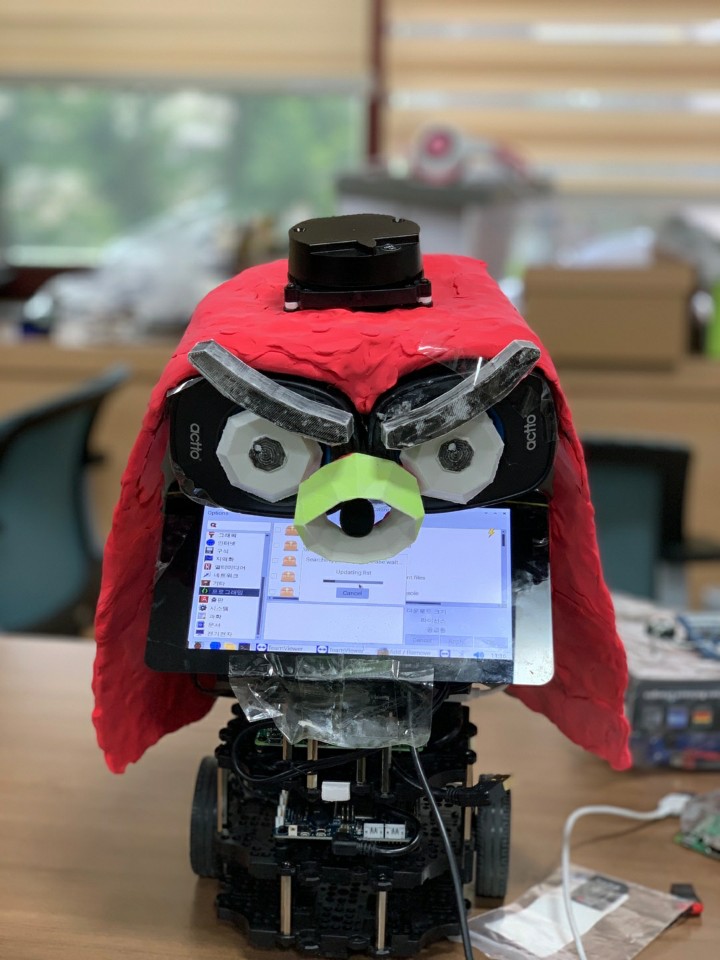
박학다식

**음성인식과 Lidar기반 자율주행 기술을 이용한**

**박태준 학술정보관 전용 맞춤 도서관 사서**



POSCO AI · Big Data 아카데미 10기 A반 1조

김혜준 박미현 안정미 하석민 한충완

**목 차**

1 프로젝트 개요

1.1 프로젝트 추진배경

1.2 문제인식 & 프로젝트 방향성

2 프로젝트 소개

2.1 구현 범위 및 방법

2.2 음성안내 로봇 구조

3 프로젝트 개발

3.1 ROS 설치

3.2 ROS 개발

3.3 음성안내 구현

3.4 디자인

4 결론

4.1 프로젝트 결과

4.2 프로젝트 한계점 및 개선사항

5 참고문헌

**1. 프로젝트 개요**

**1.1 프로젝트 추진배경**

본 프로젝트는 AI가 일상생활을 영위하는 사람들을 편리하게 만들어 주는 도구가 되어야 한다는 사고에 기초하여 진행되었다. 프로젝트를 시작하기 전에 현재 진행되고 있는 AI 프로젝트 분야에 대해 조사한 결과, 아직 AI가 인류 지성의 탑인 도서관에는 그리 많은 영향을 미치지 못한다는 것을 확인하였다. 또, 현재 도서관이 처한 열악한 환경 또한 프로젝트를 시작한 강한 동기가 되었다. 이러한 이유로 본 프로젝트는 도서관 AI 사서를 만드는 일에 착수했는데, 도서관에 AI가 필요하다고 판단을 내린 구체적인 이유는 다음과 같다.

**1.2 문제인식 & 프로젝트 방향성**

1. 도서관의 법정 최소 사서 인력 불충족 문제

2019년 공공도서관 통계 결과에 의하면 전체 도서관의 약 39%가 법정 최소 사서 인력인 3명에 미달하는 인원으로 운영되고 있는 것으로 나타났다. 이는 두 가지 이유가 있는데, 첫째는 도서관의 공공성을 충족시키기 위해 도심이 아닌 외지에 도서관이 위치한 경우가 대다수인데, 이 경우 그만큼 도서관 사서를 모집하기도 쉽지 않기 때문이다. 둘째는, 도서관의 열악한 재정적 여력 때문에 사서 일이 자원봉사 성격을 띠고 있어 사람들을 유인하지 못하고 있기 때문이다. 도서관 사서가 하는 일을 AI가 탑재된 기기가 대체한다면 이같은 문제가 경감될 것으로 판단하였다.

1. 복잡한 서가 위치

2019년 공공도서관 통계 결과에 의하면 공립도서관 이용자 중 약 65%가 도서관 이용에 혼란을 느끼는 것으로 나타났다. 도서관 이용자는 본인이 찾으려는 책을 검색하고, 코드를 알아낸 다음, 책의 위치를 찾아다니는 과정을 거친 후에야 책을 빌릴 수 있다. 이같은 과정이 복잡하기 때문에, 사용자가 원하는 책을 직접 찾아주는 자율주행 AI 기기의 필요성이 대두되었다.

1. 리모델링으로 인한 서가 배치 변경

도서관이 주기적으로 행하는 리모델링도 본 프로젝트의 주요한 동기가 되었다. 조사 결과 본 프로젝트가 주 배경으로 삼은 포항공과대학교 박태준 학술도서관의 경우 리모델링이 2020년 겨울부터 진행되는 것으로 파악되었다. 도서관이 리모델링 되어 구조가 달라질 경우 사용자의 혼란을 야기할 수 있다. 사용자에게 음성으로 길을 안내해주고 디스플레이로 보여주는 역할을 하는 AI 기기가 있을 경우 이같은 문제를 최소화할 수 있을 것으로 보았다.

위의 3가지 문제점을 해결하거나 혹은 최소화하는 AI를 구현하는 것을 목표로 삼고 프로젝트를 진행하였다.

**2. 프로젝트 설명**

**2.1 구현 범위 및 방법**

도서관 AI사서를 구현하기 위해 가장 핵심이 되는 기술은 ‘음성인식’과 ‘자율주행’이다. 음성인식은 도서관 이용자의 손을 자유케 할 것이다. 기존의 방법은 도서 검색대에서 도서 제목을 정확하게 입력하는 방법이 책을 찾는 유일한 방법이었다. 그러나 음성인식을 구현하면 거기에 드는 에너지도 절약할 수 있을 것이다. 음성인식은 기존의 과정을 거치지 않아도 사용자가 원하는 책을 바로 찾아주기 때문에, 책은 꼭 도서관 검색대에서 타자를 이용해 찾아야 한다는 기존의 관습을 타파할 것이다.

자율인식은 사용자의 시간을 절약해 줄 것이다. 검색을 통해 책의 코드를 알아낸 사용자는 수 많은 책 중에서 자신이 찾은 코드가 적힌 책을 찾아야 하는 난관에 봉착한다. 많은 노력 끝에 책을 찾으면 다행이지만, 그렇지 못한 경우 도서관 사서의 힘을 빌리게 된다. 자율 주행을 구현하고 책이 있는 곳까지 데려다주는 기계가 있다면 이러한 부작용을 해결할 수 있을 것으로 봤다.

