# 3DMapping

**Description of Program**

The zip file consists of neural network training file and testing file for the neural network.

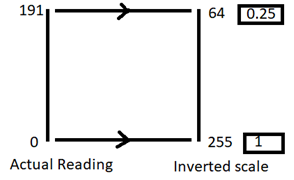
The map\_network.py and map\_network\_resnet.py are used to train the neural network using data in input\_b.csv and output\_b.csv . After training, hdf5 file will be created. Model will be stored in \_classifer.hdf5 file while weight will be stored in \_weight.hdf5 file.

The scan\_tile\*.py files will be used to test neural network using the 3D point cloud data stored in .pose and .3d data. These are the data taken in Leibnitz University.

The scan\_tile\*.py files are loading the pre-trained model stored in hdf5 files. It will then go thru the scan file one at the time. After extracting scan data in .pose and .3d file, adjust orientation using rotation matrix, conducting GPS offset at each scan data.

Preprocessing of data will be made on the x-y grid point which has multiple z values. (Taking the ground value).

Since light or waves can still be reflected from a scan point to the sensor through the space under objects such as tree leaves, these spaces are usually maneuverable. To prevent wrong prediction of space as object at high ground, we are inverting the height values according to figure below before entering neural network. This prevent MaxPooling2D to be more sensitive with the points with higher value.

[](https://github.com/optimusium/3DMapping/blob/main/scale_inversion.png)

The scan point will be analyzed to determine the slice number. The scan point will be fitted into 1024x1024 slices. In scan\_tile\*.py, map1 is 2D map consists of all scan points. Map1 will be used to interpolate the grid points in between scan points and sensor position.

Map2 is sliced map obtained from map1. Nominalized map2 will be input into neural network after salt noise reduction using maximum filter. Map2 will be backup as map5 and saved as mapim\*.png. If mapim\*.png has been generated in previous scan, the value stored in mapim\*.png file will be loaded back and concatenate with the new incoming points.

After prediction through neural network, the output will re-shaped back to original 1024x1024 slice size and scaled back with 256 as maximum value. The output (im in scan\_tile\*.py) will be compared against the input (in map5), the obstacle value in map5 will be retained. The output value will be stored in res\*.png (gray value).

Using 245 as thresholding value, pixel with value less than 245 will be marked with blue color. Pixel with value more than 245 will be marked as red color. The value is chosen in order to fit the expected value map. The colored map will be saved as im\*.png (note that value&lt;64 is considered to be unknown value, this is because we impose a clip at actual height of 191. After inversion, it will be reflected as 255-191=64)

After all the scan file pass through the process, the png files are merged. Res\*.png are merged as merged\_gray\_images.png. im\*.png are merged as merged\_images.png .

For edge\_noise reduction, padding and cropping are done using numpy.pad and numpy array function.

**Installation and execution**

1. Install anaconda3 if it is not installed yet
2. Import environment 3Dmapping using project\_environment.yml

Command:

conda env create -f project\_environment.yml

1. Activate the environment

Command:

conda activate 3Dmapping

1. If wants to run training, run map\_network.py (non RESNET) or map\_network\_resnet.py (RESNET)
2. If wants to run testing on scan point files, run scan\_tile\*.py

scan\_tile.py –without RESNET, without edge noise reduction

scan\_tile\_bound.py - without RESNET, with edge noise reduction

scan\_tile\_resnet.py –with RESNET, without edge noise reduction

scan\_tile\_resnet\_bound.py - with RESNET, with edge noise reduction

scan\_tile\_Kmeans.py - testing program for Kmeans clustering , K=7

**Python Environment File**

project\_environment.yml – Consists of anaconda3 environment and all the packages included

**Python Script List**

1. map\_network.py – training of neural network (non-RESNET)
2. map\_network\_resnet.py - training of neural network (RESNET)
3. scan\_tile.py – testing program for pre-trained neural network (without RESNET, without edge noise reduction)
4. scan\_tile\_bound.py testing program for pre-trained neural network (without RESNET, with edge noise reduction)
5. scan\_tile\_resnet.py – testing program for pre-trained neural network (with RESNET, without edge noise reduction)
6. scan\_tile\_resnet\_bound.py - testing program for pre-trained neural network (with RESNET, with edge noise reduction)
7. scan\_tile\_Kmeans.py - testing program for Kmeans clustering , K=7

