A statistical estimation analysis of indoor positioning WLAN based fingerprinting

D.I. Nastac, F.A. Iftimie, O. Arsene, and C. Cherciu

Faculty of Electronics, Telecommunications and Information Technology,  
POLITEHNICA University of Bucharest, Romania

nastac@ieee.org

*Abstract*— The indoor positioning is a new topic in today's navigation and positioning research fields, which presents quite various challenging issues. A statistical approach to treat Wireless Local Area Network (WLAN) based fingerprinting using Received Signal Strengths (RSS) is described here. This study structures this approach as a time-series of the RSS and position data. Kalman filter is applied in order to improve the position prediction based on the measured RSS signal.

Keywords—Kalman filter, Indoor positioning, WLAN based fingerprinting.

# Introduction

An indoor positioning system could be useful as an end-user application for a person that is interested to find its location in a large building. It must be first mentioned that it is very difficult to estimate a precise position using commercial Access Points (AP) as the RSS fluctuates with a large variance for the same coordinate. In this work, we first experienced with public datasets [1], then created our own using a laser telemeter. In order to map the RSS values to the coordinates in the 2D space, we kept the order of our recordings, we trained a regression neural network and the last step was to apply a Kalman filter [2] on the ordered set to improve the accuracy of localization.

# Solution

## Neural network architecture

To estimate the x and y coordinates given the vector of RSS values, the following configuration of the network was implemented. The input was a vector of 172 RSS values followed by 4 hidden layers with 1200,400,100 and 20 neurons each with ELU activation function and a layer with 2 outputs. The ELU activation function was preferred because it offers more non-linearity, it does not easily kill the gradient and it permits the presence of negative activations. sIn order to prevent overfitting, [3] recommended regularization techniques such as dropout and weight regularization. We applied dropout on the last layer during training which significantly increased the accuracy.

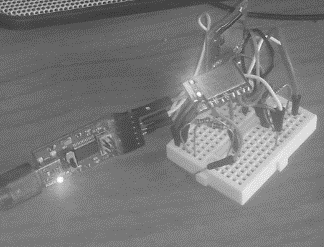
While training, dropout regularization technique was applied on the every layer of the network, but this did not worked very well because the dropout usually works best on the last layer of the network by making the first layers as feature extractors and the last layer the classifier. We also tried to apply dropout on the last layer which significantly increased the accuracy.

The training time is short as the datasets were not very large, and the machine had an Intel i7 CPU with 8 cores and a Nvidia GTX 960M with 640 CUDA cores. The training session was occasionally interrupted to modify hyperparameters such as learning rate and dropout rate. Initially the learning rate was high in order to converge faster. When the supervisor noticed that the cost was not decreasing anymore the learning rate and dropout probability were decreased.

The training was stopped when the validation set cost was not decreasing anymore.

## Methods for data collecting

In order to acquire accurate coordinates, we needed some sort of device that can measure long range distances. One such device is a laser telemeter depicted in the figure below. The technique to measure the data consisted in positioning the devices in the figure below near each other on a mobile platform then operating alternatively on each device in order to locate the position and save the vectors of RSS values. This approach is useful only in rectangular indoor environments. This approach is also a very slow one as the average time for collecting 10 measurements for a single location took 35 seconds. We had the patience to collect 3000 measurements for 300 locations.



