[[1]](#footnote-1)

Deep Neural Network Robotic Mapping based on 3D Scan

Ong Boon Ping, Cao Liang

Institute of System Science

*Abstract*—Currently, SLAM can be performed using various method such as Kalman filter, particle filter and K-Means method. With development of deep neural network, it is possible to train a neural network that can predict the mapping based on point cloud or lidar data. Deep neural network can be deployed on autonomous land or sea vehicle to predict the surrounding terrain.

*Keywords*—SLAM, K-Means

# INTRODUCTION

S

IMULTENEOUS Localization and Mapping (SLAM) for an autonomous robot moving in an unknown environment is to build a map of this environment while simultaneously using this map to compute its location [1]. K-Means clustering method is used on laser range sensor data for robot mapping [2].

Kalman filter and particle filter are also used in existing simulation or real time application. Recently, with the development of deep neural network, it is possible to create neural network that can predict the location of obstacle and mapping.

# Neural Network

## Idea

The convolutional neural network can approximate the mapping through spatial pooling and convolution filter. With the scan points, prediction can be made on unexplored grid point through the convolutional network.

In this project, we are converting scan points into 2D Numpy array before input to the convolutional network. The output will be in the form of Numpy 2D array that represents the predicted mapping.

## Dataset and Preprocessing

Based on the idea, scan data of 640 walled surfaces are gathered. Actual 2D maps serves as expected output while point cloud scan point maps as input during training.

In this neural network, pre-processing is required. This is to covert the grid points into 2D array with interpolations in between scan points and the sensor position.

The interpolation is limited to the grid point which is near to the ground level. Interpolations are not allowed for scan points with height greater than 20 pixels. This prevents wrong prediction of ground points as elevated points. It will remain as explored terrain.

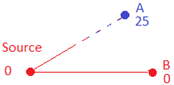


Fig. 1. Interpolation in between scan point to source. Interpolation is not made on points with height beyond 20 pixels since it is more likely to be a space rather than an obstacle. Full interpolation is made on ground or low-lying points during pre-processing.

## Design and Network Architecture

The design architecture consists of 2 phases. The first phase is interpolation area identifier and the following stage is the interpolation value refiner.

Fig. 2. Overall Design Architecture.

Interpolation area identifier focus on identification of unknown points that are in the interpolation bound. A point which is out of interpolation bound can be wrongly predicted based on nearest scan point value.

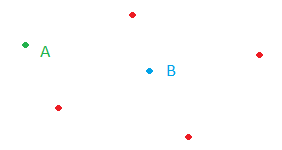


Fig. 3. Interpolation area identifier. Unknown point B is within the interpolation bound since the value can be predicted. Unknown point A is not predicted in this system since it is lack of interpolation information and not observable from source point.

To prevent the error, the neural network must have layers of MaxPooling2D and Conv2D filter layers with larger stride. MaxPooling2D layer allows maximum value in the 64x64 pixel area to be filled in with maximum values. Conv2D layer will generate different kernel at each neuron and hence create different possible 2D map at the Conv2D layer.

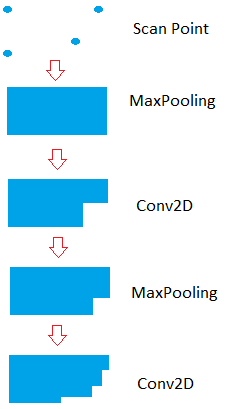


Fig. 4. Approximation of interpolated area after going through Max-Pooling and Conv2D filters twice

In this case, stage 1 is using 64x64 stride for the MaxPooling2D and Conv2D filter layers. After the training, the neural network can learn the trend and discount the pixels which are out of interpolation bound.

At stage 2, input is passing through 32x32 MaxPooling2D and Conv2D filter layers. On the other hand, output of stage 1 is upsized by 2x2 and concatenate with the output of stage 2.

At stage 3 to stage 5, it is using the same structure in stage 2. Hence, 5 stages of MaxPooling2D and Conv2D filter layers are used for the interpolation area identifier phase. Space with unknown value in the filter covered areas will obtain the predicted value closed to the nearest known value after going through the 1st phase of neural network.

Strides at each stage reduced so that the covering area is getting smaller at each stage. Spatial resolution of the interpolated area will become more accurate step by step. As a result, stride 64x64, 32x32,16x16,8x8,4x4 are used for stage 1 to stage 5.

