people tracking and prediction

Kalman Filter & LSTM Neural Network Prediction

Nicholas Shindler

March 23, 2020

Contents

Abstract

This project explores the analysis of human motion. focusing primarily on short term motion prediction using probabilistic and machine learning approaches and comparing the results. This project covers tracking multiple people. Data is designed to be collected by RGB-d camera. A basic Kalman Filter is used for prediction. A simple RNN utilizing LSTM is also examined for prediction.

1 Introduction

In this report I will cover, the intended goals of the project, the general structure of the project, how I tested the effectiveness of my algorithms, and discuss how I met my goals and the results of my tests. In this project I combing machine learning and probabilistic methods to predict human motion. I use the ATC pedestrian tracking data-set for my people data, as it allowed me both the ability to test my system with a large number of people being tracked at the same time, and a large amount of available data to train my motion prediction neural network. My goal was to evaluate real time prediction and compare a probabilistic to machine learning approach, as well as combine both approaches for hopefully the best results. I constructed this project to run in real time, and operate both with minimal data as well as a robust and fault tolerant structure.

2 Project Goals

This project is designed to evaluate the effectiveness of prediction of human motion. The core component of this project is a prediction module that uses a Kalman Filter with a basic motion model to predict tracks of people. A secondary aspect of the project was a detection node to convert data from rgb-d video (read in as point clouds) to people tracks. This was set to be done with the Point Cloud Library (pcl) and a SVM algorithm for ground plane based people detection. Additionally I wanted to experiment with the comparison of results between a probabilistic approach and a machine learning approach to people detection, using an LSTM neural network to predict motion with similar data input to the kalman filter. and then combine both methods to see if they could achieve even better results.

3 Application: design and design choices

This project was done using the robotic operating system (ROS) as the basic framework. This allowed the different tasks to be broken into sections based on task (detection, prediction, etc). Google Colab was also leveraged for training the neural network. Additionally the ATC pedestrian tracking data-set was leveraged heavily (`https://irc.atr.jp/crest2010_HRI/ATC_dataset/`).

3.1 Camera and data input

Because this project was done with Ubuntu 18/ROS Melodic, the ROS bridge for the Kinect 2 would not run, and a Kinect 1 camera was used. The data from the Kinect camera was brought in using the *openni* package. This data was represented in the ROS *PointCloud2* format.

3.2 Detection

The **Point Cloud Library (pcl)** was going to be leveraged in conjunction with a pre-configured SVM for people detection. However, due to issues with the code supplied by the tutorial, and project priorities, a pseudo detector was created the used the **ATC pedestrian tracking data-set** for a data stream and published the data to a ros topic in the same format as if it was begin taken from real time video. This method ended up giving a number of advantages. Most significantly it provided data with multiple people tracks and a variety of motion patterns. however as a result, the detector for live data was never completed.

The data both from the camera and from the ATC tracks, read in data at a rate of 30 readings/frames a second. The detector was setup to use a buffer system allowing a variable rate of data publishing based on the needs of other modules in the system. I set it up to publish data at $\tilde{1}0$/second.

3.3 Prediction

The prediction module had 2 main parts. At an interval of 10/second data was read in and passed into a full Kalman Filter utilizing both the prediction and update steps. A second part ran at a

rate of 1/second that would create a prediction vector for each of the people tracks at that stage. These predictions used the prediction stage of the Kalman Filer, and recorded a prediction every second for 8 seconds into the future. This prediction was enhanced, by using the prediction from the previous time interval to update the prediction. This worked where the prediction at time t, n seconds into the future would take the prediction at time $t-1$, at $n+1$ seconds and run the *update* section of the Kalman filter with both values. this was done for the first 7 $(n-1)$ predictions, as the final (n^{th}) prediction had no previous time interval prediction, and therefore only used the *prediction* section of the Kalman filter. These updates also used the same Kalman Filter instance as the filtering of the data, which should cause the quality of the update to improve the longer the tracks were analyzed.

