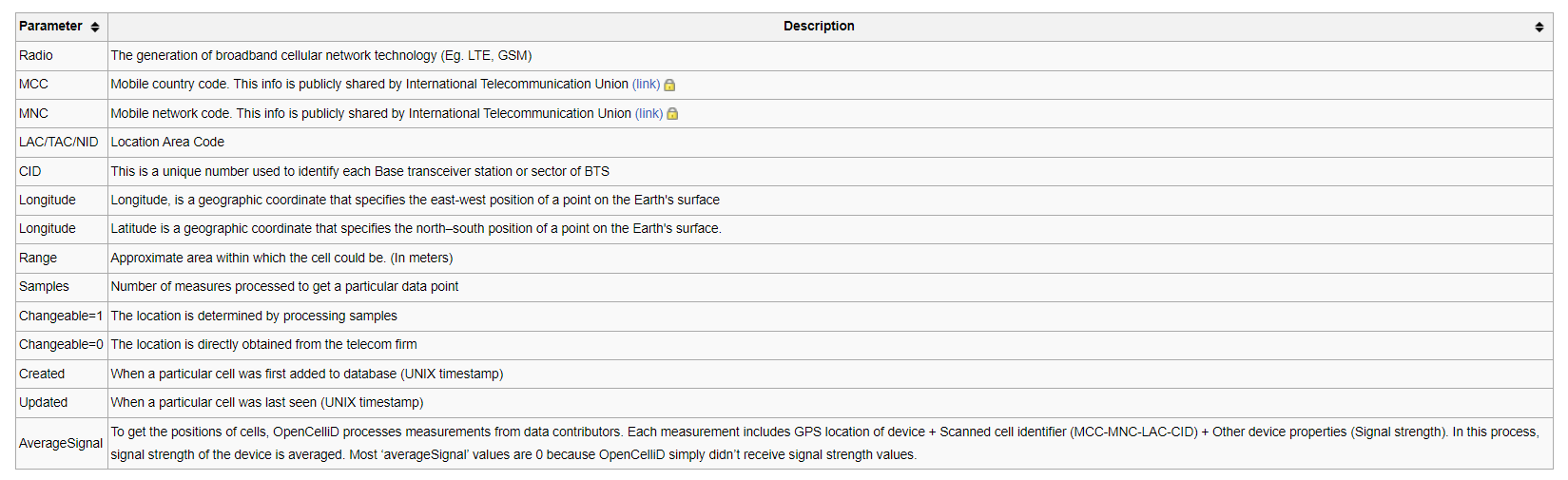
**Report**

**1) Multilateration + Kalman Filter**

**1.1) Dataset**

The main dataset used in this method is [OpenCellid dataset](https://opencellid.org/#zoom=16&lat=37.77889&lon=-122.41942). The columns in the dataset are shown in the table below.



The measurements made by the nrf9160dk board have the following configuration.

"measurements": [  
 [  
 "%NCELLMEAS: 0,\"02A5B601\",\"26201\",\"5815\",15,1300,253,46,28,27497,6400,181,33,11,32,25111",  
 "OK%NCELLMEAS: 0,\"02C9FA07\",\"26201\",\"5815\",208,6400,181,33,11,38377,6400,290,32,9,0,39591"  
 ],  
 [  
 "%NCELLMEAS: 0,\"02C9FA07\",\"26201\",\"5815\",210,6400,181,40,21,65897,1300,253,46,28,23,500,218,33,28,32,63911",  
 "O%NCELLMEAS: 0,\"02A5B601\",\"26201\",\"5815\",16,1300,253,46,26,79111,78151"  
 ],  
 [  
 "%NCELLMEAS: 0,\"02A5B601\",\"26201\",\"5815\",16,1300,253,50,30,111337,6400,181,38,17,37,109191",  
 "OK%NCELLMEAS: 0,\"02C9FA07\",\"26201\",\"5815\",208,6400,181,39,21,122217,123031"  
 ]

]

Each measurement instance has one or more elements depending on the number of neighbor cell towers. In the above example, there are two cell towers. Each measurement has the following structure.