따라서 본 프로젝트는 음성인식과 자율주행을 구현하는 것을 최우선 과제로 삼았다. 음성인식은 인식률이 관건이다. 논문 조사 결과 현재 구현된 API중 가장 인식률이 높은 것이 Google API이기 때문에 이를 기반으로 음성인식이 구현될 것이다. 사용자의 음성을 받아 이를 text로 변환하여 해당 정보를 찾고, 다시 이 정보를 speech로 구현하여 사용자에게 정보를 음성으로 전달하는 STT, TTS를 모두 실현하는 것이 프로젝트의 범위에 포함된다. 여기에 인식률을 조금이라도 더 높이는 방법을 찾아 시도를 할 것이다. 이 시도가 성공적이었는지는 후술할 것이다.

자율주행은 ROS를 기반으로 구현할 것이다. 전기수 선배들의 자료를 포함한 기존의 자율주행은 임의로 정해진 라인을 따라가는 line tracking이 거의 대다수를 이루고 있다. 그러나 도서관의 경우 특정 라인을 그릴 수 없기 때문에 다른 방법을 강구했고, 라인이 없이도 자율주행을 할 수 있는 모듈로 ROS를 선정했다. ROS의 경우 SLAM, NAVIGATION까지 설치를 한다면 Lidar를 이용하여 주위의 지형지물을 파악한뒤 적절한 경로를 찾아 자율주행을 할 수 있다는 특징이 있다. 따라서, TurtleBot에 ROS와 관련 소프트웨어를 설치하여 Lidar를 이용한 자율주행을 구현하는 것까지를 프로젝트의 범위로 삼았다.

**2.2 음성 안내 로봇 구조**

위의 과정이 다 실현되었을때 예상되는 결과는 다음과 같다. 사용자가 발화를 하면 그 내용을 Google API가 text로 변환하여 관련 정보를 찾는다. 예를 들어 “셰익스피어 소넷이 어디있는지 알려줄래?”라는 발화내용이 들어오면 Google API는 해당 책을 미리 저장된 DB에 대조하여 정보를 찾는다. 곧 그 위치와 대출가능여부가 나오게 되고, Google API는 이를 다시 Speech로 변환하여 사용자에게 음성과 디스플레이로 안내한다. 이와 동시에 Turtlebot은 책이 있는 장소를 인식하고 Lidar를 이용해 자율주행을 시작한다. 사용자는 Turtlebot을 따라가서 본인이 원하는 책을 찾게 된다. 본 프로젝트는 이 과정을 염두에 두며 다음과 같은 개발 과정을 통해 목표에 다가가려는 시도를 했다.

**3. 프로젝트 개발**

**3.1 ROS 설치**

Turtlebot3를 구동하기 위해서는 ROS를 설치해야한다. 그런데, 선행 기수가 구현한 것과는 ROS와 Ubuntu 둘 다 버전의 차이가 있어서 ROS를 새로 설치하기로 결정하고 아래와 같이 3가지 방식으로 진행했다.

**3.1.1 Ubuntu 18.04 + ROS1**

우분투를 설치한 후 무선랜 설정을 먼저 해주고 ROS1 설치를 진행하였다.

무선랜 설정하는 코드는 아래와 같다. 띄어쓰기와 간격에 주의해야 한다.

sudo vi /etc/netplan/50-cloud-init.yaml

|  |
| --- |
| network:  wifis  wlan0:  dhcp4: **true**  **access-points:**  **"olleh\_WiFi\_\*\*\*\*":**  password: "와이파이비밀번호"  version: 2 |

sudo netplan apply

ifconfig

조사결과를 바탕으로 ROS는 리눅스 체제의 OS와 호환이 잘 될거란 판단이 들어 우분투를 깔고 그 위에 ROS를 얹는 방식으로 진행해보았다. 인터넷에서 검색해본 결과 많은 이들이 Ubuntu 16.04 MATE + Kinetic으로 진행하였을 때 매끄럽게 구동되었기에 Ubuntu 16.04를 찾아보았으나 공식 사이트에서는 Ubuntu 18.04 혹은 20.04 밖에 지원을 하지 않아 일단 16.04를 깔고 그 위에 Kinetic 혹은 Melodic을 설치하는 방식으로 진행하였다. Ubuntu를 설치한 후 GUI를 별도로 깔아 강의실의 리눅스 환경과 비슷하게 만든 후 SD카드를 둘로 나눠 Kinetic과 Melodic 하나씩 설치를 시도해보았다. Kinetic보다 Melodic이 필수 Package까지는 설치되긴 했는데 터미널 창에서 실행이 너무 느려서 터틀봇을 구동시킬 수가 없었다. 호환 문제인 것 같아서 Ubuntu 16.04를 비공식적 파일로 구해 설치해보았다. 그러나 다음과 같은 오류가 계속해서 발생했다.

ROS에서 kinetic을 깔 때 가장 많이 참고한 사이트는 wiki ros(<http://wiki.ros.org/kinetic/Installation/Ubuntu>)이다. 이 페이지에 나와있는 대로하면 kinetic이 설치가 되지만 가장 많이 막힌 부분은 package를 설치하는 부분이었다. packages의 경우 약 200개의 package가 한꺼번에 설치되지만, 항상 gmapping에서 막혀서 설치가 번번이 좌절되었다. 이를 극복하기 위해 구글링을 통해 다양한 방법을 강구해봤지만 끝내 이 한계를 넘어설 수 없었다. 고심 끝에 문제는 코딩이 아닌 ubuntu라는 운영체제의 특성에 기인하고 있다고 자체적으로 결론을 내렸고, 가장 대중적인 운영체제인 Window 10으로 바꾸고 ROS설치를 진행해보기로 결정했다.

**3.1.2 Windows 10 + ROS1**

라즈베리파이에 Windows 10을 설치하여 그 위에 ROS를 구동시키는 방법을 생각했는데 일단 라즈베리파이에 Windows를 설치한 사례가 별로 없어 난항을 겪었다. 보통은 Windows Boot USB를 이용하는데 여건상 구매나 대여가 힘들었고, Windows 10에서 제공하는 ISO를 다운받아서 라즈베리파이에 설치했다. 그 후 라즈베리파이를 부팅시켰더니, 윈도우 os를 구동하는데만 몇 시간이 걸렸고 기본적인 셋팅하는데 또다시 몇 시간 걸렸는데, 결국 부팅된 윈도우가 인터넷 설정이 되지 않아 ROS를 사용할 수 없는 문제에 봉착했기에 Windows 10 사용은 포기했다.

**3.1.3 Raspbian + ROS1**

라즈베리파이의 OS에 ROS1 Kinetic을 설치를 시도하였으나 잘 되지 않아 ROS1의 최신 버전인 Melodic을 설치하여 진행했다. 역시 필수 Package가 잘 깔리지 않아 새로운 방법을 모색하게 되었다.