The Kalman Filter used a basic motion model based on constant velocity motion:

$$x_{n+1} = x_n + \dot{x_n} * \Delta t \tag{1}$$

$$y_{n+1} = y_n + \dot{y_n} * \Delta t \tag{2}$$

3.4 Evaluation

While the data detection and prediction modules operate in real time. The Evaluation module runs at the end of a data collection interval. The collection interval is configured to run for 30 seconds. The data is recorded in seconds an millimeters. The evaluation stage creates 2 data-sets. One data set is printed as a human readable log, and an example of this is included at the end of this report. this data set tracks the error for each person and prediction interval, with averages done over time. The second data set is setup to be create plots using **gnuplot**. There are gnuplot scripts for graphing in the data module of the project. This data uses the position and trajectory error at each time interval and prediction step, averaged over all people tracks in each time frame. I also found plotting tracks of the data and predictions provided a good visual indication of the results being achieved.

3.5 Neural Network

Using the ATC data a neural network was created and trained on google colab. this NN used a basic LSTM structure, predicting a person's state at the next timestep using the previous second's worth of recording data. while better results could be achieved by expanding the model, I felt it was important to keep the data being input to a similar time interval as what the Kalman Filter was using. Particularly as I hoped to feed the prediction for the NN into the KF, to create a *deep kalman filter*, which I felt would give the best results for motion prediction.

The Neural network was trained on a separate data set from the one being used by the detection module as pseudo detection tracks. I used a subset of 2500000 elements for training the model.

I exported the model into my project to be utilized by a python ROS node, that could return predictions based on inputs passed by a ROS service call from the prediction node.

While I am limiting myself to 1 second of data for what I am passing into the model, this has a vast number of permutations in terms of structure. The data input is structured to contain a history size (the basic block of previous data input), a batch size (breaking the data into repeated sections), and a step (this is simplify to just be 1). The input data needed is equal to the history size plus the batch size. For me this will be limited to 10 so it can be implemented in the prediction algorithm. my goal was to set up the integration into ROS so that multiple similar models could be tested.

