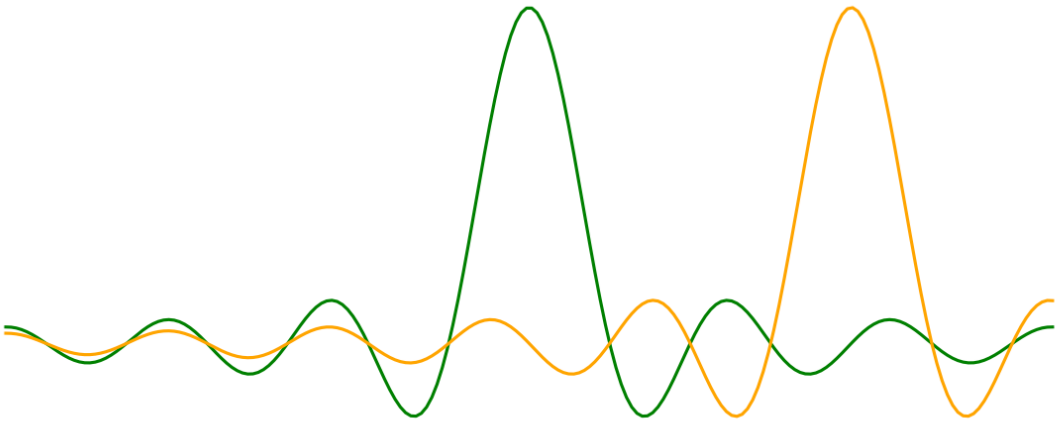
AA227C Project Report:

Comparison of Kalman filters and LSTM networks for GPS trajectory smoothing in the presence of simulated multipath noise

SUNet ID: mkosec, johnmp
Name: Matej Kosec
John Poothokaran



Contents

1 Introduction

The objective of this project is to compare the effectiveness of time-series filtering of GPS measurements using a linear Kalman Filter (traditional approach) vs. a recurrent neural network (LSTM filter) under the presence of simulated multipath noise.
The Kalman filter approach depends on heuristically defined process and dynamics noise matrices to produce its predictions. It also uses a user-specified dynamics model which may not be the best representation of the time correlation between the data.Additionally, while the Kalman filter is known to be optimal in the presence of Gaussian noise, it has major limitations on non-Gaussian noise.
The goal of the project is to overcome these issues by learning the noise and dynamics models implicitly though real and simulated data. The focus is on improving the performance on trajectories with non-Gaussian multipath noise. These are trajectories for which the Kalman filter is expected to perform poorly. However, rather than immediately testing our approach in a deep urban canyon, a stagged development approach is presented.
To first gauge the viability of the proposed approach we first trained the model to smooth a 1D trajectory as outlined in section 3. The extension to 2D trajectories is described in section 4, while testing the approach on real GPS data is shown in section 5. Firstly, however, the overall LSTM filter architecture is described in section 2.
The Python code for the filtering approaches in this project is available at:
`https://github.com/MatejKosec/GPS_LSTM_filtering`
Please reach out to the authors for details on using the code. The Matlab GNSS data processing tools used in this report were cloned from:
`https://github.com/google/gps-measurement-tools`

2 Architecture of the LSTM filter

This section introduces a neural net architecture which takes as input a timeseries of noisy measurements (velocity and position) and is trained to produce a timeseries (of the same length) representing filtered positions and velocities. To capture the time-dependence between different time-steps a long short time layer (LSTM) is used. To enable a fair comparison, the LSTM filter is only allowed to look at past measurements (not the whole timeseries), in the same way as a Kalman filter does.

2.1 Background on LSTM networks

The LSTM equations are presented in figure 1a. The LSTM is able to model temporal correlation between inputs by keeping an internal "cell" state c_t. The inputs are presented to the LSTM as a vector x_t and the outputs of the LSTM are given as h_t.
The Kalman filters internal state corresponds to the noise matrices and previous prediction. The LSTM is more general and decides how much to update the state with the current measurement using i_t, and forget previous information using f_t. This a much more general formulation than that of the Kalman filter. For instance, the LSTM can decide whether to add a measurement to the state or not using the update gate i_t, meaning it can directly discard those measurements it believes are coming from the multipath distribution rather than the true one. The state update is performed in line five and shows the previous internal state being combined with the current to produce the internal state to be passed on to the next step.

A further advantage of the LSTM filter (and neural networks in general) is that we can

(a) The LSTM equations from CS224n slides

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \tanh(c_t)$$

(b) EKF Equations from AA272C

- Predict
  - $\hat{\mu}_{t|t-1} = f(\hat{\mu}_{t-1|t-1})$
  - $P_{t|t-1} = F_t P_{t-1|t-1} F_t^T + Q$
- Update
  - $\tilde{y}_t = z_t - h(\hat{\mu}_{t|t-1})$
  - $K_t = P_{t|t-1} H_t^T \left(R + H_t P_{t|t-1} H_t^T\right)^{-1}$
  - $\hat{\mu}_{t|t} = \hat{\mu}_{t|t-1} + K_t \tilde{y}_t$
  - $P_{t|t} = (I - K_t H_t) P_{t|t-1} (I - K_t H_t)^T + K_t R K_t^T$
  - $\tilde{y}_{t|t} = z_t - h(\hat{\mu}_{t|t})$

simply decide to use a larger number of neurons for the internal state of the LSTM. In doing so we can model even more complex and non-linear dynamics. With Kalman filters additional accuracy is typically obtained by better tuning of the noise levels or addition of additional measurements such as acceleration to the model in addition to position and velocity.

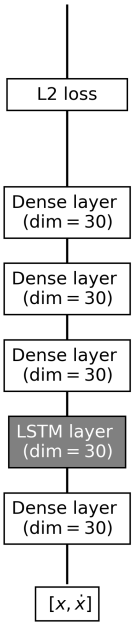
The EKF equations are included in figure 1b for comparison with the much more general equations of the LSTM. It should be noted that the added generality of the LSTM network comes at the price of robustness. The behavior of the linear Kalman filter on **unseen** trajectories is well understood while the behavior of the LSTM is only understood in so far as the average performance on the development dataset of unseen examples. Our project focuses on achieving good performance on a very small class of possible trajectories i.e. straight lines, squares, the Stanford oval. Generalizing to any general drivable trajectory would require significantly more data than we can collect.