Kinetic에서 에러가 발생한 코드는 아래와 같다. ROS 저장소 설치. 다른 방법으로 주소를 바꿔가며 해결했으나 계속 여러 에러가 나서 Melodic으로 시도하게 되었다.

sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

Melodic의 경우 아래와 같이 Package를 설치할 때 에러가 난다.

sudo apt-get install libogre-1.9-dev

이 코드를 실행하면 약 180개의 package가 일괄적으로 설치되는데, 하나의 패키지가 설치된 다음에야 다음 패키지로 넘어가서 설치가 진행된다. 그런데, robot\_navagation패키지에 계속 문제가 생겨서 구글링을 통해 적절한 해답을 찾기를 시도했다. 사실상 모든 경우의 수를 다 동원한 후에 이 문제를 해결하여 ROS설치가 비로소 가능했고, ROS개발로 프로젝트가 이어질 수 있었다.

**3.2 ROS 개발**

**3.2.1. 네트워크 설정 및 통신환경 구축**

Remote PC와 터틀봇을 제어하는 라즈베리파이는 같은 라우터의 네트워크를 공유해

야 한다. 그래서 공유기를 통해 Remote PC와 터틀봇이 같은 네트워크를 사용하도록

하였고 SSH 통신 설정을 하였다.

여기서 Remote PC를 선정하는데 있어서 애를 먹었다.

**<교육장 PC>**

****

교육장에 있는 PC는 Linux 환경이라 ROS를 설치하는데 쉽고 구동하기 유용하기며 성능도 좋기에 가장 먼저 선정하려 했던 Remote PC였다. 하지만 독서실에서 구동을 하려면 Remote PC를 독서실까지 들고 가야하는데 이 PC를 들어본 분이면 알겠지만 생각보다 꽤 많이 무겁다. 또한 터틀봇과 같은 네트워크를 공유해야 하는데 이 PC는 고정 IP가 할당되어 있어서 같은 라우터 안에 놓기 힘들었다. WIFI 또한 잡지 못하는 PC여서 애를 먹었다. 이에 다른 Remote PC를 선정하는 것을 고려하게 되었다.

**<개인 노트북>**

****

두번째로 고려했던 것이 노트북이었다. 개인 노트북이나 포스코에서 제공해준 Dell 노트북을 Remote PC로 선정하는 것을 고려했다. 왜냐하면 우선 가볍고 따로 설정을 해주지 않아도 WIFI를 수신할 수 있기에 손이 덜 가는 Remote PC라 판단했다. 하지만 어림도 없지.. ROS를 설치하는데 문제가 생겼다. Window 기반에서 ROS를 설치하는것이 꽤나 까다로웠고, VMware를 통해 가상환경을 만들어 그 안에 Linux환경을 만들고 그 안에 ROS를 설치하자니 성능저하가 우려되었다.

SLAM이나 Navigation 기능을 사용하려면 성능이 가장 중요하다 판단되어 결국 교육장 PC를 Remote PC로 선정하게 되었다. 조원중에 가장 헬스를 많이 하는 사람이 교육장 PC를 들고다니기로 했다.

**<무선랜카드>**



교육장 PC를 Remote PC로 정해주니 가장 먼저 생긴 귀찮은 일은.. 이 PC를 WIFI를 수신하게 만드는 일이었다. 다행히도 아카데미에서 무선랜카드를 소유하고 있어서 이를 PC에 설치하는것은 그리 어렵지 않았다.

**<공유기>**

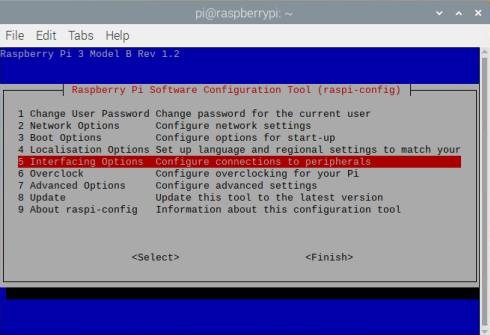
****

랜카드와 마찬가지로 공유기도 있으니, 이를 통해 같은 라우터 안에 연결시키는 것은 그리 어렵지 않았다.

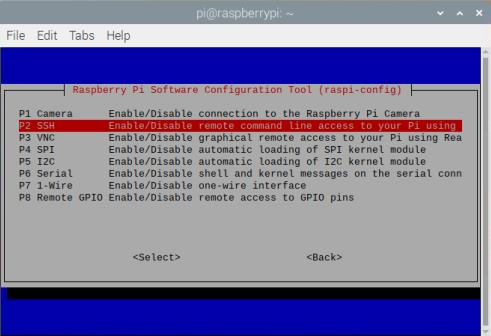
이렇게 같은 라우터 안에 넣어놨으니 SSH 통신을 구현해보았다. 해당 방법은 아래와 같다

터틀봇에서 아래와 같이 입력하여 SSH구현을 위한 환경설정을 한다.

****

****

[5 Interfacing Options]를 선택한다.

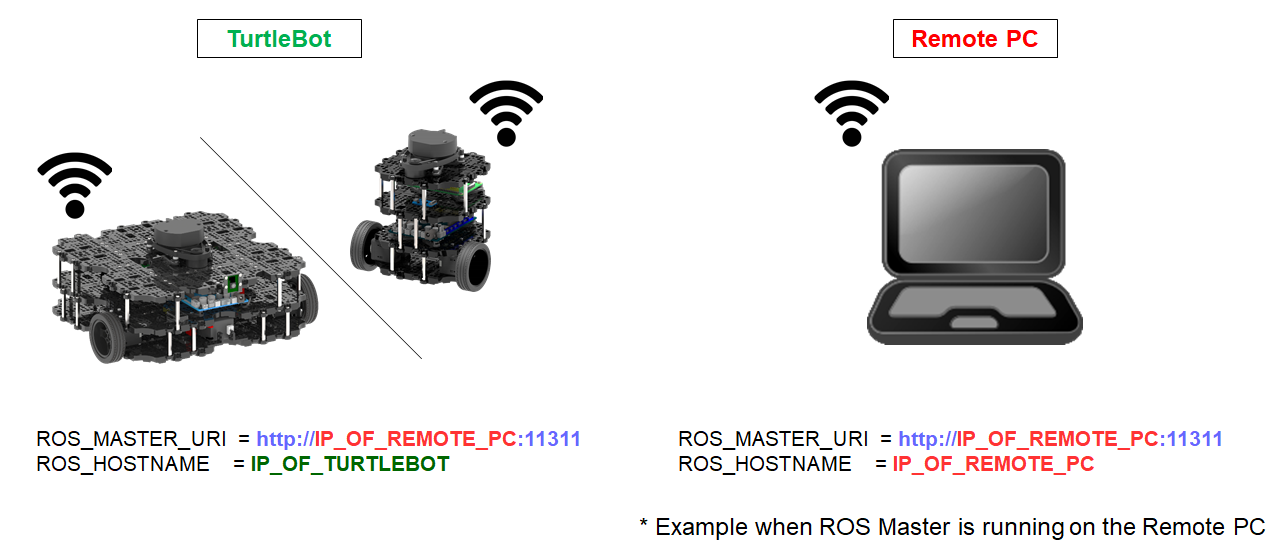
****

[P2 SSH]를 선택하여 SSH 통신 설정을 Enable로 설정한다.

자 이제 환경설정이 끝났고



Remote PC와 터틀봇 둘 다 위의 코드를 통해 서로의 IP를 맞춰줘야 한다.

****

해당 방법은 로보티즈 E-메뉴얼에 자세히 나와있다.