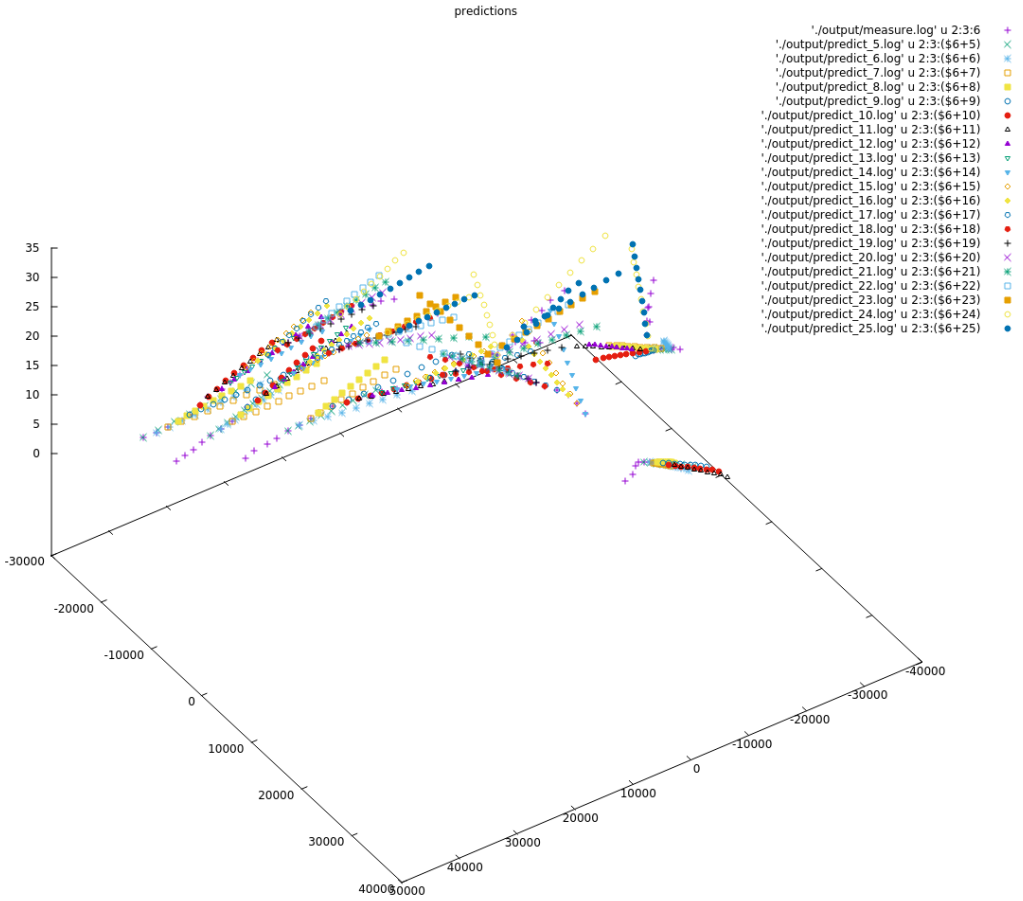
4 Experimentation Results

4.1 Probabilistic Prediction

the first note is that if you are looking at the code that creates the evaluation data, it is not constructed, efficiently, it is not ordered logically, it was written to minimize the amount of data being read into memory at any given time.

the second note is that I am presenting this data as data, without trying to say if this is good/bad error, or an acceptable deviation, or any other value based judgments. this is because I have collected and analyzed this data without a goal, in so much as I I am not doing anything with the predictions. Depending on the application the numbers mean wildly different things. As people prediction for a robot moving in a crowded shopping mall will need a much higher resolution of prediction (±0.1 meter), than a drone that needs to have an idea where all the people will be by the time it's missile hits the ground (±3 meters).