In addition to using this 'vanilla' form of LSTM we also tested using two stacked LSTM layers, as well as a self attention mechanism. However, this added generality was ultimately not needed to achieve good results. In order for the LSTM to be a viable stand-in for the Kalman filter it should be deployable in real-time and only work on past measurements (future blind). The current proposed architecture meets these requirements easily on a laptop (although testing it on a mobile device would be the real test of runtime feasibility).

2.2 Hyperparameters and overall architecture

Finding the correct architecture in terms of both depth (number of layers) and width (number of neurons per layer) for a given problem requires a hyperparameter search (i.e. testing different numbers for layer size depth). Because of the stagged approach of going from 1D to 2D to 3D examples, we were able to find a good architecture quite quickly (in terms of computation expense) by testing the network on 1D examples first. This architecture is briefly summarized in figure 2 The implementation of this neural network is achieved using

Figure 2: LSTM filter architecture (deep neural network)



Tensorflow 1.8. For details see the code repository at `https://github.com/MatejKosec/GPS_LSTM_filtering`.

The most crucial hyper-parameter is the number number of neurons per each dense layer. This was kept 30 for the 1D, 2D and 3D test cases. That is, the LSTM internal state track 30 different parameters to temporally correlate different measurements. Additionally, all other linear layers were also kept at the same count of neurons to simply the hyper-parameter search. Again, this was not changed as adequate performance was obtained with the proposed architecture. Deploying on mobile and integrated devices would be more compute constrained and require a less expensive (more optimally sized network).

Selecting the number of layers was another significant hyperparameter. It was found that preprocessing the input data with another dense layer before passing it to the LSTM reduce the power of the model much more than adding additional layers after the LSTM. **The network has a total of** 9454 **trainable parameters**. This makes it a rather small network that does not require too much data to train. Around 400 traces of 100 time-steps

is the most that was found to be needed to achieve good performance.

2.3 Cost function for training the network

In order to enable training of the network the ground truth position and velocity are needed. It was decided to collect data in an unobstructed space (the Stanford Oval) and then add noise retroactively. This way the Android reconstructed position would be multipath free and can be use as true labels for training.
For the 1D and 2D test cases, the data is simply generate in the computer and thus the truth is known precisely. In term of training signal strength, the Android position ended up being nearly as effective as using true label in the simulated test cases.
The goal of the training process was to minimize the mean L2 loss between the predicted timeseries of positions $\hat{y}$ and the underlying ground true trajectory y.

$$\mathcal{L} = ||\hat{y} - y|| \tag{1}$$

This loss is averaged over all timesteps and trajectories to produce a mean value for the L2 loss. Thus there is no performance guarantee in terms of the maximum error the LSTM can produce, only what the mean error will be on similar trajectories. It may be noted that the L2 loss does not tell the network any relationship between the velocity and position. It has been tested to change this to enforce that the velocity integrate into position, however this did not yield improved results over the pure L2 loss. This additional loss can be though of as having a regularization effect.

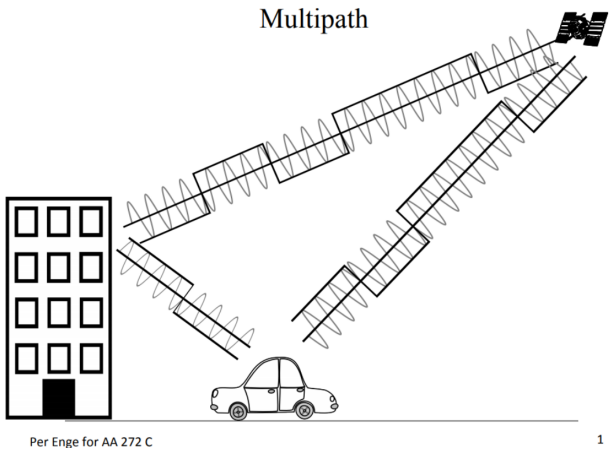
2.4 Regularizing the network (dropout and L2)

Since we are working with a fairly small dataset it is important to regularize the network and penalize it for over-fitting the training set. Specifically, the real goal of the training process is to perform well on unseen examples. This requires penalizing the network for trying to 'memorize' the seen trajectories (which would yield poor performance on non-memorized ones).
We used to common methods of regularization in this report. We penalized weights growing out of bounds by using and L2 loss on the weigh matrices. Secondly, we used dropout regularization with a keep probability of 90%. This means that with a probability of 10$ neurons are set to 0 activation and are therefore not used predicting the output, This has the effect as spreading the weights over a larger number of neurons.
By tuning these two regularization mechanisms we were able to achieve performance on unseen examples that is very close to the performance on seen examples.

3 One dimensional test case

Multipath effects enter the range rate data as an additional time bias term. The original of this bias term is roughly visualized in figure 3.