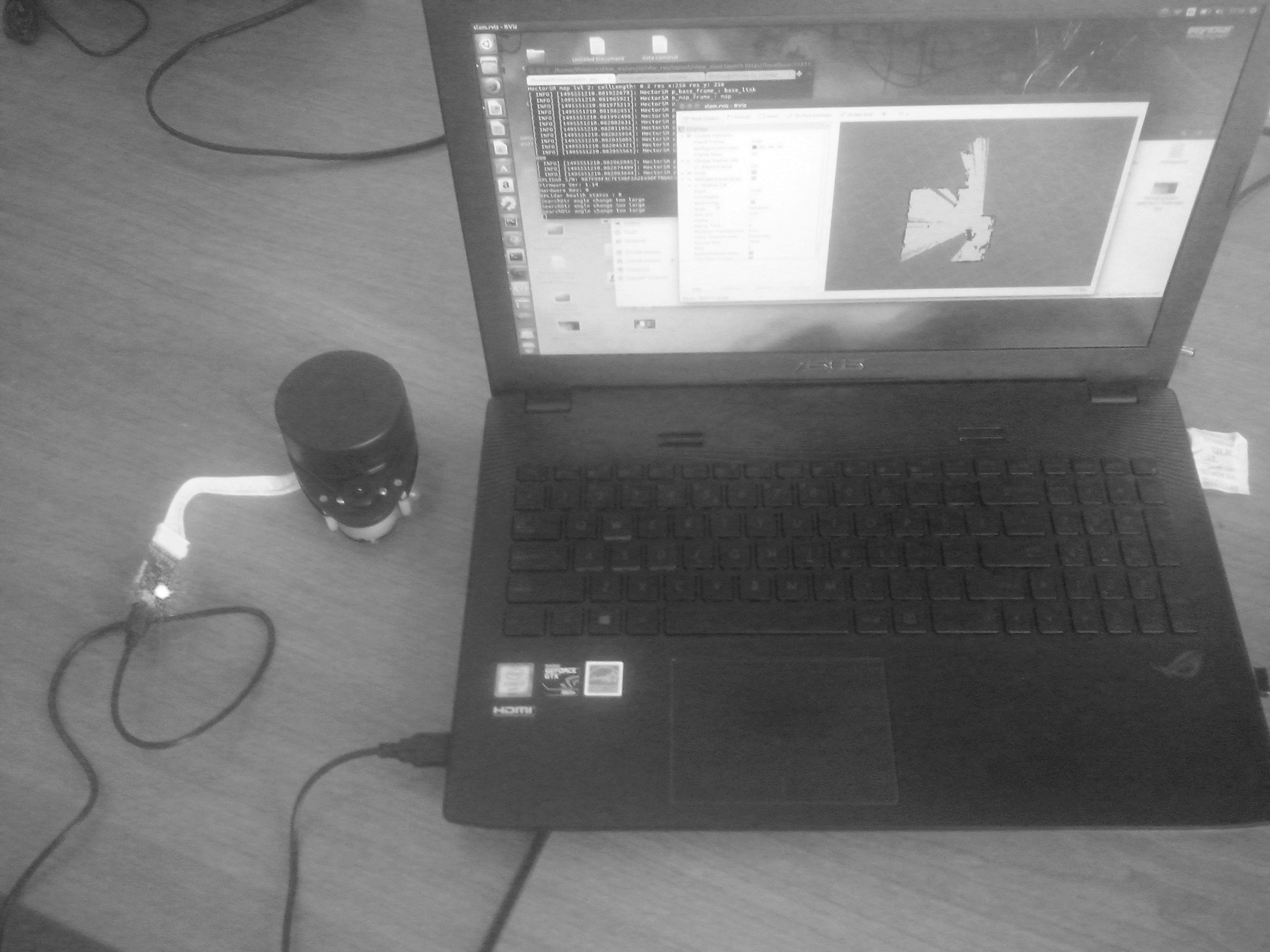
**Fig. 1:** Laser telemeter and WIFI scanner

We chose one hallway and we measured across it 3 times, each time with a distance difference in the latitude of around 1 meter as depicted in the figure below. For the longitude, we moved with a step size of 0.2 meters and for each location we recorded 10 RSS observations. While we were measuring we had a WIFI scanner that was writing to a file the RSS values. In Figure 2 is depicted the path we took for collecting measurements along the walls of a hallway.