후배 기수들을 위해 친절히 링크를 아래에 알려드리겠다.

[**https://emanual.robotis.com/docs/en/platform/turtlebot3/pc\_setup/#install-ubuntu-on-remote-pc**](https://emanual.robotis.com/docs/en/platform/turtlebot3/pc_setup/#install-ubuntu-on-remote-pc)

사실 여기만 알아도 ROS를 구현하기 한결 편하다.

뒤에 내가 쓸 내용들이 무색할정도로 저 E-메뉴얼에 자세히 나와있다.

필자는 프로젝트 마감 1주일전에 알아버렸다는게 함정이긴하지만.. 후배 기수들은 미리 알고 들어가서 그대로 따라해보길 바란다.

어쨌든 위의 방법을 통해 IP를 맞춰줬다면 꼭



위와 같은 source를 해주길 바란다. 이걸하지 않으면 수정한 파일이 적용되지 않는다.

자 이제 연결이 되었는지 안되었는지는 roscore를 실행시켜보면 알 수 있다.

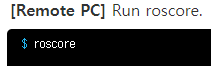


Remote PC에서 위를 실행시켰을때 잘 된다면 성공적으로 연결이 된 것이다.

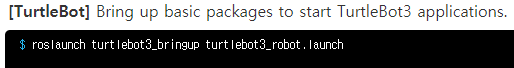
안된다고 걱정하지 마라. 필자는 혼자서 이거 실행시키는 것만 이틀이 걸렸다. 처음부터 안되는게 당연한 것이니 걱정하지말고 갓보티즈의 E-메뉴얼을 따라하기 바란다. 분명 본인이 놓친게 있을 것이다.

**3.2.2. Rviz 실행**

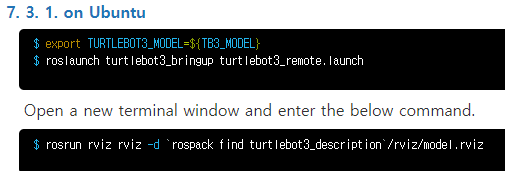
사실 여기서부터 작성할 내용은 E-메뉴얼에 더 자세히 나와있으니 해당 게시글을 보는 것을 추천한다. 하지만 여기서 이대로 마무리해버리면 귀찮아서 안 쓴 것 같아보이므로 구색을 갖추기 위해 작성하겠다.



자 우선 Remote PC에서 roscore을 실행시켜 우리의 ROS를 실행시킬 준비를 마치자

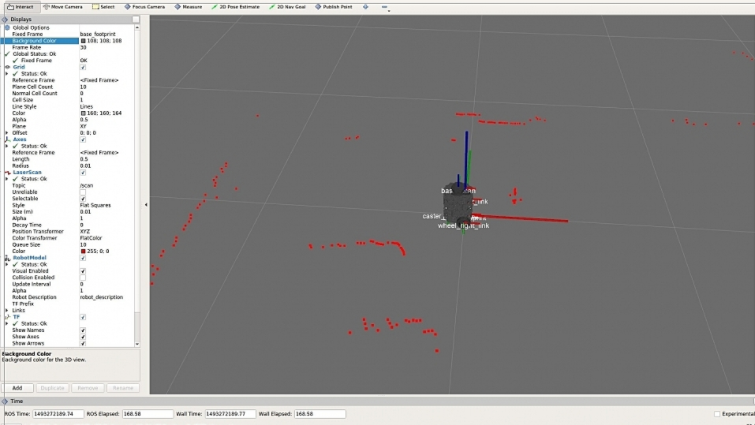


이 후에 터틀봇에서 위의 코드를 작성하여 bringup을 해준다. bringup을 통해 package를 받아올 준비를 마친다. 하지만 어림도 없지.. 보통 여기서 오류가 많이 난다. 해당 오류를 일일이 언급할 순 없고 구글링을 통해 찾아보길 바란다.



본인이 우분투 환경이라면 위와 같은 코드를 치면된다. 첫번째 줄에 TB3\_MODEL 부분을 기종을 쓰면 된다. 우리는 터틀봇을 사용하므로 =$turtlebot 이라고 작성하였다.

이 3줄을 모두 작성하면 아래와 같은 결과물이 나온다.



여러분이 처음 이 장면을 보고 눈물이 난다면 그건 울보라서가 아니다. 마동석도 눈물을 흘려야 정상이다. 여기까지 오는데 많은 시행착오가 있으리라 생각한다. 한번에 안되는게 정상이다.

우선 자주 하는 실수를 말하자면 Lidar가 안켜져 있거나

export turtlebot을 할때 뭔가를 잘못 적은 것일 확률이 높다.

그 외에 파이썬이나 아두이노의 버전 문제로 안 될 확률도 있다.

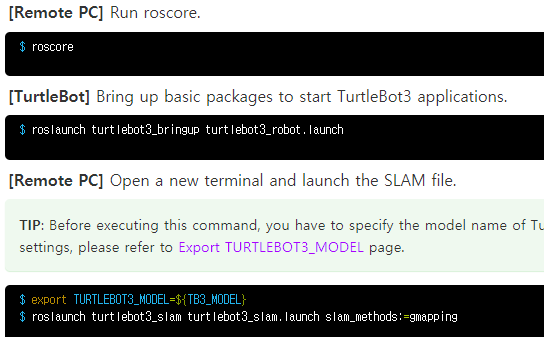
사실 여기까지 와봤으면 알겠지만 온갖 말도 안되는 오류가 많다. 보고서만 보면 쉬워보이겠지만 결고 그렇지 않다.

참고로 본인은 열심히 일주일간 삽질을 한 결과 부여받은 터틀봇의 배터리가 고장나서 안돌아간다는걸 깨달았을때 몸에서 사리가 나오는 것을 느꼈다.

여러분도 미리 여분의 하드웨어를 미리 구입하는것을 추천한다.

**3.2.3 SLAM 실행**

사실 Rviz까지 제대로 작동한다면 SLAM은 쉽게 잘 돌아갈 것이다.



위와 마찬가지로 roscore와 bringup을 실행한 뒤에

해당되는 코드를 작성하면 된다.



그럼 이렇게 실시간으로 매핑되는 그림을 볼 수 있다.

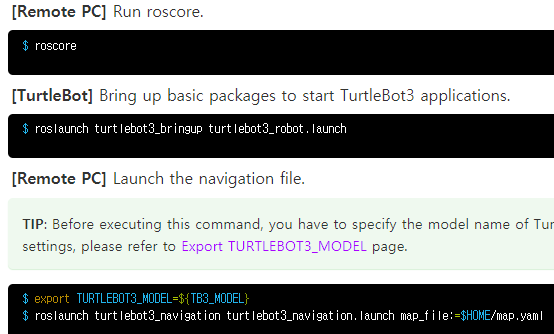
**3.2.4 Navigation 실행**

이 부분을 구현하기 전에 미리 말해주자면.. 사실 이 부분을 생각하고 여기까지 왔다면 이미 늦었다. 현실을 말해주자면 통신 문제로 포스코 아카데미에선 구현할 수가 없다.