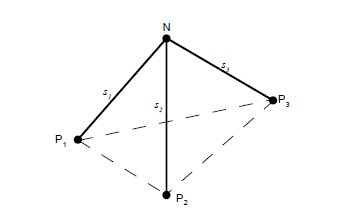
**1.2) Algorithms**

**Multilateration:** Multilateration is the calculation of the unknown coordinates of a target using distances to reference points with known coordinates. To calculate the distances, different methods can be used: RSSI, ToA, TdoA, etc. Each anchor position and distance pair produce a circle/sphere (for 2D / 3D space) equation that demonstrates the possible locations of the target. The target can be localized by finding the intersection point of these possible locations. 2D and 3D spaces require minimum 3 and 4 equations to localize the target, respectively.

The circle/sphere equations are converted into a matrix equation in the form of Ax = b. This system has a general solution in the form of:

is found by multiplying by the pseudoinverse of matrix. is found by solving and k is found by the constraints on . If there are minimum of 3/4 anchor positions in 2D/3D space and if they are not collinear, becomes zero vector since the null space of A becomes 0. Then, the solution can be found by using only the pseudoinverse.

In 2D, this procedure can be shown as follows:



Given 3 anchor locations that are not collinear (P1, P2, P3) and the distances (s1, s2, s3) the target location N can be found by solving the following system of equations:

Then, this equation system is written in matrix form as . The solution is found by multiplying b by pseudoinverse of A.

2D Multilateration in Python:

def multilateration(s, P):  
 anchor\_num = P.shape[1]  
 dimension\_num = P.shape[0]  
  
 #multilateration algorithm doesn't work for 2 anchor positions, average of the anchors is calculated instead  
 if anchor\_num == 2:  
 direction\_vec = P[:, 1] - P[:, 0]  
 direction\_vec /= np.linalg.norm(direction\_vec)  
  
 p0 = P[:, 0] + s[0] \* direction\_vec  
 p1 = P[:, 1] - s[1] \* direction\_vec  
  
  
 average\_point = (p0 + p1) / 2  
 return np.array([np.linalg.norm(average\_point)\*\*2, average\_point[0], average\_point[1]])  
  
  
 if dimension\_num == 2:  
 A = np.zeros((anchor\_num, 3))  
 b = np.zeros((anchor\_num, 1))  
  
 for i in range(anchor\_num):  
 A[i, :] = np.array([[1, -2 \* P[0, i], -2 \* P[1, i]]])  
 b[i, :] = np.array([[s[i] \*\* 2 - P[0, i] \*\* 2 - P[1, i] \*\* 2]])  
  
 x = np.linalg.inv(A.T @ A) @ A.T @ b  
 return x

**Kalman Filter:** Kalman filter is an estimation algorithm that is used for estimating the state of the system in the presence of uncertainty in measurements. The system is modelled as a matrix equation and the noise in the system is modelled as Gaussian:

Where:

* is state vector, F is state transition matrix and
* is process noise with covariance Q. The state vector is defined as

Measurements are modelled as:

Where:

* is observation vector, H is observation matrix,
* is measurement noise with covariance R

In this project, I used a library called filterpy. filterpy implements the update equations given the matrices defined in the update equations. I implemented a function to construct the matrices according to a motion model with constant acceleration. Measurement uncertainty is controlled by measurement\_sigma variable. It is assumed that distance measurements in x and y dimensions are uncorrelated. Model uncertainty is controlled by sigma\_a variable, representing the standard deviation of acceleration.

def get\_kalman\_matrices(measurement\_sigma=1, dt=1, sigma\_a=1):  
 F = np.array([[1, dt, 0.5 \* dt \*\* 2, 0, 0, 0],  
 [0, 1, dt, 0, 0, 0],  
 [0, 0, 1, 0, 0, 0],  
 [0, 0, 0, 1, dt, 0.5 \* dt \*\* 2],  
 [0, 0, 0, 0, 1, dt],  
 [0, 0, 0, 0, 0, 1]  
 ], dtype=float)  
  