Figure 3: Slide on multipath from AA272C class notes 2018

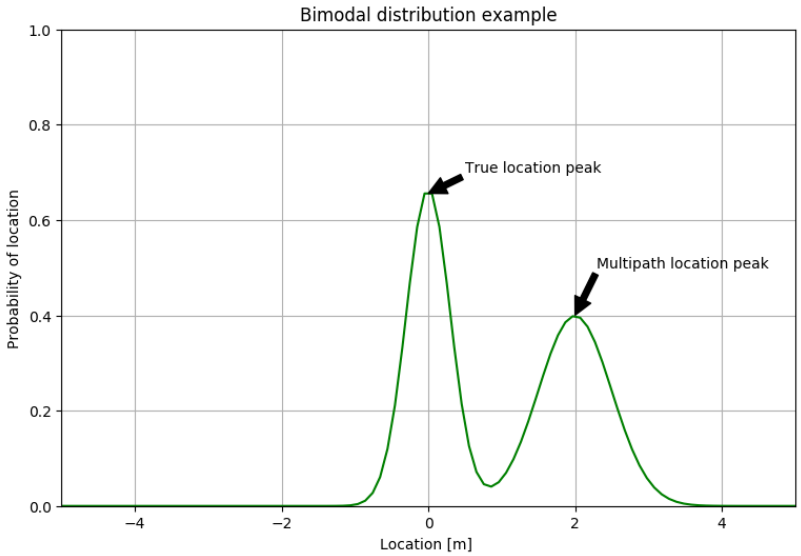


Here the signal takes both the direct path to the car and a longer path by reflecting off the building first and then reaching the car. We cannot in advance know which one of the two signals we are autocorrelating with and may incorrectly correlate with the signal on the longer path. At the stage of solving the least squares solution of line-of-sight equations this implies that we would compute our position to be farther away from this particular satellite than it really is.
The work in this project enters the picture after we have already incorrectly correlated to the reflected signal and erroneously included it our least squares solution. Thus, the proposed LSTM filtering techniques work directly on these positions and velocities and aim to reconstruct a multipath-free trajectory.

3.1 Building the 1D dataset

The assumed position-level multipath model corresponds to sampling the position disturbance from a bimodal distribution such as shown in 4. The true position is distributed normally with a mean centered at $[0, 0]$ while the multipath position distribution is at some bias value away from 0. In this 1D example the multipath distribution can be either to the left or the right of the truth. Notably, the true position and the multipath signal are modeled as having similar variances (difference in example image is exaggerated). This is done to discourage the LSTM from doing something as simple as picking to follow the distribution with lower variance. To keep the model simple, and also provide the Kalman filter with slightly better chances of success, a multipath model is **not** applied to the velocity measurements

Figure 4: A bimodal distribution for modeling multipath at position level



(in reality multipath would affect rangerate measurements as well).
With the bimodal distribution in figure 4 it can be expected that the Kalman filter will effectively attempt to fit a single normal distribution to both of them. As such the Kalman filter is expected to have a non-zero bias in the reproduced trajectory (rather than oscillating around the truth).
To train the 1D model effectively, a batch of 400 traces of 100 timesteps each was used. The traces model a person moving at a constant speed for 10 seconds (the position plots thus look like line plots). The speed itself is from a normal distribution. The mean of this distribution is sampled uniformly from the interval $[1.0, 5.0]m/s$, while the variance is in the interval $[0.9, 1.4]m/s$. This random sampling ensures ample variability between the examples in the dataset. Furthermore, the true position is computed by integrating the velocity. Noise is then added by sampling from a bimodal distribution as described in figure 4. The variance of both the multipath and the true peaks is sampled from the interval $[0.9, 2]m$ and the multipath peak mean is sampled from the two intervals $[4, 6]$(right) or $[-6, -4]m$ (left multipath). Switching which distribution is left and which is right is crucial, as otherwise the LSTM would simply learn to follow the left or the right distribution.
The model was further evaluated on a dev set of 10 traces, also of 100 timesteps. The dev set contains examples that the LSTM filter will not be trained on i.e. unseen examples. The goal of the training process is to train the filter using the training batch of 400 traces, such that it performs as best as possible on the unseen batch of dev set examples.
In the absence of a dev set, the LSTM network would simply learn the 400 train traces by rote in a process known as 'over-fitting'. In real life applications, we would wish to apply our filtering model to unseen trajectories which share some characteristics with the training trajectories. For instance, within a city people may always move on a grid. In the constructed 1D training set, they alway move on a line. It is by capturing such trajectory characteristics

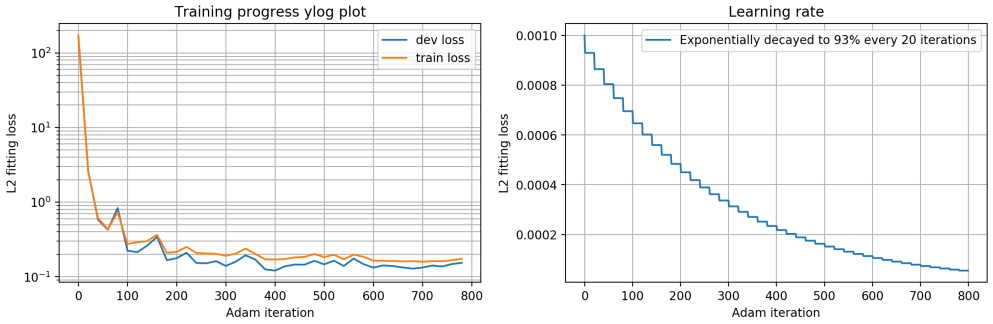
that we can hope to do better than the Kalman filtering approach.

3.2 Training on the 1D test case

The LSTM model as described in section 2 is trained over 800 iterations of the Adam optimizer using an initial learning rate of $1e^{-3}$ and an exponential learning-rate decay schedule of 93% every 20 iterations. Figure 5 left, shows the reduction in the cost function as the network is trained. Note the use of log-scaling on the y-axis as the cost function decreases several orders of magnitude.

In addition to the reduction in the cost function it is also important to consider the dif-

Figure 5: Training progress for the 1D test case



ference in train and dev set performance. From figure 5 left we see that the network is generalizing well to unseen examples as the dev loss (unseen) is nearly as good as on the training examples. Achieving this result required manually tunning the dropout and L2 regularization values to prevent the dataset from over-fitting.

Lastly, figure 5 right, shows the decay schedule of the learning rate taking place. It is necessary to reduce the learning rate over time to enable to network to converge more stably in the final stages of learning.

The Kalman filter was also 'trained' to achieve good performance on the dev set. However, this involved manual tunning of the noise matrices rather than optimization. Furthermore, since the noise profile is different across the training data (noise variance and mean sampled randomly), the Kalman filter needs to be set to a value which approximately suits all scenarios approximately (but none perfectly). Curiously, when training the Kalman filter it was found that better performance could be achieved by reducing the R values corresponding to velocity noise i.e. trust velocity measurements more than what the actual known variance of the data would suggest. In contrast the position measurement noise levles need to be boosted to achieve good performance.