Output of stage 5 is upsized by 4x4 and concatenate with the input. This concatenated array will be input into interpolation value refiner (Green block in Fig. 7).

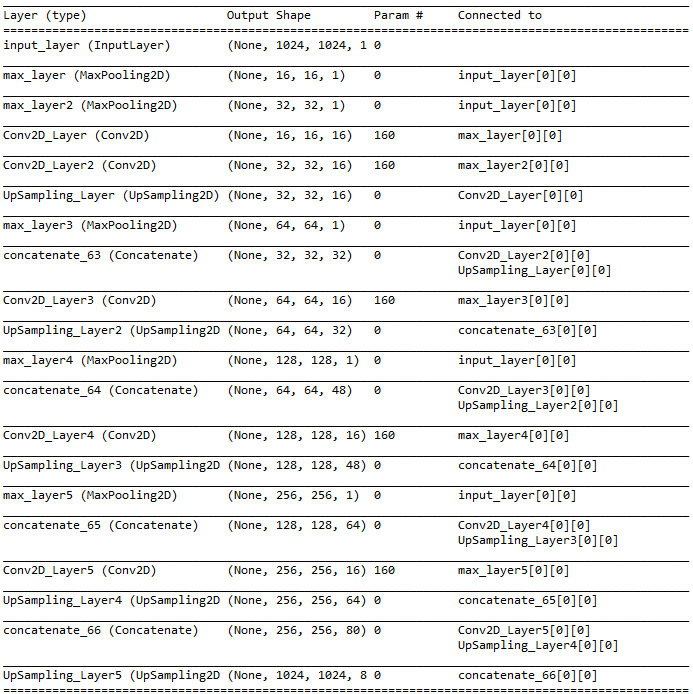


Fig. 5. Neural network layers from Stage 1 to Stage 5

Interpolation value refiner is used here to refine the predicted value. This is because MaxPooling2D network is giving maximum value for the neighboring pixels. The predicted value can be higher than the expected value.

Hence, at stage 6, MaxPooling2D filter is no longer be used at the final layers of the neural network. Conv2D are used in the final layers to smoothen the values across the neighboring grid points using strides 3x3. With small stride value, the points which is outside the interpolation area, will not be getting a prediction value at the final phase due to smooth action.

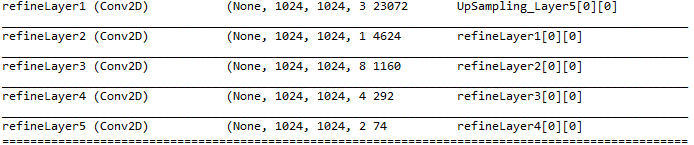


Fig. 6. Neural network layers from Stage 6

After few rounds of training, the design is finalized. The neural network accepts 2D array that contains scan point as input. To preserve the input scan point value, lambda layers are added so to replace the known point pixel value with the measured scan point data.

The final output will give final value or each grid point. To reduce the processing time, the map will be saved in the form of PNG images.

Upsampling

Upsampling

Upsampling

Upsampling

Upsampling

Fig. 7. Neural network Structure.

The design of stage 6 is further enhanced by changing the Conv2D with 6 residual blocks. Each residual block is having 2 Conv2D layers with 3x3 kernel. With this enhancement, the accuracy is increased because deeper network provides better approximation to the real terrain. Residual blocks also reduce overfitting and hence provide better performance than simple Conv2D layer [3]. As a result, RESNET enhancement of the neural network.

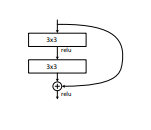
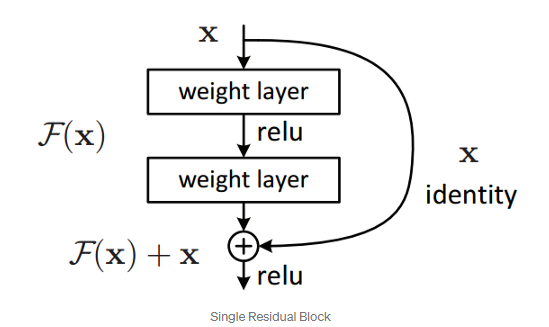


Fig. 8. Single residual block [3]. 6 residual blocks are used in the RESNET enhanced neural network.