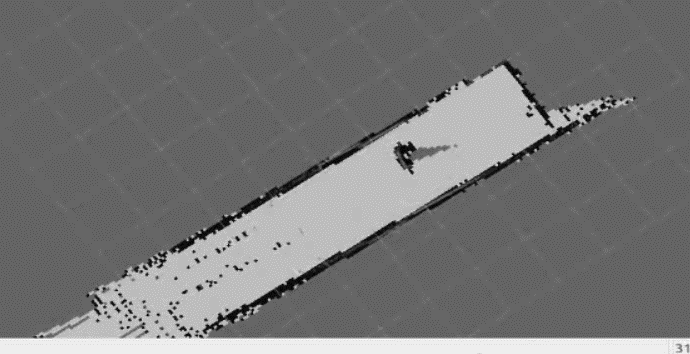
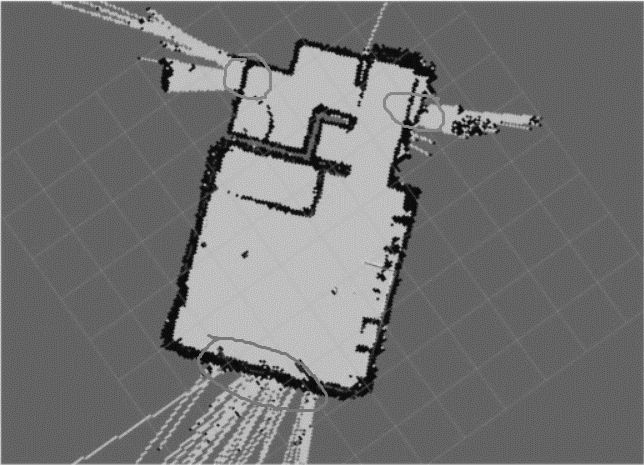
**Fig. 2:** Path for collecting RSS

Our second attempt was using a lidar. RPlidar[3] is a 2D lidar manufactured by RoboPeak with a distance range of around 8 meters, an angular resolution of 1 degree and 5.5hz scan rate. It comes with a ROS package that makes it easy to integrate it into a robot. The package also contains 2d map reconstruction algorithms specifically hector slam that work very well in a detailed indoor location.



**Fig.3:** RPLidar

We created a node in ROS [4] that subscribed to the topic where the coordinates computed by the hector slam algorithm were published. The same node was listening on the serial port for RSS signals coming for a WIFI scanner in order to create our training dataset. The pair of RSS and coordinates were written to a file. We had two ways to create the same dataset, one of them included a JAVA server that was reading the RSS signals from the serial port and another one integrated all programs into the ROS architecture. We are still deciding which one to choose.