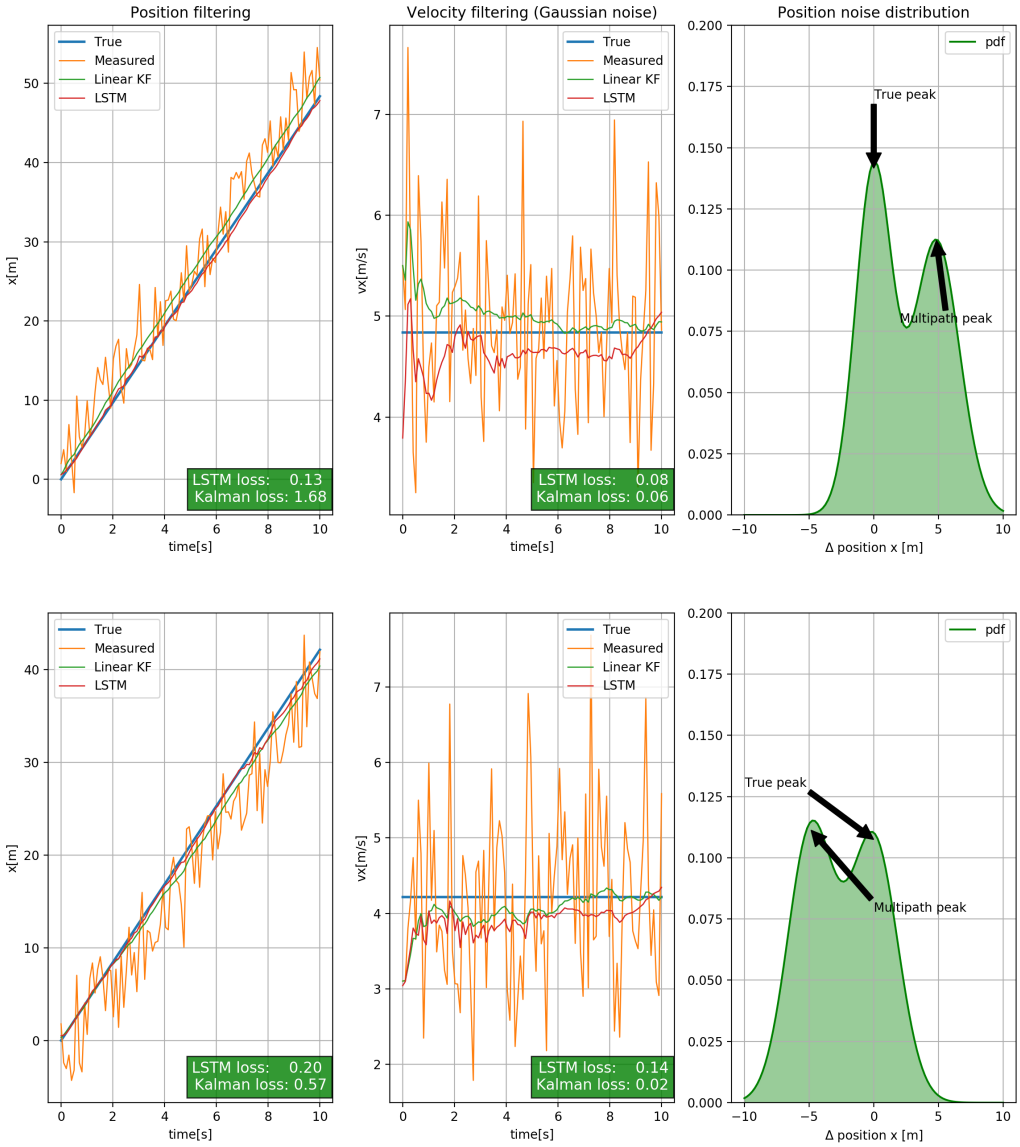
3.3 1D results and conclusions

Figure 6 shows the performance of both the LSTM filter and the 1D linear Kalman filter on unseen 1D trajectories. The underlying true velocity setting and positions are shown in blue

and correspond to straight line trajectories.
The left column shows the filtered positions as produced by the Kalman filter (green) and LSTM (red). The LSTM trajectories lie at a nearly constant offset from the true trajectory line. That is, the Kalman filter is not able to detect the bias in the data and therefore shifts its mean to the mean of the bimodal multipath noise distribution. In-contrast the LSTM filter is able to follow the true trajectory very well, despite the fact that most of the samples fall strictly on one side of the true trajectory line. The L2 loss of the position filtering is shown in figure 6 as well. In both cases the LSTM filter perform significantly better. In the first trajectory example the position loss is more than $\sqrt{\frac{1.68}{0.13}} = 3.6$ times lower. In the second trajectory, the difference is less significant, but still $\sqrt{\frac{1.68}{0.13}} = 1.7$. That is, the error is decreased by 70% with the LSTM filter.

The central column of figure 6 shows the velocity filtering performance. The velocity is assumed to only be affected by Gaussian noise as outlined before. Thus, a properly tuned Kalman filter is expected to outperform the LSTM filter. This is indeed the case for both trajectories despite the fact that the Kalman filter was not specifically tuned for the noise setting in each trajectory. The combined loss for both trajectories is still much lower for the LSTM filter however. That is, the loss is better balanced between position and velocity in the case of the LSTM.
Additionally, figure 6 right shows what noise distribution was applied to the data. The results show that the LSTM filter is consistently able to pick the correct distribution to follow regardless of whether the multipath peak is on the left or on the right of the true position peak. Furthermore, the lower trajectory features a noise distribution where the multipath and the true noise distributions are nearly identical in magnitude (multipath is actually slightly higher), yet somehow the LSTM network is able to correctly pick between them.
While more work would be needed to determine the robustness of the LSTM filter in this setting, it is likely that the LSTM is able to leverage the learned characteristics of typical trajectories encountered during training (constant velocity trajectory) and apply those statistics at runtime on unseen examples. The same network could be extended to work on multiple different classes of trajectories if enough examples could be provided. In this report the limited constant velocity trajectory class was chosen to prove that the concept of filtering using an LSTM network was valid, and allowed key hyper-parameters to be tuned with relatively low computational cost. Training time was on the order of 1 to 5 minutes.