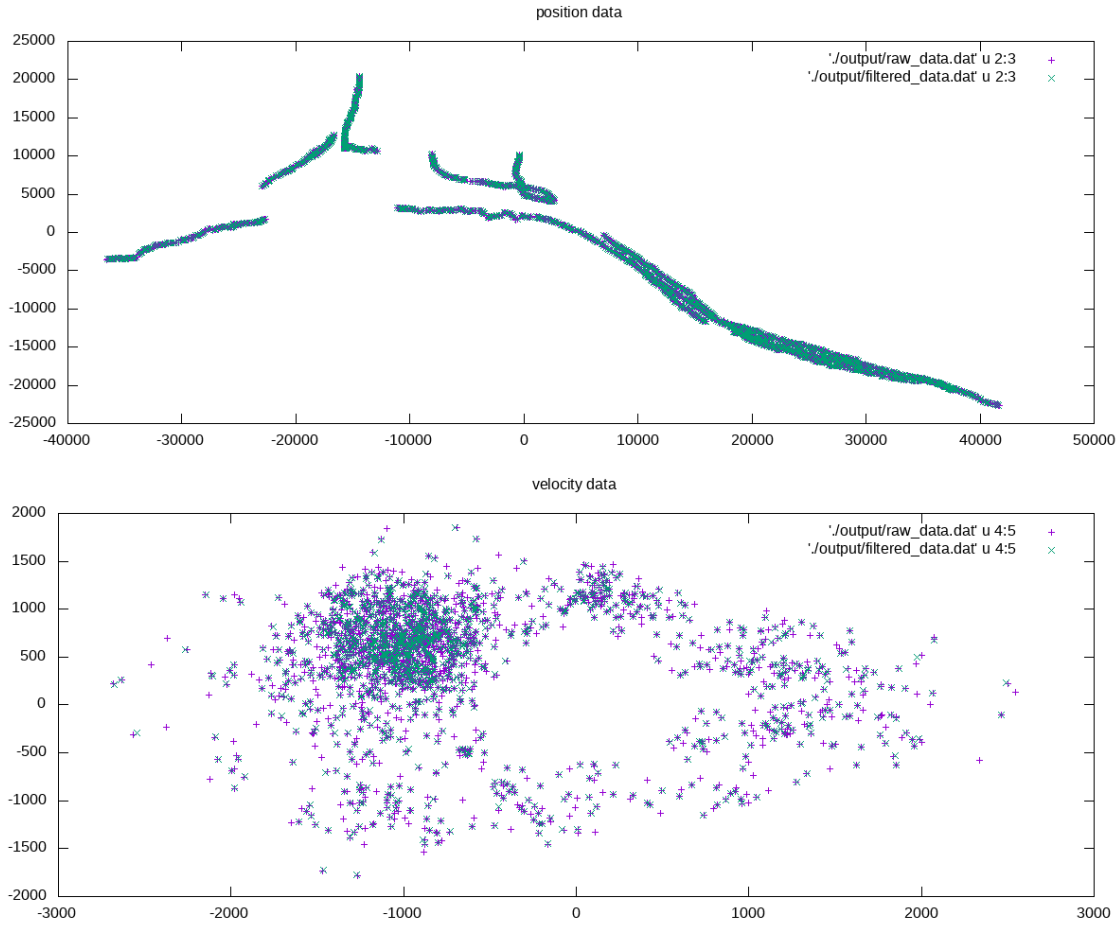
4.1.1 sorting through the data



My decisions about how to represent this data focus on the idea of representing the error, and how it changes, rather than trying to present **all** the data.

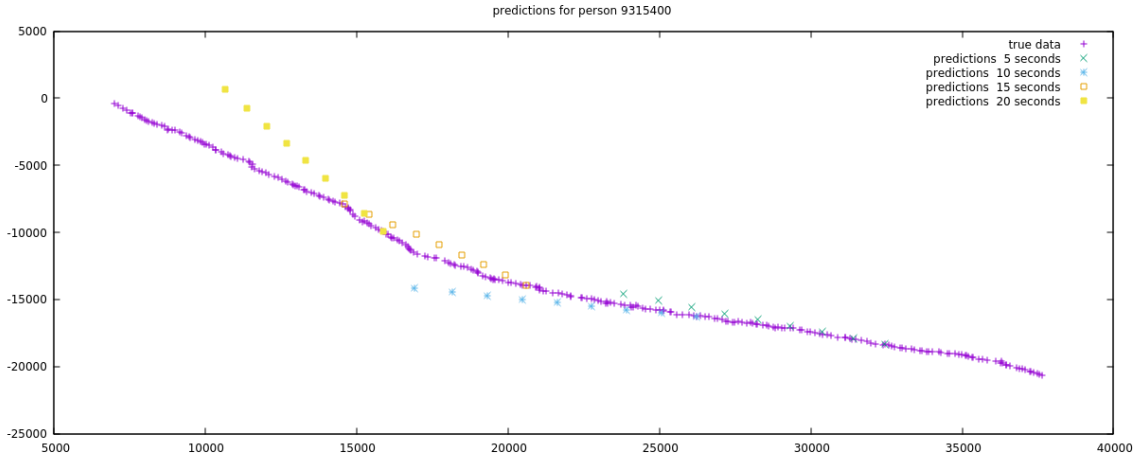
with that in mind I have provided an examination of the data tracks for a visual error approximation, as well as calculations of how the error changes numerically over time.