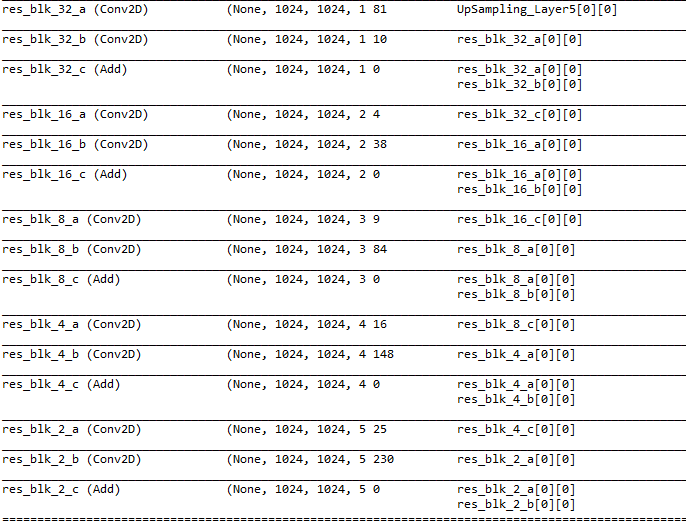


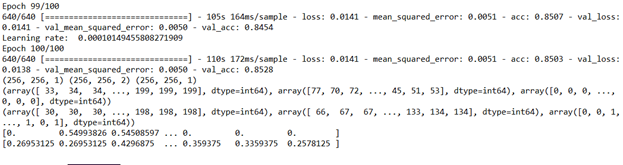
Fig. 9. Residual layers as improvement to stage 6 neural network.

# Network Training Result

The neural networks are trained using 640 2D Lidar maps. Each map is taken from lidar scan on wall and various building surfaces. The scan points are all collected from a single point cloud scan. The sensor is placed at different angle from the wall or building surfaces.

Each 2D lidar map has a matching terrain map. As a result, the 2D lidar map serves as input to the network while the expected output is the matching 2D terrain map.

After 100 epochs, the accuracy reaching 85% with mean squared error reduced to 0.0051. In this training, we are using RMSprop as Keras optimizer and root mean square as error metric.



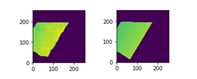


Fig. 10. Network Training Result. Output map at the left and expected output map at the right.

Similar training is also imposed on RESNET enhanced neural network. After 100 epochs, the accuracy reaching 86.2% with mean squared error reduced to 0.0044. In this training, we are using RMSprop as Keras optimizer and root mean square error as error metric. The RESNET enhanced neural network is having a better result than non-RESNET enhanced neural network.

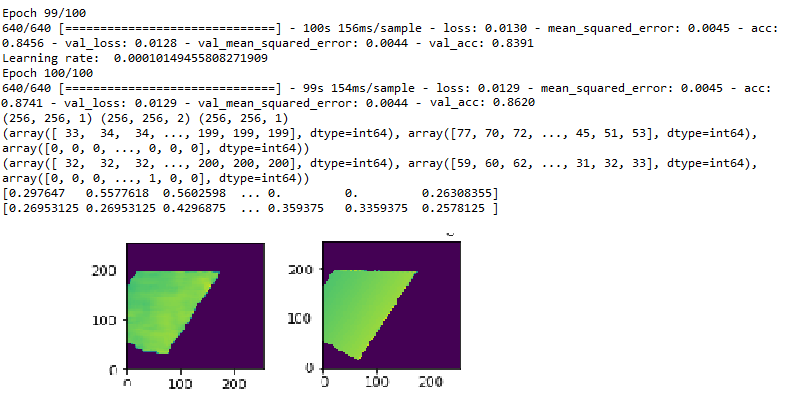


Fig. 11. RESNET Enhanced Network Training Result. Output map at the left and expected output map at the right.

The neural network weight and model are saved in hdf5 format. This will enable the network to be called during the testing stage.

|  |  |  |
| --- | --- | --- |
|  | Accuracy | RMS Error |
| Neural Network (without RESNET) | 85.0% | 0.0051 |
| RESNET Enhanced Neural Network | 86.2% | 0.0044 |

Table 1. Network Training Result. The RESNET enhanced neural network is having a better result than non-RESNET enhanced neural network

# Network Testing Result

The neural network is applied on the point cloud scan data gathered in Leibnitz University using autonomous robot. [4]

During testing, the network model and weights are preloaded during system initialization. Python script will be converting point cloud scan data into 2D map.