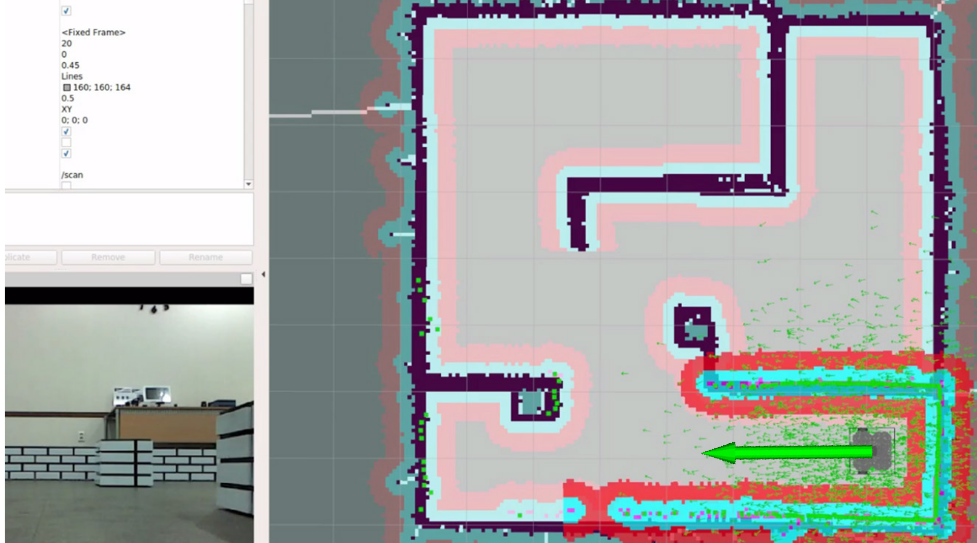
억지로 억지로 구현을 해도 속도가 매우 느리고 움찔움찔 움직인다. 또한 통신 환경이 너무 열악해서 자율주행을 하다가 도중에 끊겨버리면 그냥 가던 방향으로 돌진해버린다.

고로 구현을 하고 싶다면 라즈베리파이가 아닌 더 좋은 하드웨어를 사용하거나 더 좋은 통신 환경에서 구현하기 바란다.

우선 방법은 알려드리겠다.



아까 SLAM할때와 비슷하다. 위와 같은 코드를 작성하면 아래와 같은 그림이 나오는데



지도에 이동하고 싶은 위치를 찍으면 해당 위치로 자율주행하여 움직인다.

이 얼마나 신기하고 신박한 기술인가

하지만 이동하는데 정말 한세월 걸린다.

하드웨어와 통신 문제로 인해 사실상 구현되어도 문제다.

본인은 이 문제를 알아버린게 최종 발표 전 날이었다.

여러분들은 더 좋은 환경에서 만들기 바란다.

**3.2.5 터틀봇 원격조정**

****

(... 이하 코드 생략)

터틀봇이 알아서 자율주행 하는 것이 가장 이상적이지만 통신 문제와 라이다 성능 및 배터리 고장 문제 등으로 인해 생각보다 완벽한 자율주행 모습이 연출되지는 못했다. 자율주행 외에도 터틀봇이 잘 구동되는지 확인하기 위하여 방향키를 이용해 원격 조종을 시켜보는 것을 추천한다. 방향키로 터틀봇 제어하는 코드는 터틀봇 api를 이용했다. teleop 구동 조건은 파이썬 2.7버전 이상, pynput모듈 1.6.8이상부터이다.

“ roslaunch turtle\_teleop turtle\_teleop\_multi\_key.launch “

명령창에 다음을 입력하면 api를 바로 이용할 수 있다.

**3.3 음성안내 구현**

**3.3.1 STT & TTS 구현**

찾고자 하는 책의 이름을 음성으로 입력 받아 음성으로 위치를 출력하기 위해 음성 안내 코드를 구현하였다. 진행 순서는 다음과 같다.

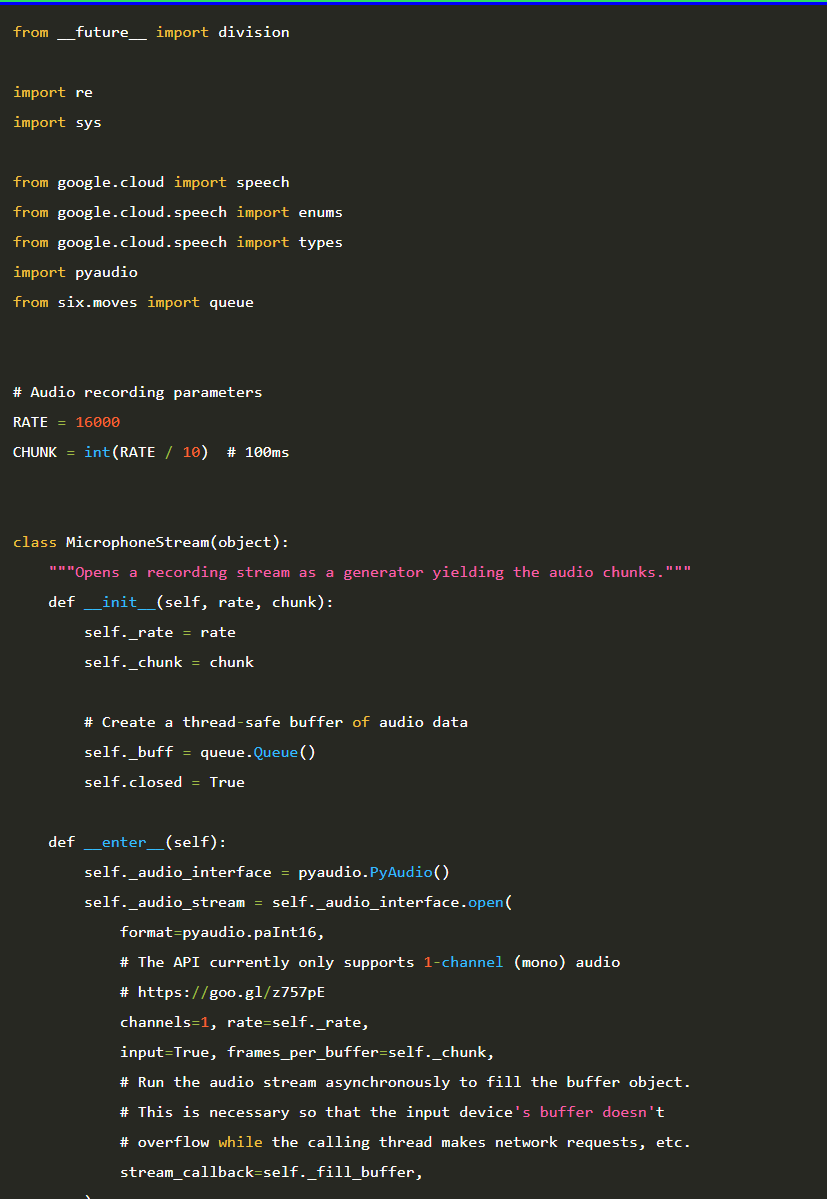
1. STT(Speech To Text) 구현

2. TTS(Text To Speech) 구현

3. 두 코드 합치기

음성 안내 코드를 구현하기 위해 Google Cloud Speech API를 이용하였다. 해당 API를 이용하기 위해서는 구글 계정 키를 할당 받아야 한다.