 H = np.array([[1, 0, 0, 0, 0, 0],  
 [0, 0, 0, 1, 0, 0]], dtype=float)  
  
 R = np.array([[measurement\_sigma\*\*2, 0],  
 [0, measurement\_sigma\*\*2]], dtype=float)  
  
 Q = sigma\_a \*\* 2 \* np.array([[dt \*\* 4 / 4, dt \*\* 3 / 2, dt \*\* 2 / 2, 0, 0, 0],  
 [dt \*\* 3 / 2, dt \*\* 2, dt, 0, 0, 0],  
 [dt \*\* 2 / 2, dt, 1, 0, 0, 0],  
 [0, 0, 0, dt \*\* 4 / 4, dt \*\* 3 / 2, dt \*\* 2 / 2],  
 [0, 0, 0, dt \*\* 3 / 2, dt \*\* 2, dt],  
 [0, 0, 0, dt \*\* 2 / 2, dt, 1]], dtype=float)  
  
 return F, H, R, Q

**1.3) Advantages – Disadvantages**

Advantages:

* Localization can be done without prior data collection.
* Prediction uncertainty can be calculated

Disadvantages:

* Prediction accuracy depends on the used cell tower location dataset
* Cell tower might not exist in database
* Number of detected neighbor cell towers can be low (multilateration algorithm doesn’t work)
* Distance calculated by the device is not high resolution (step size: 78.125 m )

**2) Learning from Data**

**2.1) Dataset**

The dataset is generated in four steps:

1. Measurements of the nrf9160dk board are captured via the capture\_measurement.py script. To use the script run script on the project root folder.

python3 capture\_measurement.py

To change the filename, the filename variable can be adjusted.

1. GPS measurements are recorded with the [GPS Logger application](https://play.google.com/store/apps/details?id=eu.basicairdata.graziano.gpslogger&hl=en&gl=US). The application produces recordings with the extension kml. This file should be put into the raw\_measurements folder. (Measurement accuracy ≈ 10m)
2. Board measurements and GPS measurements are combined with the construct\_dataset.py script. Construct dataset script needs measurement\_filename (file produced by the capture\_measurement.py), coordinate\_filename (file recorded by the GPS tracker application), and dataset\_filename (output file name). To run the script run the following command inside the single\_measurement folder:

python3 construct\_dataset.py

1. Data points are divided by a grid through the grid page of the application. In “Grid Adjustment” mode select the file generated by the construct\_dataset.py script from the file selector. Select the length of each grid element in the “Grid Length” slider and select the length of top and left edge lengths. Adjust the position of the grid by “Top Left Longitude” and “Top Left Latitude” sliders. When all of the data points are inside the grid, press the “Save” button to generate the dataset files. Dataset files are generated in the “datasets” folder.