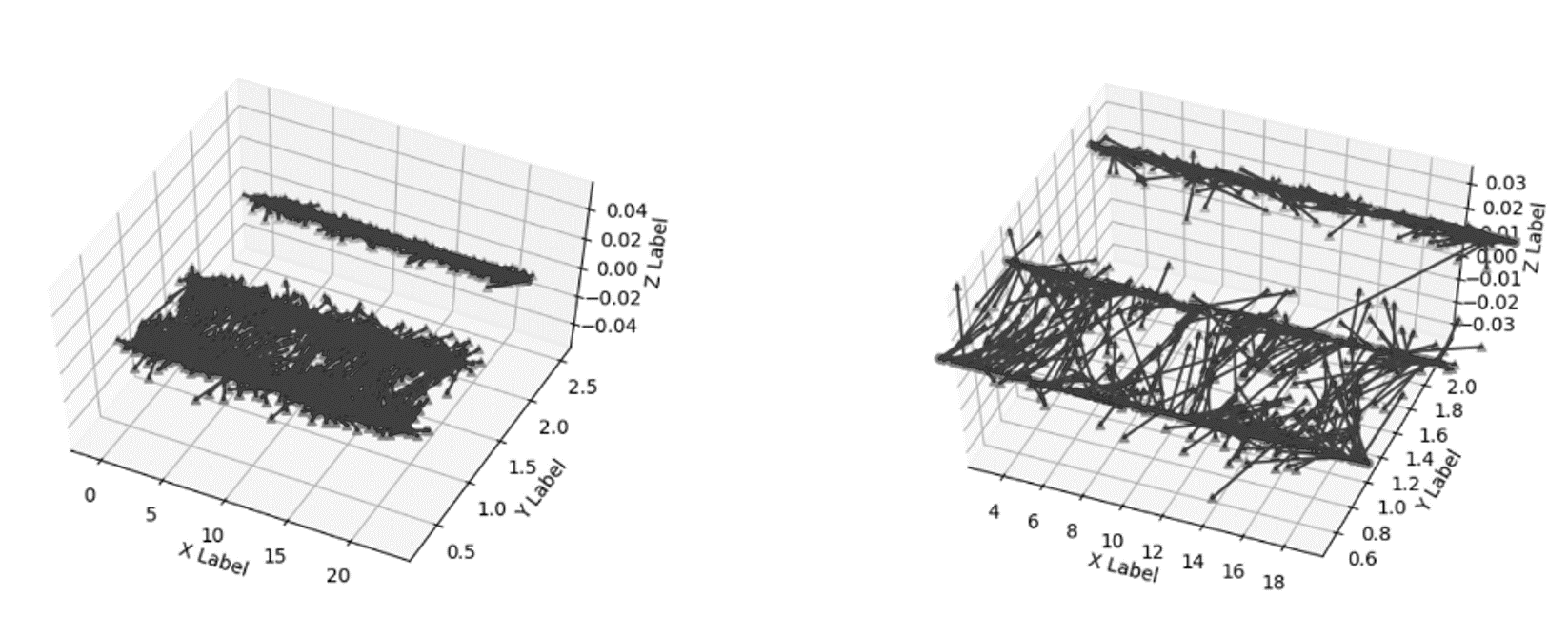


**Fig.4:** 2D Reconstruction using RPLidar

One problem that we found using this RPLidar was that in a long hallway this device alone would not work because there are not enough features and variety for accurate 2d reconstruction. In Figure 4 are found the reconstructions of two different indoor environments of a building. The first one, to the left is a student dorm room and it has a lot of features and the reconstruction is done easily. There are however certain points in the room with glasses or mirrors and some infrared rays are reflected more and the reconstruction over those points is noisy. In the second graphic of the same figure there is the reconstruction of a long featureless hallway. Although the graphic does not look bad, in reality it is because the length of the hallway was a lot longer. The few doors that covered the wall did not contained enough information in order to generate accurately the map. This is why a robot must integrate also other sensors like wheel odometer or depth vision in order to combine their power together for an accurate localization algorithm.

## Data preprocessing

The dataset was split into 3 parts. A training set with 80% of the records, a validation set with 10% of the records and a test set with 10% of the records. We augmented 8 times the dataset by cloning the observations on the training set and adding small gaussian variations both on the RSS signals and the x, y coordinates. There is a trade-off between the gaussian variance and the final accuracy thus we empirically fine-tuned them in order to achieve the best accuracy. In order to normalize the data, only the inputs and the outputs of the training set were used to fit standard scalers. The scalers were later used to transform the inputs and the outputs of the validation set and the test set. Because the Kalman Filter needs the inputs to be ordered in time, the dataset was left complete and not shuffled. The network was still trained by separating the dataset and shuffling, but when applying the Kalman Filter, the data was left untouched.



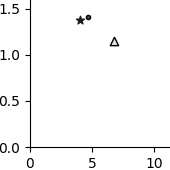
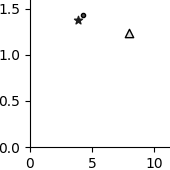
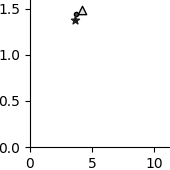
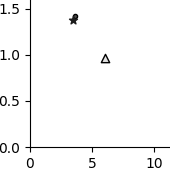
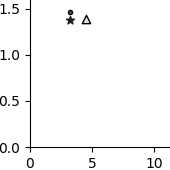
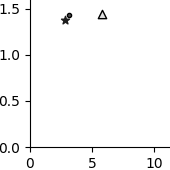
**Fig. 3:** Noisy neural network estimation. Training set and test set.