Figure 6: Example results on **unseen** 1D trajectories



4 Two dimensional test case

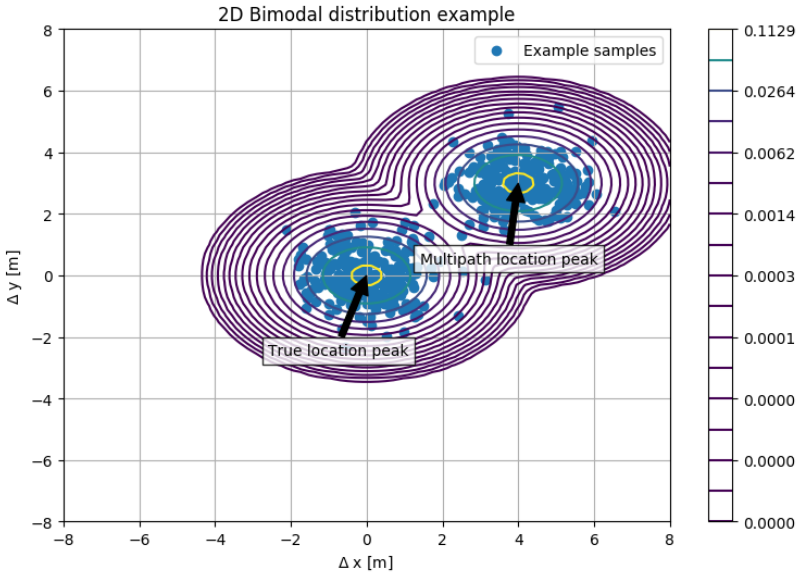
Having successfully demonstrated the ability of the LSTM filter to handle simulated multipath on 1D trajectories, the next step was to prove its viability on 2D trajectories.

This is considered the final stepping stone before working on real GPS trajectories (which is the real goal of this project).

4.1 Building the 2D dataset

The goal of the 2D test case is to confirm the LSTM can leverage learned statistics about 2D trajectories. For this reason the 2D dataset contains a single class of rectangular trajectories intended to simulate someone navigating a city block with an obstructed view of the sky.
The 2D training data is created in a similar fashion to the 1D test case described in section 3. Specifically, the bimodal mixture of two Gaussian distributions is extended to two dimensions as visualized in figure 7. The distribution centered at $[0, 0]$ corresponds to the true position, while the offset distribution corresponds to the multipath position. The variance of the true and multipath distributions is taken to be the same (i.e. sampled randomly from the same range). This is done to discourage the LSTM filter from being able to pick which distribution to follow simply based on its variance.

Figure 7: A bimodal distribution for modeling multipath at position level in 2D



Additionally, figure 7 shows an example of how 500 samples drawn from this distribution would cluster. Once again the Kalman filter is expected to batch the samples together into a single cluster and thus introduce a bias into the trajectory.
To generate the 200 trajectories needed to train the LSTM filter, the constant 'x' and 'y' velocities are uniformly sampled in a range from 1.5 to 5.0m/s. That is, the velocity is assumed to be constant along each leg of the rectangle. The Gaussian noise to be applied to the velocities has mean 0 and variance sampled uniformly from the range 0.8 to 1.5m/s. The variance for the multipath and true position Gaussians is also sampled from the same

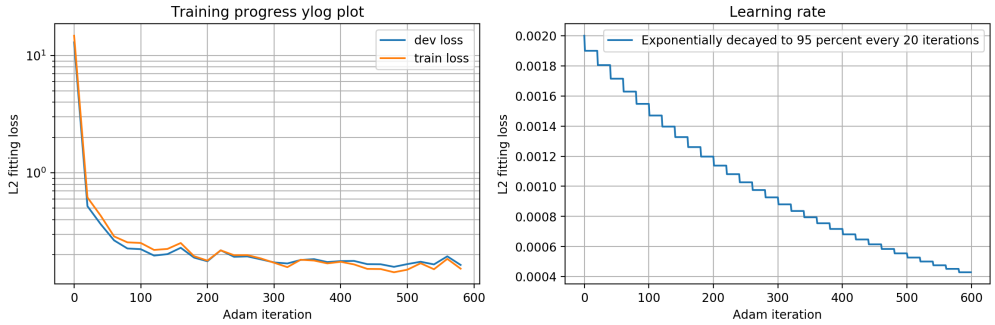
range. Specifically, only the diagonal terms of the covariance matrix are sampled from this range. The off-diagonal terms are set identically to 0 without significant loss of generality. Furthermore, the mean of the multipath noise is constrained to be sampled between 3 to 5 meters of the true location (this prevents the true and the multipath distributions from every lying on top of each other).

4.2 Training on the 2D test case

The fundamental architecture of the neural network is kept the same between the 1D and 2D datasets with only smaller tweaks to learning rates and regularization.

The training progress proceeds similarly to the 1D test case with the training and dev set

Figure 8: Training progress for the 2D test case



(unseen examples) achieving nearly identical performance. Thus we can reasonably expect that the LSTM filter will work on unseen examples from the same distribution. The learning rate decay is also kept the same as in the 1D case.

The fact that so few changes needed to be made between the 1D and the 2D test case encouragingly suggest that the network will also work for smoothing the 3D trajectories of the real GPS data.

The linear Kalman filter was again also manually tuned to achieve good overall performance on the training set. However, since the noise does not always come from the same distribution (but from a range of distributions), the Kalman filter is not particularly tuned for any individual trajectory. It was found that the best performance was achieved by lowering the measurement noise for the velocity components and trusting the model for the position.

For more details on how the training process works, please refer to the 1D section 3.