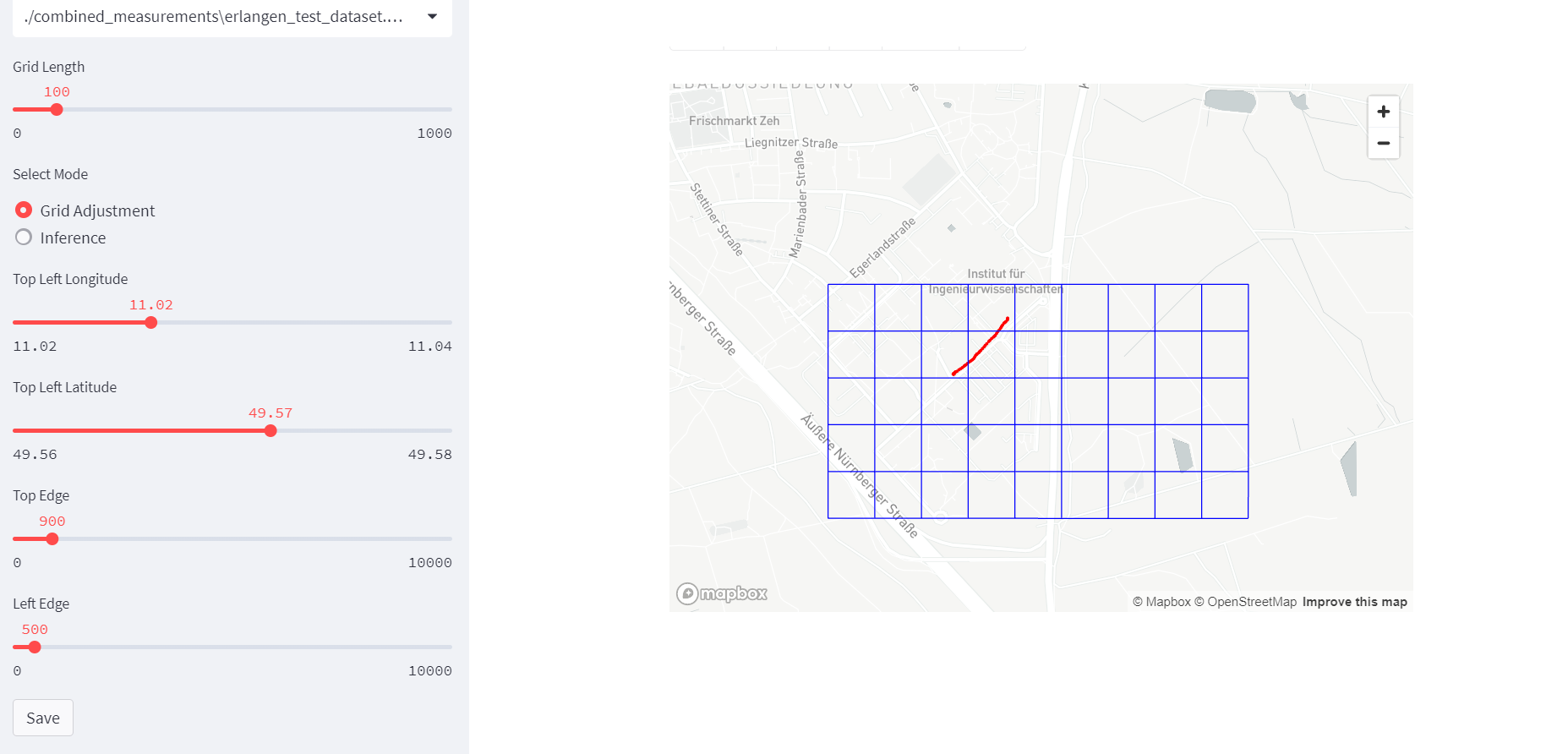


Figure : Grid page in grid adjustment mode

Three files are generated: first CSV file contains the data required for training with column names: c*urrent\_phys\_cell\_id,current\_rsrp,current\_rsrq,1\_phys\_cell\_id,1\_rsrp,1\_rsrq,2\_phys\_cell\_id,2\_rsrp,2\_rsrq,3\_phys\_cell\_id,3\_rsrp,3\_rsrq,4\_phys\_cell\_id,4\_rsrp,4\_rsrq,5\_phys\_cell\_id,5\_rsrp,5\_rsrq, longitude, latitude,original\_index*. The second CSV file contains label data with columns: *row, col, idx, longitude, and latitude*. The last file is a JSON file which contains the grid layout for visualization during the inference mode.

**2.2) Used Models**

**1) Multi-Layer Perceptron**

This model is composed of 4 fully connected layers with neuron sizes (64, 128, 64, output\_size). In between layers ReLU activation function is used. Network predictions are obtained by the softmax activation function at the last layer. Due to the softmax, outputs are obtained as a probability distribution over the grid.

**2) LSTM**

This model is composed of 1 LSTM layer with 32 hidden dimensions and 1 fully connected layer with (output\_size) neurons. Pack Propagation Through Time algorithm is used during training, meaning that gradient updates are limited. 5 or 10 data points are used during training. Network predictions are obtained by the softmax activation function at the last layer. Due to the softmax, outputs are obtained as a probability distribution over the grid.