## Improving the estimation of the network by using a Kalman Filter

The Kalman Filter is a mathematical model that is used to filter out noisy data. It tracks an observed variable and a hidden variable. In our project, we assumed that the user of our application would be moving in time, thus the observed variable was the 2D coordinate, and the hidden variable was the velocity vector. The model works in two steps. It has a prediction step, and an update step. The prediction step is used to estimate the next position by using the inferred velocity vector, and an update step is used change the state variable with the new measurements. The core idea of the Kalman Filter is to choose whether to trust more the prediction or the measurement by using a self-adjusting variable called Kalman gain.

In our project, we assumed that the user of our application would be moving in time, thus the observed variable was the 2D coordinate, and the hidden variable was the velocity vector. As it can be seen in the Fig. 3, the results of the regression neural network are very noisy. Thus, in our project we wanted to trust more the prediction step, by fine-tuning the ratio between the standard error in the measurement, and the standard error in prediction.

Because the Kalman Filter needs the inputs to be ordered in time, the dataset was left complete and not shuffled. The network was still trained by separating the dataset and shuffling, but when applying the Kalman Filter, the data was left untouched.

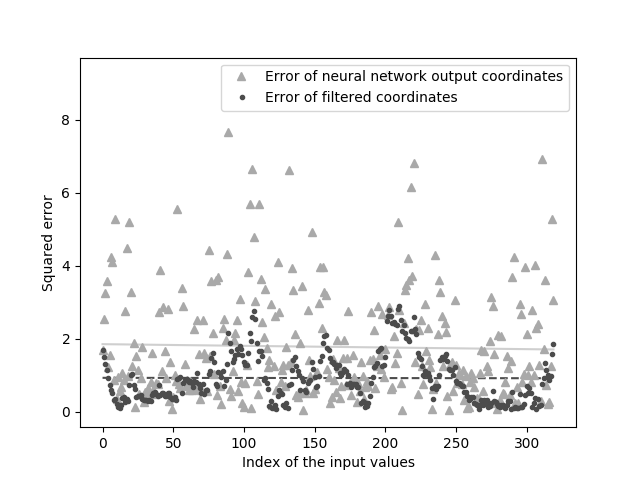


**Fig. 4:** Applied Kalman Filter

# Results

The accuracy was computed as the number of estimated coordinates that were in the range of 1 meter from the ground truth coordinates over the total number of observations. As the neural network outputs were standardized, the inverse transformation of the outputs was applied with the scaler described above in order to bring back the real values that will be used for computing the accuracy. We obtained an accuracy of 97.51% on the training set and 62.30% on the test set. In the figure above, with the triangle marker is the estimation of the neural network in 2D coordinates, with the star marker is the ground truth location and with the circle is the filtered coordinate. Figure 4 consists as a collage of screenshots taken from a video, and it can be imagined that the triangle marker was constantly bouncing, and the circle marker was staying quite close to the blue dot. It can be observed that the estimation converges much easier to the ground truth location.

Using the Kalman filter we also tested on the ordered test set that contained 1/10 of the observations of the training set with a reported 77.57% accuracy. Also, the training set is reported with a 97.68% accuracy which is just a 0.17% increase.



**Fig. 5:** Error comparison with filtration and without filtration

In figure 5 are drawn the error values for each input value in the test set. The Kalman Filter has a great impact over the precision of the model. From this graphic, the filter manages to highlight some problems of the model. There are three modal peaks that may express the fact that the network is biased towards the centroid of the coordinates. A solution to this problem would be to increase the number of the training data for the locations where the measurement device distances from the center of the indoor environment. We are still researching this issue.

# Future work

Robot Operating System represents a collection of packages and algorithms written and maintained by professionals and university members in order to facilitate the development of a robot. It is a layer over Linux and its architecture is based on the publisher subscriber paradigm. The publishers, the subscribers and the messages exchanged between them are governed by a general server named roscore.