4.3 2D results and conclusions

Figure 9 summarizes the performance of the LSTM and linear Kalman filters on **unseen** trajectories. The left column shows how well the predicted positions match the underlying true trajectory. The underlying noisy measurements are shown in orange. These high noise levels make the filtering problem considerably harder but also comparable to the GPS noise level.

The first and most important observation to make is that the LSTM filter is able to successfully learn that the class of trajectories we are feeding it is always a rectangular trajectory. This leads to the LSTM trajectories to be closed circuits, while the Kalman filter predicts diverging spirals. This spiral pattern is a direct result of the non-zero bias multipath noise. The Kalman filter integrates this bias over time, resulting in an ever increasing error that is simply not corrected by additional position measurements. This observation is very interesting as the GPS test data is also be collected on a closed circuit around the Stanford oval and the same behavior is presumably to be anticipated.

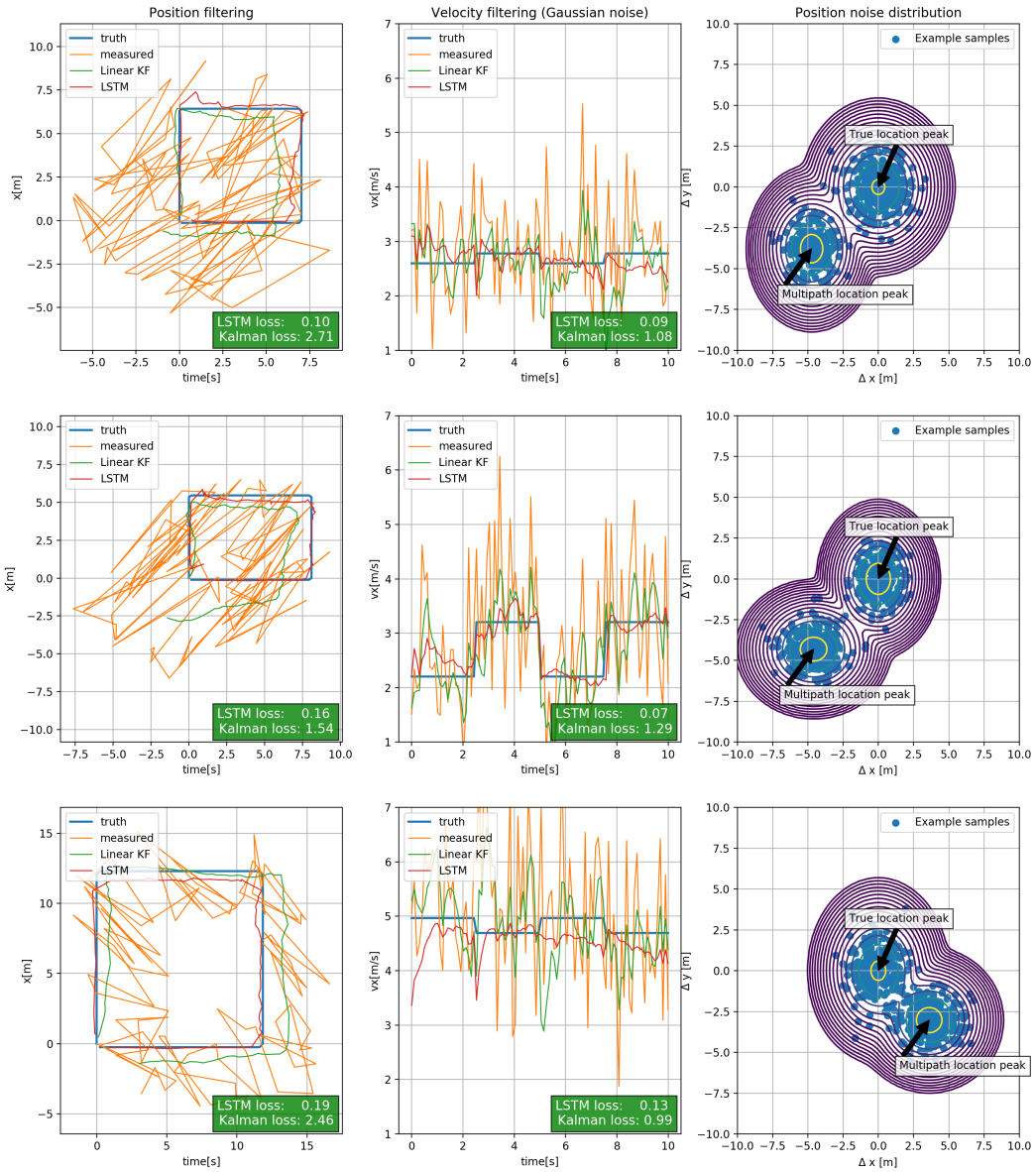
In terms of the L2 fitting loss the LSTM filter greatly outperforms the linear Kalman filter. The difference in fitting performance is even greater in 2D than it was in the 1D test case. Additionally, while the Kalman filter was able to outperform the LSTM on filtering the 1D velocity with only Gaussian noise in 2D, the LSTM performed much better in 2D. Again it should be noted that the Kalman filter is tuned for the noise setting of all trajectories and thus does not perform too well on any one of the trajectories in particular. Overall the L2 loss for the LSTM filter is about ten times smaller on velocity fitting.

In position filtering the same 10X trend is observed. It should again be noted that the examples shown are unseen rectangle examples i.e. their dimensions are sampled randomly. Thus the LSTM does not know the exact trajectory in advance but must figure it out along the way. From figure 9 it can be seen that one the last side of the rectangle (bottom) the LSTM knows nearly exactly to take a straight line to position $[0, 0]$. Thus the loss is especially low on this last edge. On the other hand for the Kalman filter the loss is the largest on the last side.

The rightmost column of figure 9 shows the bimodal normal distributions from which the position noise is sampled. Comparing the second and third row of the figure it can be observed that pulling the two distributions further apart improves the performance of the LSTM and degrades the performance of the Kalman filter. This is to be expected as the LSTM is more readily able to distinguish between which samples come from the true distribution and which for the multipath one. In the case of the Kalman filter, increasing the separation between the two distributions only increases the position bias, resulting in degraded performance as the mean of the samples is pulled further from $[0, 0]$.