In this dataset, Euler angle and GPS position is appended along with all the scan point in local Cartesian coordinates. Hence, conversion is made using the rotation matrix. [5]

Interpolation is made on the newly created 2D map from the scan point to the source point.

After data conversion, preloaded neural network will be used to predict the terrain. Due to limited computing power and time limitation, input map will be processed in 1024x1024 pixel region. Output map will be saved as PNG image.

After all the regions are processed by neural network, all the PNG image will be combined into one single output terrain map.

Since there are multiple scan files, previous output saved in PNG image will be preloaded before subsequent scan points are loaded. The latest scan points will be overriding the existing values.

Fig. 12. Network Testing Process.

The neural network predicts the height value in unknown grid points. The testing accuracy is 93% when comparing the generated map against the expect mapping.



Fig. 13. Expected mapping.

Inaccuracy comes from noise reflected by the tree leaves and other objects. Due to slicing, certain bordering pixel is not showing intended pixel.

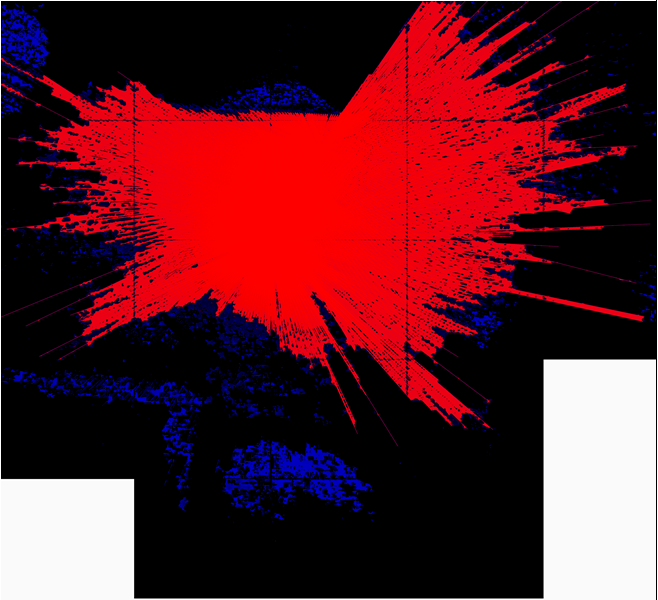


Fig. 14. Neural network generated map without edge noise reduction.

The error at the bordering pixel at the edge of 1024x1024 error can be resolved by padding. After padding all the edges with nearest values, the input and output shape become 1088x1088 pixels.

After each prediction, the padded edges are removed at the output. Hence, the output of 1024x1024 pixels will be returned. It will be free from noise at the edge of 1024x1024 boundary.

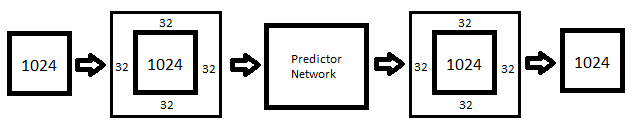


Fig. 15. Edge noise reduction

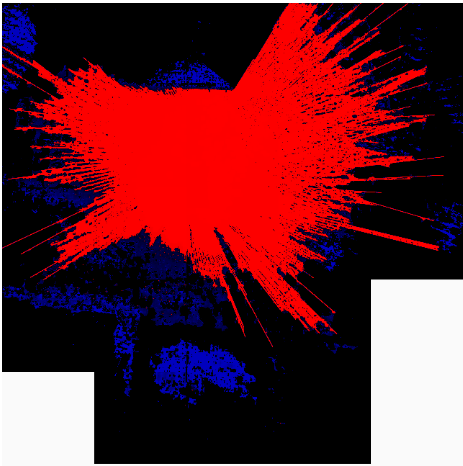


Fig. 16. Neural network generated map with edge noise reduction.

The RESNET refined neural network has a higher testing accuracy of 94.8% when comparing the generated map against the expect mapping.

Noise reflected by the tree leaves and other objects is reduced in the map generated by RESNET refined neural network. It is also more sensitive to border noise but it can be cleared by using padding on the input.