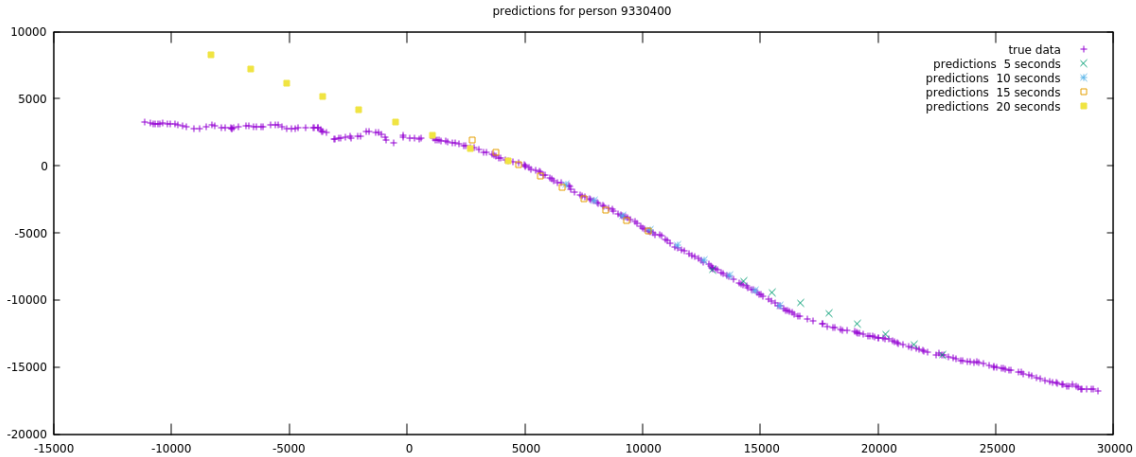
4.1.2 people tracks



this is a basic plot of what the data looks like. As you can see the distribution of the velocity is not incredibly meaningful by itself, however it does give a good idea of how the Kalman Filter (filtered data) changes the raw data, something not as visible in the position, displaying the failing of the motion model to account for changes in velocity, particularly at higher speeds. the velocity plot also does well to show the direction of motion for the people as a group.

The position tracks show the people motion on the x y plane. As you see most of it is relatively linear. However there are also many short segments representing potentially people that the detector lost track of for some time. As well as non-linear tracks.