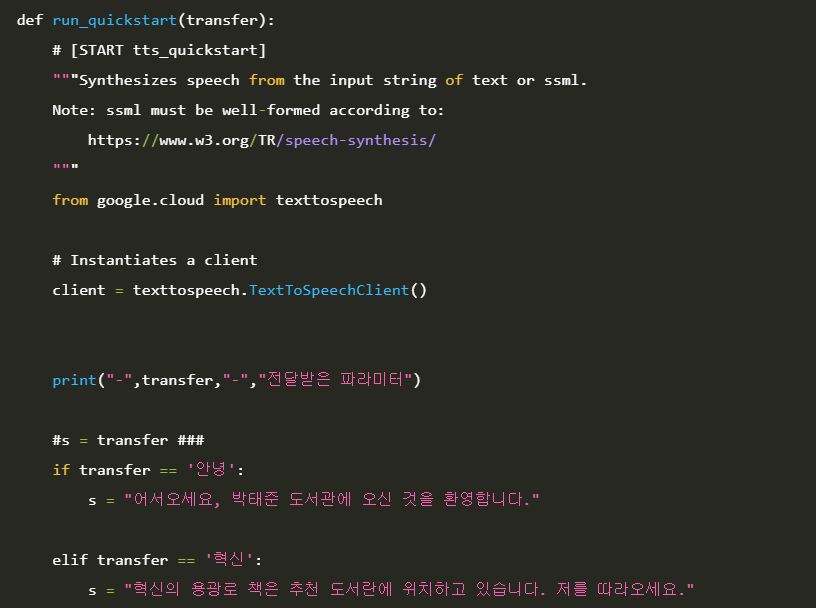
1. STT(Speech To Text)



(. . .이하 코드내용 생략)

STT는 Speech To Text의 약자로 사람의 자연어를 텍스트로 변환하는 기술이다. transcribe\_streaming\_mic 코드를 사용하였으며 해당 api에 우리에게 필요한 코드를 추가해서 사용하였다. 코드를 돌리기 위해서는 필요한 모듈들 설치가 요구된다. pyaudio 모듈을 사용하기 위해서는 파이썬 3.6 버전에 설치하는 것이 좋으나 가상 환경을 따로 만들어서 해도 좋다. 우리는 필요에 맞게 listen\_print\_loop 함수를 일부 수정하였으며 transcript+overwrite\_chars 변수를 통해 음성 데이터를 받아 넘겼다.

2. TTS(Text To Speech)



(. . .이하 코드내용 생략)

TTS는 Text To Speech의 약자로 텍스트를 사람의 자연어로 변환하는 기술이다. 첨부된 소스 코드의 quickstart 파일이며, 메인함수가 있는 파일(stt파일)에서 import시켜 구동하였다. 해당 코드가 정상적으로 돌아가기 위해서는 google.cloud의 texttospeech모듈을 반드시 import 해야 한다. 참고로 google.cloud 모듈은 파이썬 버전 3.6이상부터만 지원하니 이하 버전에서 코드를 돌리려는 수고는 범하지 않기를 바란다. 이제 코드 내용으로 돌아가서... 우리는 필요에 맞게 run\_quickstart()함수를 일부 수정하였다. 원래는 매개변수가 설정돼 있지 않았지만 값을 받기 위해 run\_quickstart(transfer)로 인자를 받도록 매개변수를 설정하였다. 즉 메인 stt파일에서 tts함수 호출 시 transfer인자가 넘어올 것이며, 이 transfer로 문장과 대조시켜 필요한 기능을 수행하도록 할 것이다. run\_quickstart(transfer)함수 역시 return값을 내보내는데, 들어온 음성 입력 값에 걸맞는 출력을 내보낼 것이다. 마지막으로 도서관 안내에 필요한 상황들을 정리하기 위해 분기문을 이용하였다.



원래의 TTS API의 run\_quickstart() 함수에는 매개변수도 return값도 설정돼 있지 않았는데 우리는 코드를 합치고 파라미터 값을 전달받아 넘기기 위해 다음과 같이 매개변수도 설정하고 return값도 설정해주었다.

3. 두 코드 합치기



STT와 TTS코드를 합쳤다. 즉, 사용자의 음성을 입력 받으면 텍스트로 변환하여 그 텍스트를 대조하여 상황에 걸맞는 음성 출력 값을 다시 스피커를 내보내도록 만들었다. main함수는 STT 파일에 내장돼 있으므로 STT파일을 실행시키면 된다. TTS파일은 필요한 함수만 정의되어 있고 STT파일에서 해당 모듈을 import 시켜놓으면 된다. 실질적으로 run시킬 때 구동되는 코드는 위 코드의 맨 아래 main함수 부분이며 이외는 모두 메인 함수에서 호출시킬 함수들이다.

while문을 사용하여 음성을 입력받고 출력하는 행위를 10번 반복하도록 만들었다. 처음에는 while문을 넣으려고 했을 때 오류가 굉장히 많았는데, 오디오 파일을 그때마다 새로 생성시키고 시간 간격을 두니 오류를 말끔하게 해결할 수 있었다.

가장 중요한 것은 앞에서도 말했듯이 두 모듈 간의 ‘파라미터 넘기기’이다. 파라미터가 잘 넘겨져야 두 코드 간의 연결고리가 생기기 때문이다. 처음에는 전역변수로 접근하려 했으나, 그냥 함수의 매개변수와 리턴값을 설정하여 파라미터 값을 보내는 것이 가장 효율적이고 현명한 방식이였다.

**3.3.2 음성안내 디스플레이를 통해 구현**

도서관 책의 위치를 음성을 통해 안내하고 디스플레이를 통해 지도 상의 위치를 표시하기 위해 다음과 같은 순서로 프로젝트를 진행했다.

1. 디스플레이 조립

2. 라즈베리파이 sd 카드에 Debian 설치

3. 라즈베리파이에 파이썬 3.6.1 설치

4. 필요한 사진과 앞서 구현한 코드 라즈베리파이에 내려받기

5. 음성 안내 코드 Error 해결

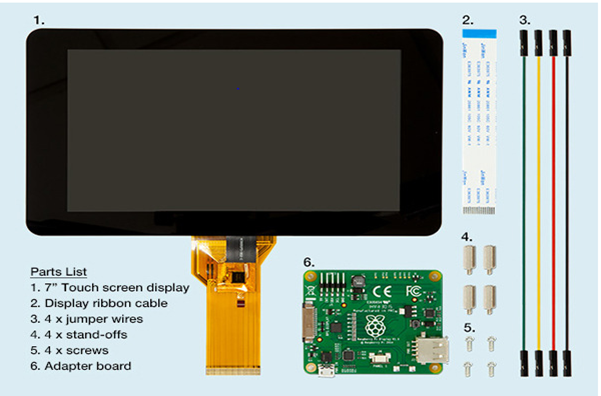
6. LSTM추가 시도

지도상에 박학다식 로봇의 현재위치와 이동경로를 rpm에 기반해 이동속도를 측정해 표시하려 했으나 시간 관계상 구현하지 못하고 목표위치를 표시해주는 기능만을 구현 할 수 있었다.