**3) Random Forest**

This model is composed of 100 decision tree classifiers. Decision trees are trained with sampled datasets from the original dataset to reduce overfitting. The Gini coefficient is used for fitting the function.

**Note:** output\_size is a parameter dependent on Grid Length

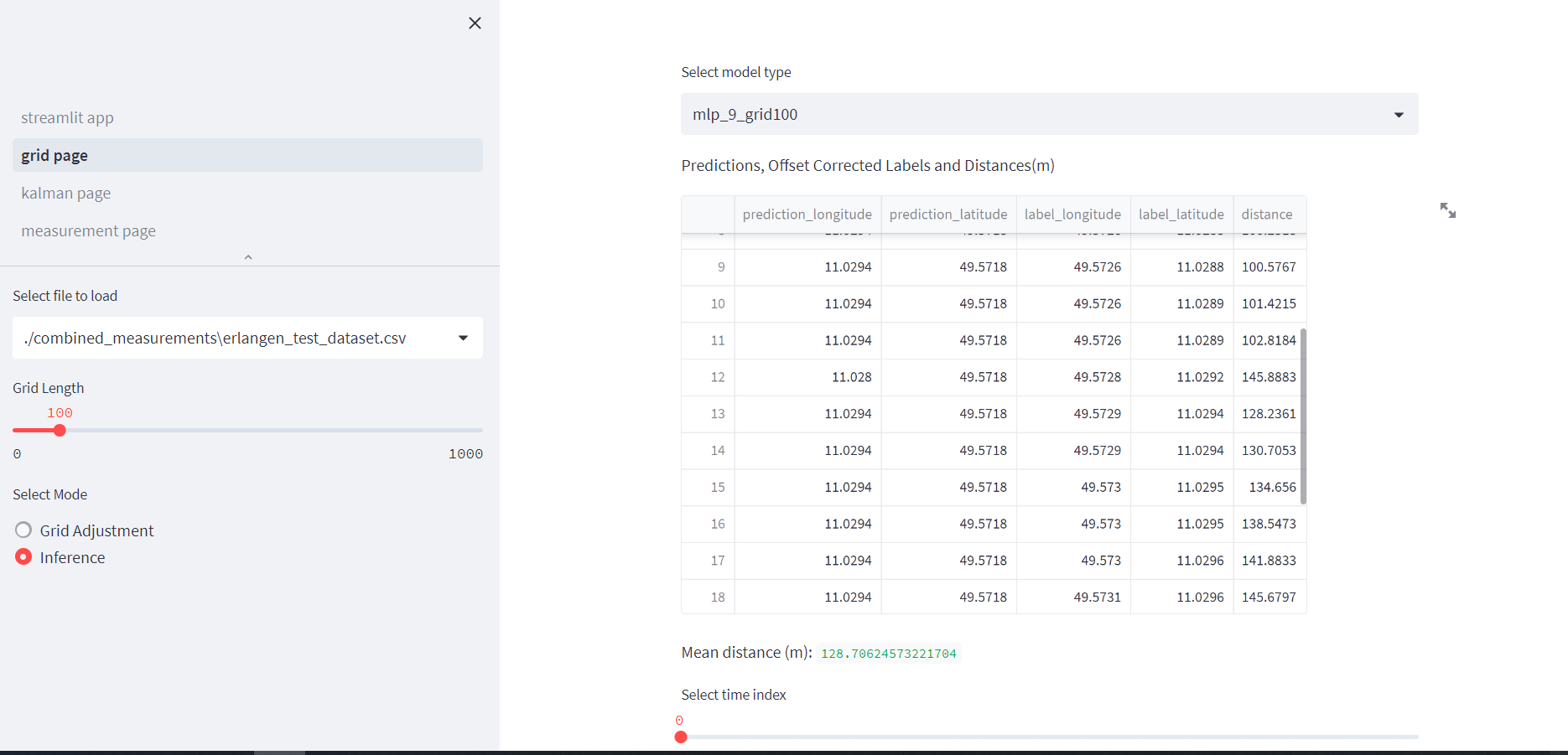
**2.3) Training Configurations**

Each model is trained with a training dataset and a validation dataset. The training is finished when validation loss starts to diverge from the training loss.