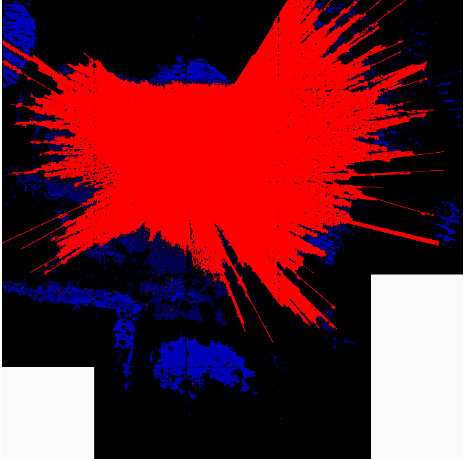


Fig. 17 RESNET enhanced neural network generated map without edge noise reduction.

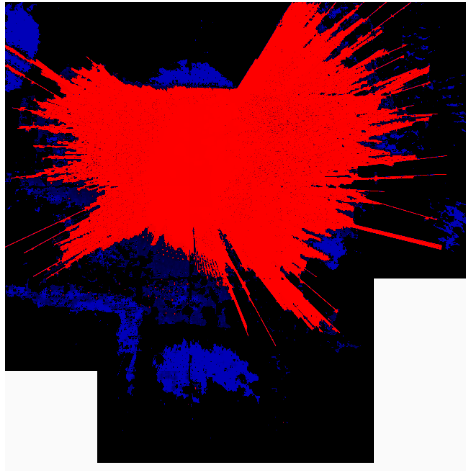


Fig. 18 RESNET enhanced neural network generated map with edge noise reduction.

K-means algorithm [2] is also used to generate the result. The testing accuracy is 70%. The method is considering the value of nearest 7 neighbors.

Hence, neural network can generate the 2D mapping better than the K-means filter on Lidar/Point cloud scan data.

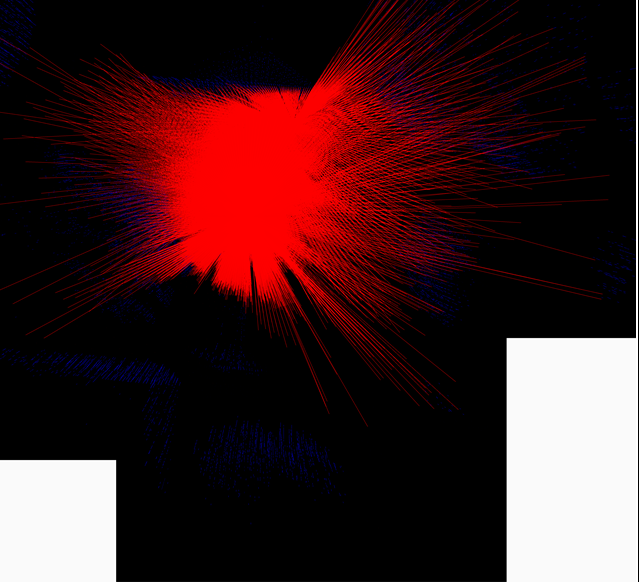


Fig. 19. K-means generated map.

# Comparison

## Background

The robot localization problem is vital important as it presents the estimate of the robot’s position and orientation. For the localization problem, a list of well-known solutions is available ranging from Adaptive Monte Carlo Localization (AMCL), Kalman Filter to Markov and Grid Localization. The AMCL is also referred as Particle Filter Localization, and is the most the most popular localization algorithms in robotics field. Therefore, we select AMCL as the comparison algorithm on the deep neural network robotic mapping.

## AMCL vs Deep Neural Network Robotic Mapping (DNNRM)

The Monte Carlo Localization (MCL) uses the particles to localize the robot’s pose. Each particle has its position and orientation which provide a chance for robot localization. Each time these particles are re-sampled when robot collects the sensor data from its environment. The AMCL is the improved version of MCL, since AMCL dynamically adjust the number of the particles over the period of time to achieve higher efficiency as the robot moves around the environment. In case the environment map is unknown, the AMCL does not perform well [6]. In open environments with less map features, the accuracy of ACML also decreases significantly [7].

The Deep Neural Network Robotic Mapping (DNNRM) relies on GPS to localize the robot, and produces the map based on the 3D scan data from LiDAR. In case the map was not available, the DNNRM will benefit AMCL with the more accurate map comparing the map generated with traditional probabilistic approach. In the other side, when GPS was not available to DNNRM, the ACML provides the reliable localization to DNNRM.

In summary, AMML and DNNRM are good candidates to complement each other on the localization and map tasks in certain conditions.

# Visualization

## System Design

The prediction output of the Keras neural network is visualized using Open3D library and Tkinter python packages.