1. **디스플레이 조립**

라즈베리파이 공식 7inch 터치 디스플레이

<http://melonicedlatte.com/embedded/2017/11/25/172858.html>



1. **라즈베리파이에 Debian 설치**

설치 방법은 위의 디스플레이 조립에 첨부한 url 참고할 것

라즈베리파이에서 운용할 OS 선정을 위해 Ubuntu, Window10, Debian 모두 설치하여 진행

- Ubuntu : 설치과정이 어렵지 않으나 전체적으로 렉이 심하고 인터넷 속도가 매우 느리다.

- Window10 : 설치과정도 복잡하고 오래 걸리고 알 수 없는 오류로 인터넷조차도 안된다.

(ps. 라즈베리파이4에서는 Window10을 지원한다고 한다.)

- Debian : 라즈베리파이 전용 OS답게 속도가 가장 빠르다.

다만 install하고자 하는 패키지를 찾기 어려운 항목이 있다.

**음성안내의 핵심은 실시간 반응이므로 이에 가장 적합한 Debian을 사용하기로 결정**

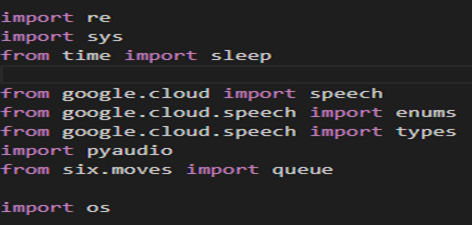
1. **라즈베리파이에 파이썬 3.6.1 설치**

<https://m.blog.naver.com/PostView.nhn?blogId=immaculacies&logNo=221036477034&proxyReferer=https:%2F%2Fwww.google.com%2F>

라즈베리파이에서 코드를 실행시키기 위해 파이썬을 설치한다. 당연한 말이지만 가장 중요한 부분이다. 위 과정을 제대로 수행하지 않아 디스플레이에서 코드를 실행시키는데 1주일이 소요됐다. (Debian설치 시 python2.7.13이 dafault로 설정되어 있다)

- 반드시 가상환경을 만들고 파이썬 새로운 버전을 설치해야 한다.

- 실행시킬 코드가 포함된 패키지를 지원하는 파이썬 버전을 정확히 설치한다.



음성안내 구현을 위해서는 GOOGLE speech, pyaudio, pygame 등 다양한 패키지를 필요로 하는데 GOOGLE speech의 경우 일정 버전 이상의 파이썬에서만 지원 되며 pyaudio의 경우 지원되는 버전이 몇 개 없다. 모든 필요한 패키지를 사용하는데 이상이 없는 파이썬을 가상환경을 만든 후 설치해 진행해야 검색해도 해결되지 않는 오류 문제를 최소화 할 수 있다.

1. **필요한 사진과 앞서 구현한 코드 라즈베리파이에 내려받기**

인터넷이 되기 때문에 구글드라이브 또는 메일을 통해 다운 받기만 하면 된다.

1. **음성 안내 코드 Error 해결**

마이크와 스피커를 설치해야 하는데 수많은 블로그를 찾아봤으나 usb 마이크는 구동이 되나 스피커가 구동이 안됐다. 결국 3.5mm잭을 이용한 스피커를 사용했다. **마음 편히 구글 AIY Voice kit를 구매**하거나처음부터 잭을 이용한 스피커를 구매하는 것이 좋다.

|  |  |
| --- | --- |
|  |  |
|  |  |

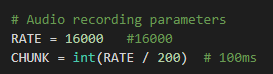
코드를 실행시키면서 위와 같은 에러를 엄청나게 많이 볼 수 있었는데

입력 마이크 또는 출력 스피커에 문제가 발생할 시 내장 프로그램인 ALSA ERROR가 발생한다.

디스플레이창에서 터미널을 켜 ‘nano ./asoundrc’ 명령어를 입력하고 아래와 같이 입력하면 된다는데 아무런 효과가 없었다.



해결방법은 코드에서 선언한 RATE = 16000을 RATE=44100으로 변경하는 것인데 라즈베리파이의 경우 특정 주파수(여기서는 44100)만 지원하기 때문에 다른 값을 쓰면 음성을 인식하지 못하고 오류가 나는 것이었다. 쭈니 ㅎㅇ



라즈베리파이에서 음성관련 프로젝트를 진행한다면 sample\_rate = 44100으로 설정하는 것을 기억하는 것이 좋다.

**3.3.3 LSTM 추가 시도**

음성 인식 부분에 LSTM을 추가로 구현하는 것을 시도했다. STT의 경우 다음의 문제점이 있을 수 있다. 발화자의 발음의 부정확할 경우 인식의 문제가 생길 수 있다는 점이다. 예를 들어 “한 여름밤의 꿈이 어디에 있는지 알려줄래?”라고 발화를 했을 때, 발음이 부정확할 경우 기계가 음성인식을 못할 가능성이 있었다. 이 경우를 대비하여 LSTM의 메커니즘을 삽입하려고 계획을 세웠다.

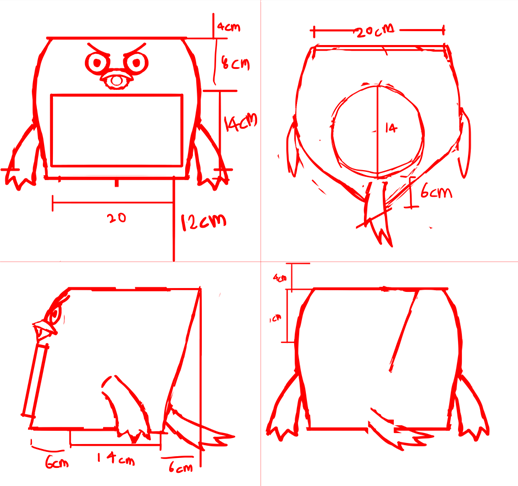
LSTM의 경우 미리 완성된 문장을 학습하면 앞, 뒤 글자와의 연관성을 파악하여 부정확하게 인식이 된 곳을 word correction해주는 기능을 갖고 있다. 예를 들어, “한 여름밤의 꿈이 어디에 씻는지 알려줄래?”라고 발음을 할 경우, LSTM은 기존에 학습된 “한 여름밤의 꿈이 어디에 있는지 알려줄래”를 통해 글자 ‘씻’을 ‘있’으로 바꿔줄 수 있다. STT의 경우 정해진 문장이 들어와야 정해진 대답이 나간다는 점에서 음성인식률을 높일 필요가 있었고, 그 수단으로 LSTM이 적절한 기능을 해줄 것으로 기대했다.

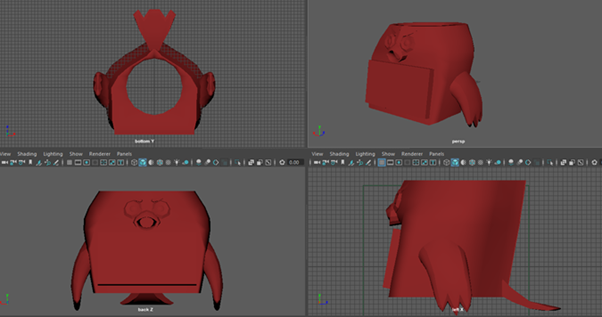
그러나 본 프로젝트가 차용한 기계인 라즈베리파이 속 라즈비안의 용적에 비해 LSTM의 무게가 너무 무거웠다. 인식률을 높이기 위해 LSTM을 넣는다면, 이로 인해 기계의 성능과 속도를 현저히 떨어뜨릴 위험이 있었고, 이 위험을 감수하면서까지 LSTM을 넣어야 하냐는 의견이 존재했다. 그러나 AI로 음성인식률을 높힌다는 아이디어자체는 충분히 의미가 있었기 때문에, LSTM을 넣지는 못하더라도 차선책으로 KoBERT, LightBERT의 탑재 가능성을 조사했다. KoBERT와 LightBERT는 새로 학습할 필요도 없이 이미 학습된 문장을 대상으로 word correction이 이뤄지기 때문에, 그만큼 시스템에 적은 부담을 준다는 장점이 있었다. 그러나, 이를 넣기에도 라즈비안의 용적이 매우 작았기 때문에 KoBERT와 LightBERT도 넣을 수 없었다. 결국 google api만의 음성인식률을 가지고 프로젝트를 진행하게 되었다.