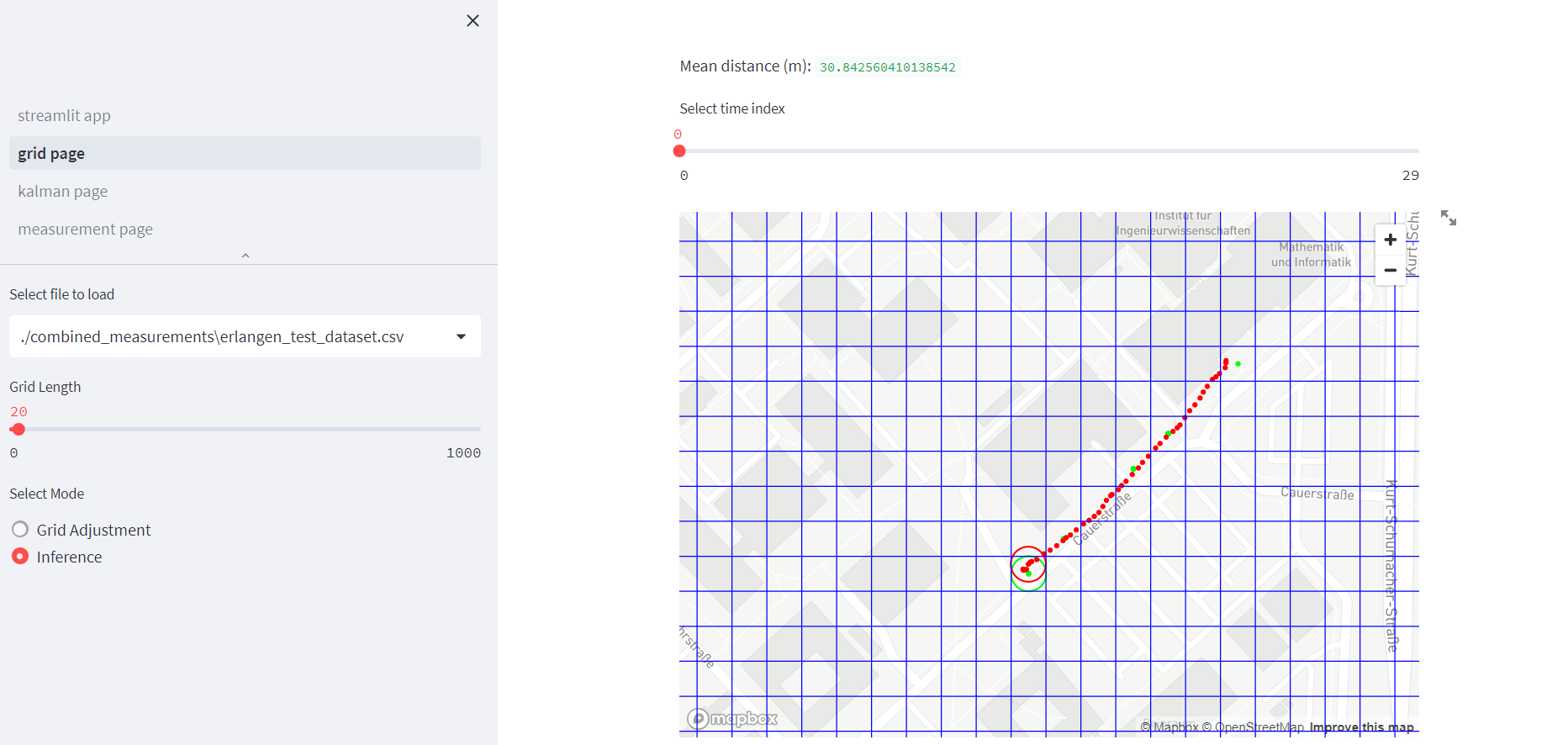
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model Name | Epochs | Batch Size | Output Classes | Optimizer | Learning Rate | Data Points |
| Multi-Layer Perceptron with 1 data point (Grid Length: 100m) | 300 | 128 | 64 | Adam | 0.001 | 1 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 100m) | 20 | 128 | 64 | Adam | 0.001 | 5 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 100m) | 65 | 128 | 64 | Adam | 0.001 | 10 |
| Multi-Layer Perceptron with 1 data point (Grid Length: 50m) | 60 | 128 | 256 | Adam | 0.001 | 1 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 50m) | 29 | 128 | 256 | Adam | 0.001 | 5 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 50m) | 29 | 128 | 256 | Adam | 0.001 | 10 |
| Multi-Layer Perceptron with 1 data point (Grid Length: 20m) | 65 | 128 | 1600 | Adam | 0.001 | 1 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 20m) | 21 | 128 | 1600 | Adam | 0.001 | 5 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 20m) | 67 | 128 | 1600 | Adam | 0.001 | 10 |
| LSTM with 5 data points (Grid Length: 100m) | 600 | 128 | 64 | Adam | 0.001 | 5 |
| LSTM with 10 data points (Grid Length: 100m) |  | 128 | 64 | Adam | 0.001 | 10 |
| LSTM with 5 data points (Grid Length: 50m) | 67 | 128 | 256 | Adam | 0.001 | 5 |
| LSTM with 10 data points (Grid Length: 50m) | 100 | 128 | 256 | Adam | 0.001 | 10 |
| LSTM with 5 data points (Grid Length: 20m) | 820 | 128 | 1600 | Adam | 0.001 | 5 |
| LSTM with 10 data points (Grid Length: 20m) | 123 | 128 | 1600 | Adam | 0.001 | 10 |