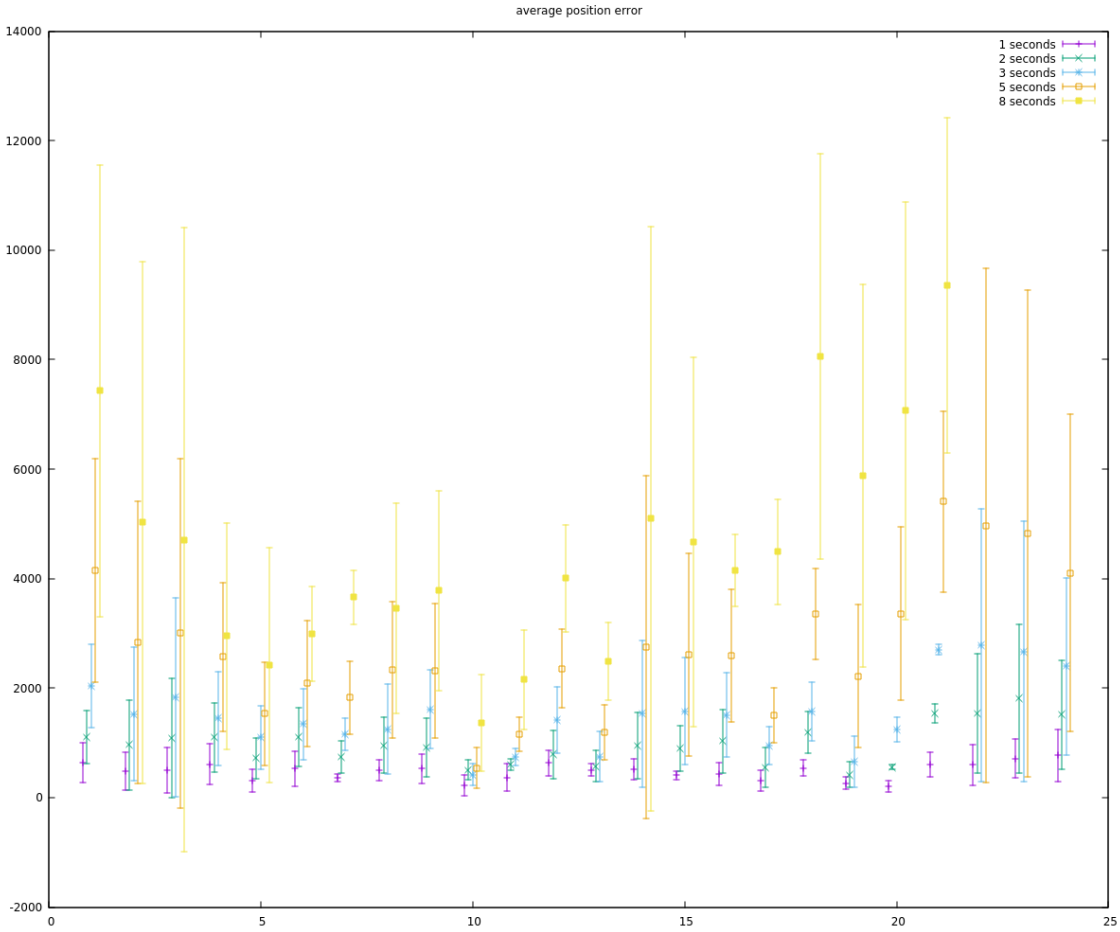




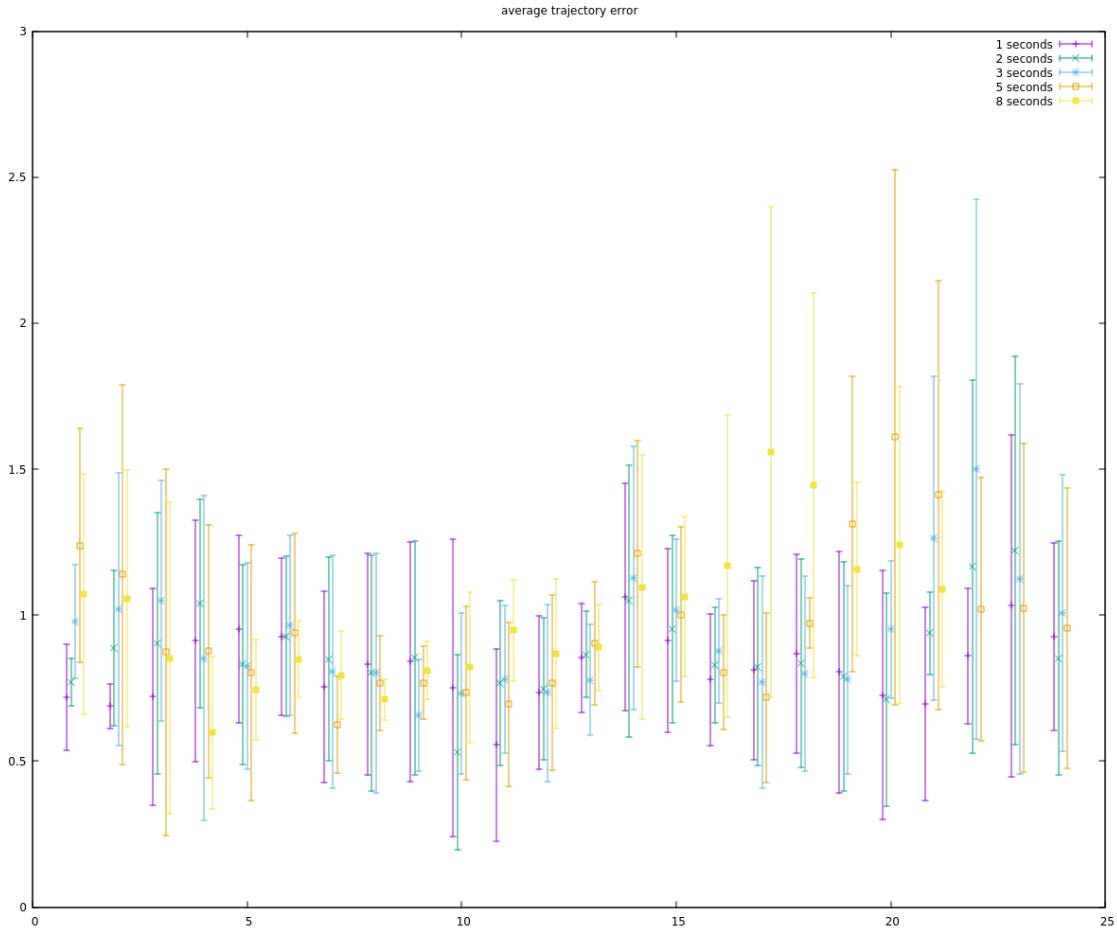
These two people tracks were chosen for the most part because they are 2 of the longest continuous tracks. In these tracks you can see both were the predictions succeed and when they fail. the predictions at 15 and 10 seconds of person 9330400's track are very good, over the entire 8 seconds, however, as soon as there is a change in trajectory like at 20 seconds, the prediction quickly gets very bad.