The Map Tracking Viewer GUI sends robot position and orientation data to the backend services through Rpyc remote procedure call. The Open3D Visualizer generates the Map/Front/Rear view based on robot position and orientation, and subsequently sends back the Map/Font/Rear view to the backend service. The Map Tracking Viewer finally receives the Map/Front/Rear view and display them on the GUI.



Fig. 20. Overall Architecture.

## Open3D Visualizer

The Open3D Visualizer creates the 3D mesh from the 3D map generated by the DNNRM. The 3D map uses gray scale value to represent the height.

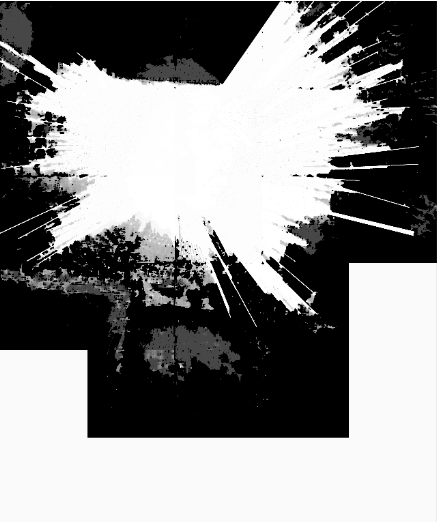


Fig. 21. 3D Map.

The Open3D Visualizer reads the robot position and orientation from the backend services, and generates the 2D Map, Front / Rear view pictures dynamically using Open3D library. Finally, the 2D Map, Front / Rear view pictures are sent back to the backend services.

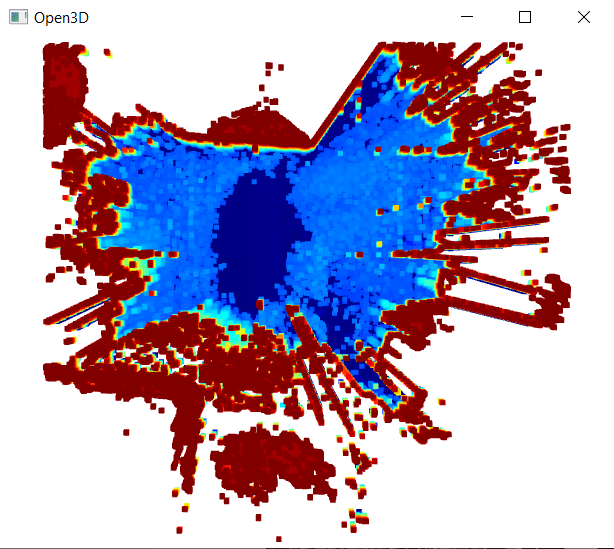


Fig. 22. Open3D Visualizer.

## Map Tracking Viewer

The Map Tracking Viewer reads the robot position and orientation from the text file to simulate the real-time robot movement. The robot position and orientation is sent to the backend services. The Map Tracking Viewer retrieves the 2D Map, Front / Rear view pictures from the backend service, and finally displays the view pictures on the GUI.

