였다. 이와 같은 한글화 작업은 한글 수화 번역 시스템의 핵심 요소로 작용하여 사용자에게 직관적이고 편리한 인터페이스를 제공하였다.

2. 도전적 이슈

2-1) 한글의 초, 중, 종성 구분: 한글은 영어와 달리 초성, 중성, 종성으로 구성되는 문자 체계를 가지고 있다. 이러한 특징 때문에 한글 수화를 인식하고 번역하는 시스템에서는 각 성분을 정확히 구분할 수 있는 컨트롤러가 필요하다. 이를 위해 두 개의 버튼을 사용하여 초성, 중성, 종성을 구분하고 제어할 수 있도록 하였다. 사용자는 버튼을 눌러 각 성분을 선택하고 입력할 수 있으며, 이를 통해 한글 문자를 구성할 수 있다. 이러한 방식은 사용자가 한글 수화 입력을 보다 직관적이고 정확하게 수행할 수 있도록 돕는다.

2-2) Flex sensor의 동작: Flex sensor는 구부러짐에 따라 저항 값이 변하며, 이 값을 0부터 1024까지의 범위로 읽을 수 있다. 이를 통해 손가락의 구부러짐 여부를 판단할 수 있었으나, 센서 값이 일관되지 않는 문제가 발생하였다. 이를 해결하기 위해 flex sensor의 값을 이진화하여 처리하였다. 즉, 센서 값이 0에서 999 사이일 경우 0으로, 1000 이상일 경우 1로 변환하여 구부러짐 여부를 명확하게 구분하였다. 이러한 이진화 과정을 통해 센서 데이터의 일관성을 확보하고, 수화 인식의 정확성을 높였다.

2-3) 출력 – 소리, 텍스트: 초기 계획은 수화를 목소리로 번역하여 음성으로 출력하는것이었다. 그러나 적절한 음성 출력 센서나 디바이스를 확보하지 못하여 이 기능은 구현하지 못하였다. 대신 수화 번역 결과를 LCD에 텍스트로 출력하였다. 그러나 사용된 디바이스의 제약으로 인해 LCD에 두 글자 이상을 연속으로 출력하는 것이 어려웠다. 이러한 제한 사항을 극복하기 위해, PC나 앱을 통해 출력하는 방법을 고려하였다. Raspberry Pi가 아닌 다른 플랫폼을 사용하여 출력한다면, 더 많은 글자를 쉽게 출력할 수 있을 것으로 예상된다.

3. 팀원별 역할

이지수	RPI 1: Glove Flex Sensor와 Tilt sensor를 이용한 장갑 제작 Sensor Data를 입력 받아 Core로 송신 구현
남현원	RP1 2: Control 버튼을 이용한 글자 입력 제어 기능 제작, LED를 통한 제어 정보 출력 Place Data 송신 및 Error 정보 수신 구현, Buzzer를 통한 Error 정보 출력 Tilt Sensor 구현, 이진수 입력과 글자 Mapping 함수 구현
정용훈	RP1 3: Output 한글 CustomChar bit-mapping & LCD 패널과 Buzzer를 통해 완성된 글자 정보 출력 숫자 정보 송신 구현 및 정보를 LCD 조건에 맞게 한글로 Setting
전진	RPI 4: Core LCD 패널과 LED를 통해 Error 정보 출력 서버 생성 및 수신한 정보를 조합 및 mapping하여 Output으로 송신 구현

4. 제언

플렉스 센서에서 입력을 굽힘과 펴짐 이진데이터로만 검출하는 것이 아닌, 정확한 각도값을 입력받아 사용할 수 있다면 특정 동작을 나타내는 수화를 추가적으로 식별하는 것이 가능하다. 따라서 고성능의 센서를 활용한다면 수화 인식의 정확도 및 활용성을 향상시킬 수 있을 것이다.
Output의 경우 한글 자음, 모음에 해당하는 CustomChar의 bit 정보를 한 코드내에 정의하여 사용했는데, 가로로 배치되는 모음의 경우 자음 14자 x 모음 5자 로 총 70개의 bit mapping정보가 필요했다. 해당 매핑에 사용되는 정보가 많아 코드가 길어지고 가독성이 떨어지는 부분이 있다 생각이 되어, 별도의 텍스트파일로 정의하여 해당 파일을 읽는 식으로 구현하면 이 부분을 개선할 수 있을 것으로 생각된다.
프로그램의 전반적인 구조로 보았을 때 하나의 코어에 서로 다른 상호작용을 하는 클라이언트 3개가 연결되어 있는 구조이다. 따라서 동일한 클라이언트 쓰레드 함수로 관리하는 것보다 별도의 함수를 두어 관리할 경우, 프로그램 수행 과정에서 더욱 적은 시간과 자원을 소모할 것이다. 따라서 동일 쓰레드 함수 내에서 모드로 역할을 할당하는 방식이 아닌, 역할에 따른 함수

자체를 할당하는 방식으로 바꾼다면 더욱 성능이 좋아질 것이라 예상한다.
부저를 구현할 때 처음에는 GPIO를 사용하였다. GPIO를 사용하였을 때 소리가 잘 나오긴 하였다. control부분에서 에러의 종류를 구분하기 위하여 usleep 함수를 사용하여 소리출력을 두번 출력하는 경우도 있다. 그러나 이부분에서 소리가 꺼졌다가 켜지는 것이 아닌 계속 출력된다. 처음에는 코드 문제가 있나해서 여러 번 수정하였지만 바뀌지 않았다. 그리하여 PWM으로 구현하여 작동시켰더니 정상적으로 작동하기 시작하였다. GPIO에서는 되지 않고 PWM에서는 동작이 된 이유를 생각해본 결과 소리를 몇 초 내에 몇번 켜졌다가 꺼지게 하려면 디지털 신호에서 HIGH 신호와 LOW의 신호를 일정한 주기로 조절할 수 있어야 하는데 PWM은 듀티 싸이클을 직접 조절하여 매우 정확한 타이밍에 제어가 가능하다는 것을 조사하게 되었다. 그리고 PWM 신호는 일정한 주기를 갖는데 이러한 특징 덕분에 신호의 주파수가 더욱 정확하게 유지되어 이러한 특징들 때문에 정밀한 제어가 가능하다는 것을 알게 되었고 이로 인해 부저를 PWM으로 구현하였을 때 원하는 방향대로 된 것 같다. 그리고 GPIO는 LED 구현과 버튼 구현 부분에서 많은 GPIO가 연결되어 있다 보니 많은 다른 작업에 의해 방해받은 부분이 있는 것 같다. 정확한 시간 제어가 필요한 부분은 PWM으로 구현하는 것이 더 유리할 것 같다는 생각이 든다.

5. 프로젝트 진행 일정

5. 25 (원천 정보관)	각 파트별 센서 분배 및 구현 계획
5. 30 (원천 정보관)	각자 구현한 센서 확인 및 통신 format 지정 (flex외 구현 완료)
6. 1 (원천 정보관)	수화 예외 케이스 확인 및 수정
6. 5 (원천 정보관)	5개의 flex sensor 동시 동작 구현 및 통신 구현 확인
6. 6 (원천 정보관)	Flex sensor를 포함하여 전체 통신 테스트
6. 7 (원천 정보관)	발견된 예외 케이스 처리
6. 8 (원천 정보관)	최종 통신 확인 & 에러 케이스 확인 및 데모 영상 촬영

□ 프로그램 소스 코드

https://github.com/hun9008/signTranslateMachine.git