4.1.3 error over time

there are small x-axis offsets to the data this is artificial to allow all the data to be easily read on the same graph. in reality all data occurs on 1 second intervals.



this error is the calculated absolute positional error, averaged over each person on the time interval, and point is plotted for 5 of the 8 prediction intervals, the error bars are based on the standard deviation. The error is calculated as the absolute distance between the measured and predicted points on the x y plane. I was surprised that there did not appear to be any improvement in prediction over time, based on the Kalman Filter updating it's values. I think that this was because the changing amount of people and in the system, and the similarity of real tracks to the model, effected the data so much more than the accuracy of the KF. I also found it notable that at the longer prediction intervals the deviation grew substantially, demonstrating once again the limitations of the model being used.



This data was constructed similarly to the position data in the previous section, however, the error is calculated as the angle between the predicted and measured velocity vectors. From the way that the trajectory error is relatively constant over both trial time and prediction time, we can see a reinforcement that some tracks *follow* the model, traveling in a straight line, and have a rather low trajectory error, while other tracks turn, or curve, creating consistently wrong trajectories that do not get either better or worse over time.

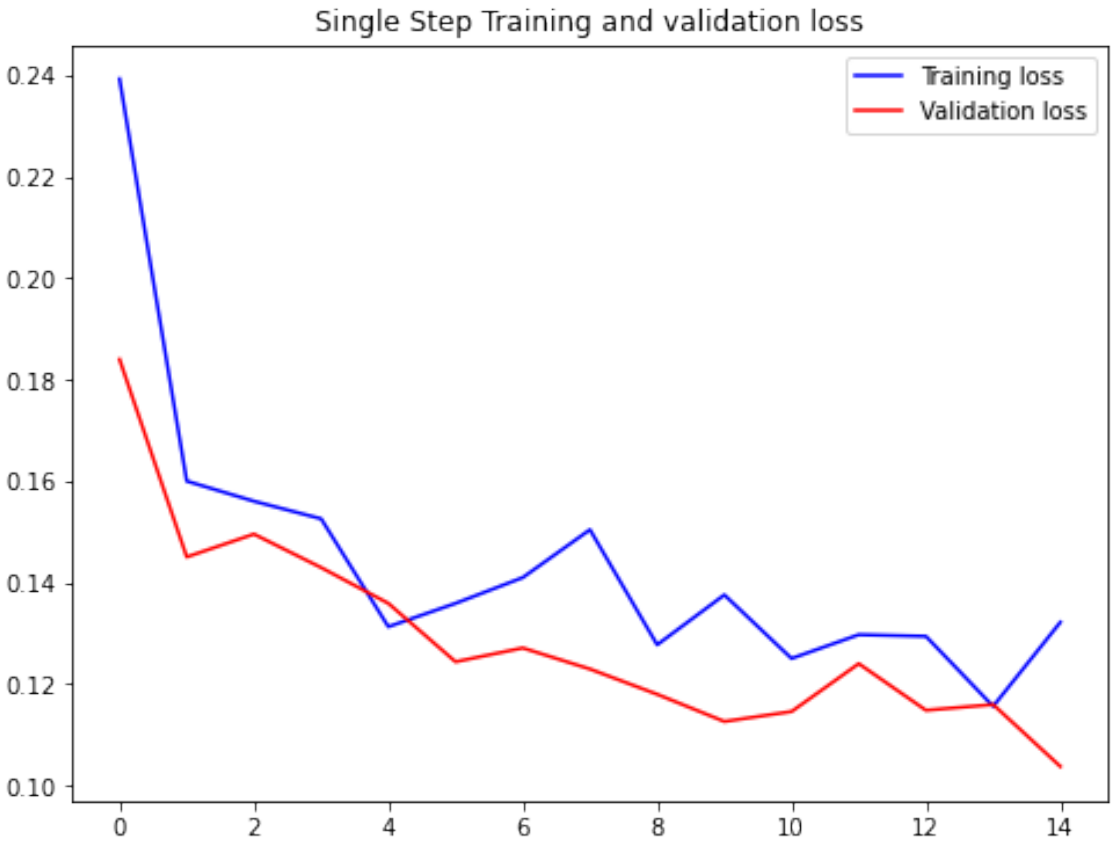
4.2 Machine Learning Prediction

Due to the ML model not being fully integrated into the ROS infrastructure, live testing has not yet happened. However basic analysis of the models was done during training.