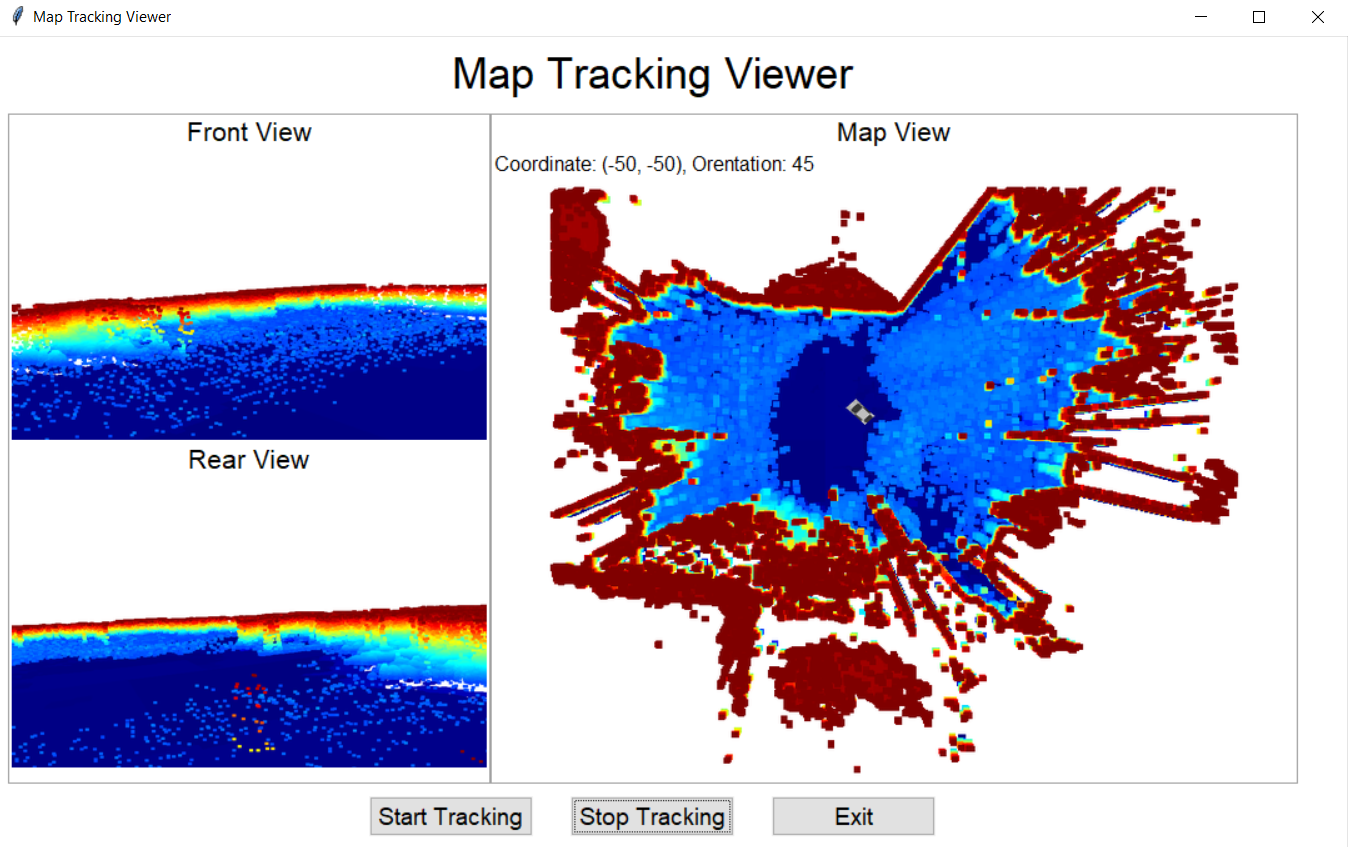


Fig. 23. Map Tracking Viewer.

# CONCLUSION

The Keras neural network can be used to make prediction on the surrounding terrain based on the lidar or point cloud data. With appropriate training with actual lidar scan data, the accuracy is comparable to the existing method such as K-Means method. The output of the Keras neural network can be used for further 2D-3D visualization and allow autonomous vehicle driver to have a better information on the terrain map and predicted view of the surrounding.

References

1. S. Soro and W. Heinzelman, “A survey of visual sensor networks,” Adv.

Multimedia, vol. 2009, pp. 1–22,

1. *Ravankar, Ankit & Hoshino, Yohei & Emaru, Takanori & Kobayashi, Yukinori. (2012). Robot Mapping Using k-means Clustering Of Laser Range Sensor Data. Bulletin of Networking, Computing, Systems, and Software. 1. pp-9.*
2. *Kaiming He & Xiangyu Zhang & Shaoqing Ren & Jian Sun. (2012). Deep Residual Learning for Image Recognition.*
3. *Andreas Nüchter, Kai Lingemann, and Joachim Hertzberg, Hartmut Surmann (2007).* *6D SLAM—3D Mapping Outdoor Environments*. *Journal of Field Robotics 24(8/9), 699–722*.
4. Yan-Bin Jia *(2020).* *Rotation in the Space. http://web.cs.iastate.edu/~cs577/handouts/rotation.pdf.*
5. Sagarnil Das (2018). Robot localization in a mapped environment

using Adaptive Monte Carlo algorithm. https://www.academia.edu/37292915/Robot\_localization\_in\_a\_mapped\_environment\_using\_Adaptive\_Monte\_Carlo\_algorithm.

1. Miguel Ángel de Miguel, Fernando García and José María Armingol (2020). Improved LiDAR Probabilistic Localization for

Autonomous Vehicles Using GNSS. https://www.mdpi.com/1424-8220/20/11/3145.

1. [↑](#footnote-ref-1)