**Data File List**

1. pose – Consists of GPS and IMU feedback (Euler angles) of vehicle at each scan.
2. 3d – Consists of scan point at each scan.
3. input\_b.csv – training data of neural network. (too large in size, go to this [link](https://drive.google.com/file/d/12FK79gaTQEVDxtx6LMv53039HnaOsNAi/view?usp=sharing))
4. output\_b.csv – training data of neural network. (too large in size, go to this [link](https://drive.google.com/file/d/14L1qxBQCsMJgZiPlPVUS7BAofWe2exCK/view?usp=sharing))

**PNG file (Remove all PNG files when running afresh)**

1. merged\_images.png – colored output (red is ground, blue is object)
2. merged\_gray\_images.png – colored output (gray values)

**hdf5 files**

1. map\_network\_classifier.hdf5
2. map\_network\_resnet\_classifier.hdf5
3. map\_network\_weight.hdf5
4. map\_network\_resnet\_weight.hdf5

# Visualization of 3DMapping

**Description of Program**

The visualization source codes are included at "visualization" folder.

The visualization\ViewerBackendService.py starts the backend service which enable the map tracking viewer application and 3D visualizer application to exchange data.

The visualization\TrackingViewer.py starts the map tracking viewer application. It reads the position and orientation data from visualization\view\config\tracking\_tasks.txt, then sends the position and orientation data to the backend service to simulate the real data captured in the sensors. It then waits for the 3D visualizer to generate the map/front/rear view images, and retrieves the map/front/rear view images from the backend service.

The visualization\Processed3dView.py starts the 3D visualizer application. It reads the visualization\data\processed\merged\_gray\_images\_reslyr\_bound.png which is generated by the scan\_tile\_resnet\_bound.py, then constructs 3D mesh from the gray image. It will read the position and orientation data from the backend service, and generate the map/front/rear view images with 3D mesh, and send back to the backend service.

The visualization\view\GenerateTasks.py will generate the position and orientation number to visualization\view\config\tracking\_tasks.tmp.txt. The tracking\_tasks.tmp.txt can then be renamed to view\config\tracking\_tasks.txt so to provide the data to the map tracking viewer application.

The view\config folder stores the base camera settings for various direction in "\*.json". The real camera settings will be adjusted based on the base camera settings and real orientation data.

The visualization\view\images folder stores the image for the car image which is used to represent the current robot/vehicle position in the map pictures. There are other images which are used as the temporary images for map/front/rear view images. These images can be used for debugging and troubleshooting.

The visualization\bin folder stores the batch script for starting all of the applicaations. The visualization\bin\start\_all.bat will start all 3 application at once, but requies to be editied to start the anaconda3 environment correctly.

**Installation and execution**

1. Follow the steps to setup anaconda3 environment as described at 3DMapping "Installation and execution" section.
2. Install the required modules below.

2-1. Activate the environment

conda activate 3Dmapping

2-2. Install Open3d

conda install -c open3d-admin open3d

2-3. Install pillow

conda install -c anaconda pillow

2-4. Install tqdm

conda install -c conda-forge tqdm

2-5. Install rpyc

conda install -c prometeia rpyc

1. Execute all the applications.

3-1. Activate the environment

conda activate 3Dmapping

cd visualization\bin

3-2. Start the backend service

start start\_backend\_services.bat

3-3. Start the map tracking viewer

start start\_map\_viewer.bat

3-4. Start the 3D visualizer

start start\_3d\_visualizer.bat

**Files List**

1. visualization\bin folder start\_3d\_visualizer.bat - starting 3D visualizer application start\_all.bat - starting all of application (Note: It is required to edit the settings inside) start\_backend\_services.bat - starting the backend service start\_map\_viewer.bat - starting the map tracking viewer
2. visualization\data\processed folder merged\_gray\_images\_reslyr\_bound.png - Input image file for the 3D visualizer
3. visualization\view GenerateTasks.py - creating the position and oritentation sample data file Processed3dView.py - the 3D visualizer application TrackingViewer.py - the map tracking viewer application ViewerBackendService.py - the backend service application
4. visualization\view\config base\_camera.json - base camera settings for any other direction base\_renderoption.json - base rendering settings east\_camera.json - base camera settings for the east direction front\_camera.json - base camera settings for front view map\_camera.json - base camera settings for map view north\_camera.json - base camera settings for north direction pinhole\_camera\_parameters.json - base camera settings for testing rear\_camera.json - base camera settings for rear view south\_camera.json - base camera settings for south direction tracking\_tasks.tmp.txt - positon and oritentation temporary file tracking\_tasks.txt - position and oritentation data file west\_camera.json - base camera settings for west direction
5. visualization\view\images car\_marker\_small.png - position marker image front\_image.png - temporary front view image map\_image.png - temporary map view image rear\_image.png - temporary rear view image