**2.4) Test Results**

Outputs of the models are displayed in the “grid page” menu under the “Interference” option. From “Select file to load” menu the dataset file that will be used for testing is selected. After that “Grid Length” parameter is selected. Finally, model is selected form the “Select model type” menu.

****

Model names are constructed as model type + “\_” + the number of features + “\_” grid length + “\_” + the number of time steps used. The table under the model selection menu shows the individual predictions’ coordinates and ground truth(label) coordinates. Also, distances between predictions and labels are shown under the “distance” column.

****

After the model selection, the model predictions and the test dataset are loaded into the map as shown above. Ground truth values are represented by red and predictions are represented by green. Predictions at each time step can be seen by adjusting the “Select time index” slider. The selected time index’s coordinates are circled.

Models are trained on a dataset that contains 8605 data points recorded by the capture\_measurement.py script. Tests are performed on a dataset with 486 data points. Tests are performed with and without outlier removal. The outlier removal algorithm removes data points with predictions that are 30 meters away from the average of the 3 last predictions.

**a) Without probability weighting and outlier removal**

Table : Model performances without outlier removal

|  |  |
| --- | --- |
| **Model Type** | **Mean Error (meter)** |
| Multi-Layer Perceptron with 1 data point (Grid Length: 100m) | 142.3 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 100m) | 73.7 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 100m) | 79.3 |
| Multi-Layer Perceptron with 1 data point (Grid Length: 50m) | 90.9 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 50m) | 101.9 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 50m) | 88.3 |
| Multi-Layer Perceptron with 1 data point (Grid Length: 20m) | 93.8 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 20m) | 67.2 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 20m) | 75.6 |
| LSTM with 5 data points (Grid Length: 100m) | 94.0 |
| LSTM with 10 data points (Grid Length: 100m) | 71.5 |
| LSTM with 5 data points (Grid Length: 50m) | 115.2 |
| LSTM with 10 data points (Grid Length: 50m) | 59.7 |
| LSTM with 5 data points (Grid Length: 20m) | 93.1 |
| LSTM with 10 data points (Grid Length: 20m) | 33.4 |
| Random Forest with 100 decision trees (Grid Length: 20m) | 55.4 |