Starting from the premise that a robot has multiple sensors and multiple actuators, these components are called nodes in the ROS architecture. Each node is actually a Linux process that communicates with the roscore server through a TCP connection using topics. A topic is a data structure where the messages between nodes are stored. Each node can be a subscriber or a publisher or both at the same time. The advantage of this architecture is that the developer can add or remove multiple devices to robot with the minimum modification of the source code.

For example, a robot may have a lidar and the developer might want to make it explore autonomously a building. There are packages that facilitate the creation of a node that publishes the lidar data to a scan named topic. From that scan topic, a node that recreates the 2d map of the building is taking the lidar data, processes it and publishes the 2d reconstruction on a topic named map. From that topic, another node can use the map data in order to control the wheels of the robot for the autonomous navigation.

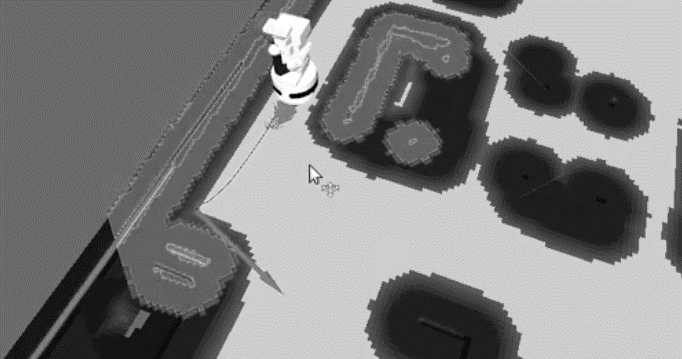
Our Tiago robot manufactured by Pal Robotics is used in order to overcome the constraints of the RPLidar as it integrates a set of sensors such as Lidar, RGBD camera and wheel odometer. It is delivered with software packages that are able to fuse the data of the sensor for better localization. It also comes with a simulator that can help understanding it's architecture, the correct way to operate with it and avoid possible damages.

Following the tutorials provided with the simulator, we found that it is not well documented and we tried to reverse engineer it in order to find more about how to control the robot from written code, autonomously.

In the first tutorial [4], we were introduced to the way the robot constructs the 2D map using its sensors specifically, a lidar, a wheel odometer and RGBD camera. In this tutorial, the user must specify a desired location into the unknown space and the robot will navigate towards it until it reaches it or will find a wall blocking the way. In the same time, the robot localizes itself using the sensors and reconstructs the map using Hector slam algorithm. After the user decided that the map has been fully explored, it can be saved for later usage. In addition to the 2D map, the robot is able to create a cost map that takes into consideration the surfaces that should be avoided given the center position of the robot and its radius in order not to touch any obstacles.

Thus, the map must be manually constructed using the utility program Rviz that helps visualizing the process of 2D reconstruction.

In the next tutorial, the user is introduced to the way the robot uses the previously built map for localization using a Particle Filter [5] as the robot position can be randomly initialized. In the same way, using the utility program Rviz, the user can specify a destination using Rviz and the robot will navigate towards if there is a path available.



**Fig 6:** Tiago robot covering every patch of the map

Given the map, and the robot capacity of navigation, we created a ROS node that is able to cover every 30 〖cm〗^2 the map, in order to uniformly collect WIFI RSS signal for our training dataset.

This is more a development stage that will help us collect better data in order to improve our initial model.

##### References

1. Measurement data for wireless positioning, URL: http://www.cs.tut.fi/tlt/pos/Software.htm, 2016
2. Narayan Kovvali; Mahesh Banavar; Andreas Spanias, "An introduction to Kalman Filtering with MATLAB", Morgan & Claypool Publishers, 2013
3. Andrej Karpathy, Justin Johnson, “Convolutional Neural Networks for Visual Recognition”, URL: http://cs231n.github.io/neural-networks-1, 2016
4. Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew Ng, “ROS: an open-source Robot Operating System”, ICRA Workshop on Open Source Software, 2009