**3.4 디자인**

**3.4.1 초기 3D 프린팅 디자인**

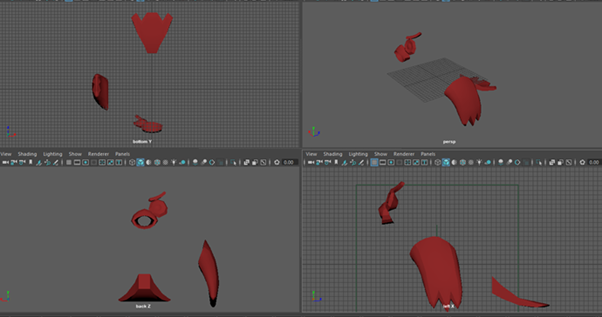
Turtlebot3에 포닉스를 모티브로 한 외관을 씌우기로 결정해서 3D 프린팅을 디자인하게 되었다. Turtlebot3는 총 6층으로 이루어질 것이고, 디스플레이를 입히기로 하였으므로 바퀴를 포함한 맨 밑의 2층과 LiDAR가 설치된 맨 윗층을 제외하고 외관을 디자인하게 되었다. 머리가 뚫린 디자인으로, 크기에 맞춰 3D 프린팅 디자인을 하였다.





**3.4.2 최종 3D 프린팅 디자인**

Turtlebot3 구동 문제로 Turtlebot3 구성 요소가 계속 수정되었고, 3D 프린팅 일정이 미뤄지게 되었다. 3D 프린팅은 출력 속도가 매우 느려 원래 제작하기로 한 28\*26\*26크기는 시간 내에 인쇄가 불가능할 거란 판단아래, 눈썹과 부리, 날개, 꼬리, 뒷통수 같은 필수 부분만 3D 프린팅으로 제작하기로 결정하였다. 나머지 부분은 아이클레이로 제작하게 되었다.



**4. 결론**

본 프로젝트는 음성인식과 자율주행기능을 탑재한 터틀봇으로 도서관 AI사서를 구현하는 것을 목표로 삼았다. 음성인식의 경우 참고문헌[3]을 토대로 파악한 결과 음성인식 API중에 Google API가 인식률이 가장 좋았기 때문에 이를 기반으로 구현했다. 이를 디스플레이로 구현하기 위해 라즈베리파이에 Debian과 Python을 설치했고, 음성인식 코드를 삽입했다. 이 과정에서 인식률을 조금이라도 더 높이기 위해 LSTM, LightBERT등의 딥러닝 모델을 추가하려고 했으나 라즈베리파이의 사양의 한계 때문에 실행에 옮기지는 못했다. 물론, 이 딥러닝 모델이 없어도 Google API는 70%이상의 음성인식률을 보여줬다. 도서관의 특성상 소음이 크지 않다는 점, 그리고 이 정도의 음성인식률이라면 발음의 부정확성이라는 변수 또한 커버가 된다는 점 또한 딥러닝 모델들의 필요성을 크게 부각시키지 못하는 요소가 되었다.

자율주행은 ROS를 기반으로 구현했다. ROS를 설치하는 단계에서 OS를 무엇으로 할 것인지에 대한 문제에 크게 봉착했다. 후보군으로 라즈비안, 우분투, Window가 있었고, 모두 설치하여 성능을 시험한 결과 라즈비안이 제일 적합한 것으로 판명되어 라즈비안으로 진행하였다. 이 라즈비안에 ROS를 설치하고, SLAM과 Navigation을 이용해 자율주행을 구현했다. 이 과정에서 wifi 인식률이 문제가 되기도 했는데, 서버 컴퓨터와 통신하는 수단이 wifi였기에 연결이 조금만 약해져도 기계가 작동을 멈추거나 속도가 느려졌다. 이러한 기계적인 한계는 추후에 보완할 문제이다.

디자인은 포항공대의 상징인 포닉스를 모티브로 삼아 구현했다. 최대한 포닉스의 특징을 유지하면서 기계에 덧입힐 수 있는 3D디자인을 고안했다. 이 중 눈, 코, 입, 날개 등 특징적인 신체 부위는 3D프린터로 인쇄하여 포닉스의 특징을 유지하였으나, 몸통은 지점토로 구현하여 포닉스를 재구성했다.

이러한 과정을 통해 9주간의 피와 땀이 서린 도서관 AI 사서 ‘박학다식’이 탄생했다. AI가 점점 발전됨에 따라 인간의 생활속 곳곳에 침투할 것으로 예상되는 바, 도서관에도 머지 않아 AI가 접목되는 시도가 이뤄질 것이다. 본 프로젝트는 그러한 시도에 첫 발을 내딛은 것에 그 의의를 둔다.

**5. 참고문헌**

[1] 노희경, & 이강희. (2017). 구글, 네이버, 다음 카카오 API 활용앱의 표준어 및 방언 음성인식 기초 성능평가. 예술인문사회융합멀티미디어논문지, *7*, 819-829.

[2] Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., & Konolige, K. (2010, May). The office marathon: Robust navigation in an indoor office environment. In 2010 IEEE international conference on robotics and automation (pp. 300-307). IEEE

[3] Google Cloud Speech API <https://cloud.google.com/speech-to-text/?hl=ko&utm_source=google&utm_medium=cpc&utm_campaign=japac-KR-all-ko-dr-bkws-all-super-trial-b-dr-1009137&utm_content=text-ad-none-none-DEV_c-CRE_266404458846-ADGP_Hybrid+%7C+AW+SEM+%7C+BKWS+~+T1+%7C+BMM+%7C+ML+%7C+1:1+%7C+KR+%7C+ko+%7C+Speech+%7C+google+speech-KWID_43700032535821335-kwd-440625966600&userloc_1030760-network_g&utm_term=KW_%2B%EA%B5%AC%EA%B8%80%20%2B%EC%8A%A4%ED%94%BC%EC%B9%98&gclid=Cj0KCQjw7ZL6BRCmARIsAH6XFDL7UqT7uI6kf99cAsOdWy5CMlRIFc1R6CN425RpInCGWEbxyG5FF7saApdwEALw_wcB>

[4] 음성인식, <https://blog.naver.com/chandong83/221066873320>

[5] Turtlebot, <https://emanual.robotis.com/docs/en/platform/turtlebot3/raspberry_pi_3_setup/#5-network-configuration>

[6]<https://github.com/ROBOTIS-GIT/turtlebot3/blob/master/turtlebot3_teleop/nodes/turtlebot3_teleop_key>