**b) ) Without probability weighting and with outlier removal**

Table : Model performances with outlier removal

|  |  |  |
| --- | --- | --- |
| **Model Type** | **Mean Error (meter)** | **Percentage of Remaining Data Points** |
| Multi-Layer Perceptron (Grid Length: 100m) | 128.7 | 48.1 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 100m) | 39.7 | 32 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 100m) | 67.2 | 17 |
| Multi-Layer Perceptron with 1 data point (Grid Length: 50m) | 50.2 | 17 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 50m) | 46.6 | 16 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 50m) | 36.1 | 25 |
| Multi-Layer Perceptron with 1 data point (Grid Length: 20m) | 38.4 | 21 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 20m) | 28.4 | 29 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 20m) | 17.0 | 23 |
| LSTM with 5 data points (Grid Length: 100m) | 53.2 | 19 |
| LSTM with 10 data points (Grid Length: 100m) | 71.7 | 29 |
| LSTM with 5 data points (Grid Length: 50m) | 68.6 | 21 |
| LSTM with 10 data points (Grid Length: 50m) | 31.1 | 28 |
| LSTM with 5 data points (Grid Length: 20m) | 77.9 | 25 |
| LSTM with 10 data points (Grid Length: 20m) | 30.8 | 30 |
| Random Forest with 100 decision trees (Grid Length: 20m) | 26.1 | 44.2 |

**b) With probability weighting and without outlier removal**

LSTM and MLP models give probability distribution over the grid positions. Each grid element has a probability associated with it. In this method, these probabilities are used to combine multiple cells’ positions. In order to do so output probabilities are sorted and grid positions with the largest k probabilities are used to obtain the final prediction. Each grid position is multiplied by its probability and then summed up.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model Type** | **k=1** | **k=2** | **k=3** | **k=4** | **k=5** |
| Multi-Layer Perceptron (Grid Length: 100m) | 142.3 | 142.0 | 142.0 | 142.0 | 142.0 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 100m) | 73.7 | 64.2 | 64.1 | 64.1 | 64.0 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 100m) | 79.3 | 76.9 | 77.0 | 77.1 | 77.1 |
| Multi-Layer Perceptron with 1 data point (Grid Length: 50m) | 90.9 | 91.5 | 89.5 | 90.5 | 89.8 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 50m) | 101.9 | 92.8 | 88.4 | 89.1 | 88.5 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 50m) | 88.3 | 80.2 | 78.6 | 76.7 | 76.9 |
| Multi-Layer Perceptron with 1 data point (Grid Length: 20m) | 93.8 | 98.4 | 97.5 | 98.0 | 99.8 |
| Multi-Layer Perceptron with 5 data points (Grid Length: 20m) | 67.2 | 63.4 | 62.4 | 61.7 | 62.0 |
| Multi-Layer Perceptron with 10 data points (Grid Length: 20m) | 75.6 | 77.1 | 76.5 | 77.6 | 77.2 |
| LSTM with 5 data points (Grid Length: 100m) | 94.0 | 87.0 | 84.0 | 85.1 | 86.1 |
| LSTM with 10 data points (Grid Length: 100m) | 71.5 | 68.2 | 72.3 | 72.4 | 73.0 |
| LSTM with 5 data points (Grid Length: 50m) | 115.2 | 128.2 | 131.0 | 135.3 | 144.2 |
| LSTM with 10 data points (Grid Length: 50m) | 59.7 | 68.4 | 71.0 | 72.0 | 71.5 |
| LSTM with 5 data points (Grid Length: 20m) | 93.1 | 80.2 | 93.8 | 107.9 | 116.2 |
| LSTM with 10 data points (Grid Length: 20m) | 33.4 | 39.6 | 42.6 | 46.6 | 55.8 |

Using probability weighting provided a slight performance improvement on the "Multi-Layer Perceptron with 5 data points (Grid Length: 50m)" model. However, it didn’t provide any performance improvements over other models. This might be due to the training method of the networks. Networks are trained with labels that are one-hot encoded. This means the probability distribution is 1 at one point and 0 elsewhere. Due to this, networks learn to give a peak value and try to minimize other points. During the minimization, the shape of the distribution is not important as long as it is minimized. This might be the reason behind the lack of performance improvement when using points other than the peak value.