In conclusion the LSTM filter shows promising performance in 2D when fitting on a restricted class of trajectories (rectangular circuits). As such, it is hoped that similar performance can be expected on real GPS data when collected on a similarly restricted class (cycling around the oval). Extension of the methodology to multiple classes (and blends thereof) would be straightforward but time and data intensive.

Figure 9: Example results on **unseen** 2D trajectories



5 Real GPS data test case

5.1 Building the GPS dataset

Data collection including performing one stationary measurement at the center of the Stanford oval, and then completing 10 full trajectories around the oval on a bicycle. The oval has relatively open skies making it a convenient testing space for collecting data which is multipath-free. All multipath noise effect are added artificially after the fact.
The data logged on the Android phone was processed using a script modified from the open source MATLAB code. In order to obtain the true labels (GPS fixes) corresponding to the measurements, the script reading the logged data was modified to include measurements at the same time step as GPS fixes, thereby skipping measurements that did not have an associated GPS fix. The raw data is then processed with the modified open source MATLAB code and a bimodal distribution of noise is added to segments of the raw measurements to simulate multipath. The pseudoranges are smoothed and then corrected for ionospheric/tropospheric effects using the errors obtaned from the GNSS Analysis tool. The corrected pseudoranges are then used to solve for the least squares solution at each timestep. The GPS fixes are read from the log file, and the positions and velocities from the least squares solution and the GPS fixes are written to file to be used as the input to the deep learning models and the true labels respectively.

The effect of multipath was simulated in the dataset by addition of different bimodal noise profiles to the received time in the raw GNSS log data. The random noise was added to a continuous subset of the received times. Different bimodal noise profiles were used in different training sets. The details of the noise profiles are provided in Table 1

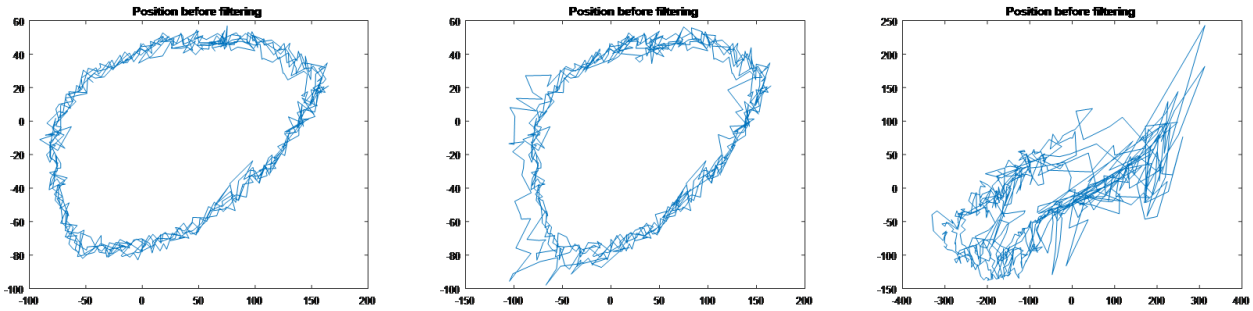
Distribution	Distribution Range	Timesteps	Multipath SvIndices
Bimodal	(50,20),(120,40)	150-300	(2,4)
Bimodal	(90,10),(120,40)	100-250	(2,4)
Bimodal	(60,30),(150,30)	150-300	(2,5)
Bimodal	(50,20),(100,30)	100-250	(1,5)
Bimodal	(70,10),(100,30)	100-250	(2,5)
Bimodal	(60,30),(160,50)	50-100	(1,3)
Bimodal	(70,10),(130,10)	150-300	(2,4)
Bimodal	(50,20),(150,30)	100-250	(1,3)
Bimodal	(60,30),(100,30)	0-100	(1,2)
Bimodal	(60,30),(150,30)	50-200	(2,5)

Table 1: Noise Profiles simulating Multipath

5.2 Training on the GPS dataset

The fundamental architecture of the neural network is kept the same as the 1D/2D cases. It is used for the 3D GPS data that is preprocessed using least squares and used as training

Figure 10: Preprocessed GPS Data



(a) Raw Data - No added noise (b) Noisy Data with Multipath (c) Noisy Data - No added noise

data with the true labels obtained from the GPS fixes provided by the phones internal location determination system during data collection. The training progress proceeds similarly to the 1D and 2D test case with the training and dev set (unseen examples) achieving nearly identical performance. Thus we can reasonably expect that the LSTM filter will work on unseen examples from the same distribution. The learning rate decay is also kept the same as in the previous cases.
As expected, the LSTM was able to learn the characteristics of the position and velocity from the same distribution and performed significantly better on segments with multipath effects
The linear Kalman filter was manually tuned to achieve good overall performance on the training set. The Kalman filter had some difficulty with the noisy data collected in tne first set of data, but performed better on the less noisy dataset in generally following the measurements. The filter was tuned to rely heavily on the dynamics of the process and consider that the measurements - particularly position - are quite noisy.

5.3 GPS dataset results and conclusions

Figure 11 - 14 summarizes the performance of the LSTM and linear Kalman filters on **unseen** trajectories. Figure 11 shows the results of the trained LSTM and the Kalman filter on the preprocessed (rather smooth) data with no added noise. Figure 12 and 13 show the results on very noisy data as well as on the less noisy data with easily identifiable multipath effects. The left column shows the decrease in the train and dev loss over the training iterations. The middle columns in these figures show the horizontal position and the right column shows the vertical position as we move along the Stanford Oval in the collected data.

The LSTM is able to successfully learn that the data that goes through it is in the general shape of the oval, whereas the Kalman filter (which is fed the true position as the starting point) is seen to drift away from the true labels with the measurements and follows these measurements when they are noisy and also when the effect of multipath is seen in Figure

Figure 11: Results from GPS Data Training

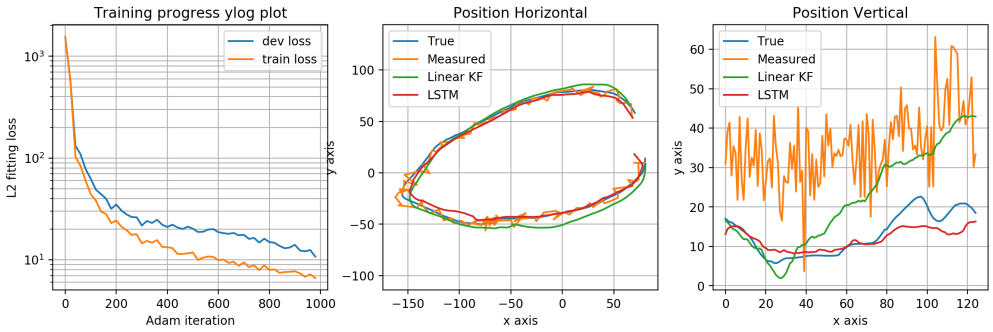


Figure 12: Results from GPS Data Training

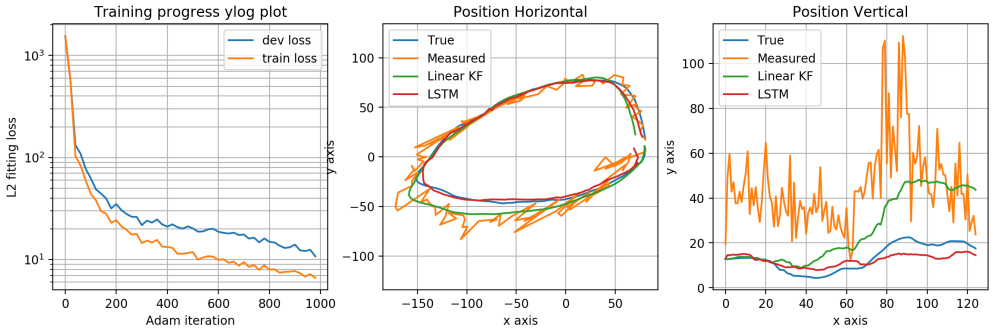
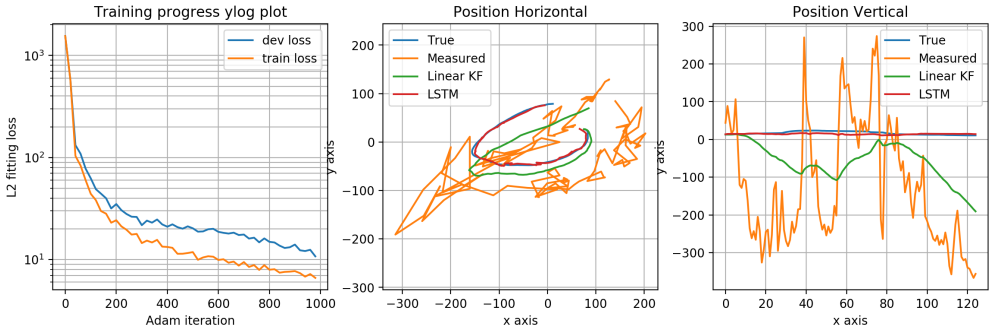
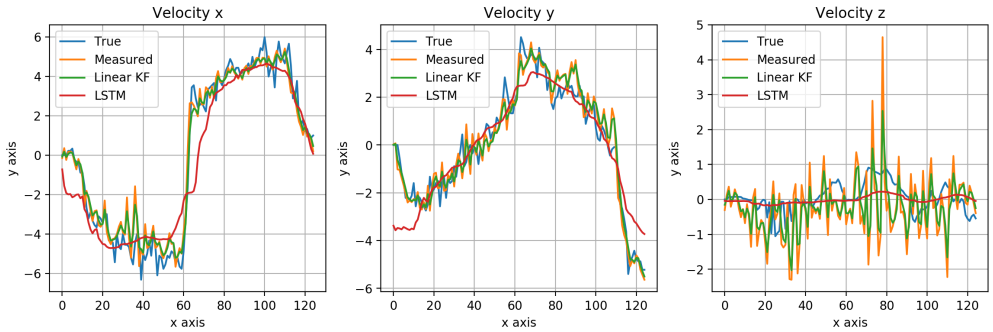


Figure 13: Results from GPS Data Training



12. The Kalman filter does show the same issues as is seen in the 2D test case, where the path drifts with noisy measurements and is not aided by additional position information. Figure 14 shows the results of the filters on the velocity data obtained. It is seen that the velocity is fit well by both the LSTM and the Kalman filter and this could be due to the relative smoothness of the velocity data as compared to the position data. In terms of the L2 fitting loss (for position and velocity), the LSTM outperforms the Kalman filter and has a loss that is roughly 1000 times lower. Again it should be noted that the Kalman filter is tuned for the noise setting of all trajectories and thus does not perform too well on any one

Figure 14: Velocity Results from GPS Data Training



of the trajectories in particular.

6 Conclusions and future work

The results of the training on the GPS data, as well as the 1D and 2D test cases show that LSTMs could potentially be incorporated into positioning systems and trained to recognize and ignore multipath effects in a wider context. The time series nature of the data and knowledge of the movement characteristics of the device (such as walking, biking, driving) could be incorporated in the training data to help the LSTM more accurately determine position based on previous measurements. Furthermore, map images could be used in the case of driving navigation to train the LSTM to learn to restrict itself to drivable areas on the map and provide improved positioning and navigation experience to end users.
Further demonstrating the viability of the LSTM filter boils down to making it capable of generalizing to a more general set of trajectories and ensuring it can be deployed on mobile and integrated devices. While it may be impossible to meet the computational parsimony of the linear Kalman filter, the current architecture already achieves a performance of several thousand samples per second.
Further work should thus focus on collecting more data and extending the approach to more general trajectories. Working with real multipath data has the further issue that true labels are unavailable (the Android location will be off). Overcoming this limitation is crucial to future viability.