4.2.1 the model

training the model required iterative testing to determine the optimum format for the data, particularly the ability to run it in a real time process. This means I was not able to set a long history length for example as I these predictions are iterative and short term, requiring small amounts of data. Particularly as most single motion tracks are less than 10 seconds. Therefore I chose to use a 1 second interval for history, requiring the 10 previous readings. Additionally, I worked to get a good relationship between the size of the data set I was using for training, and the fitting interval length and epoch number. training over 15 epochs of length 1250, with 150 validation steps, and 2500000 data points, I think I have a well trained model.

Figure 1: Training and validation loss



4.2.2 the predictions

While I didn't want to duplicate work, by creating a complex evaluation program in Colab, as part of the training evaluation I picked out graphs for people tracks.

Figure 2: x and y position prediction

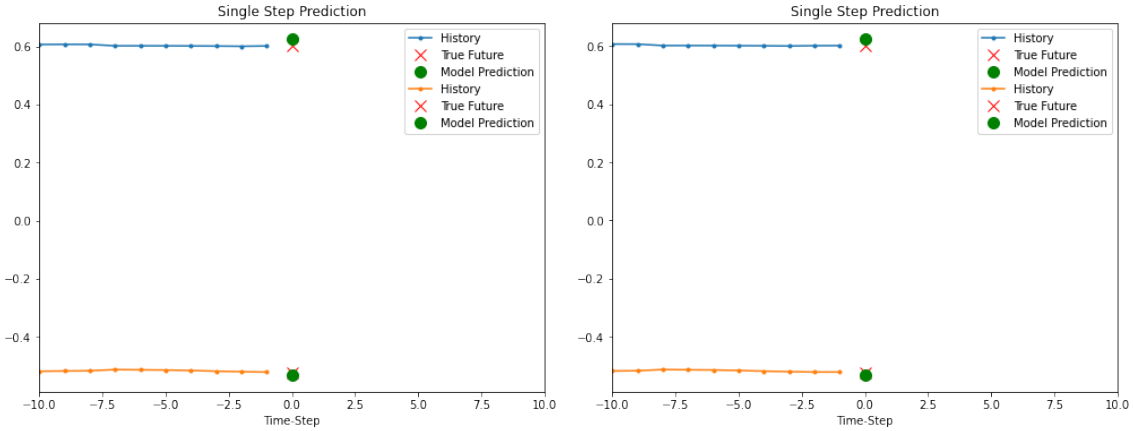
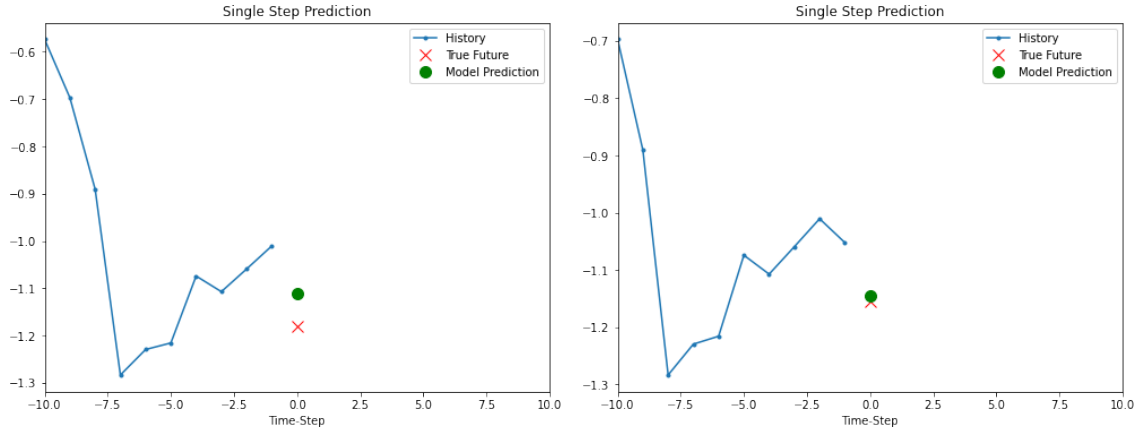


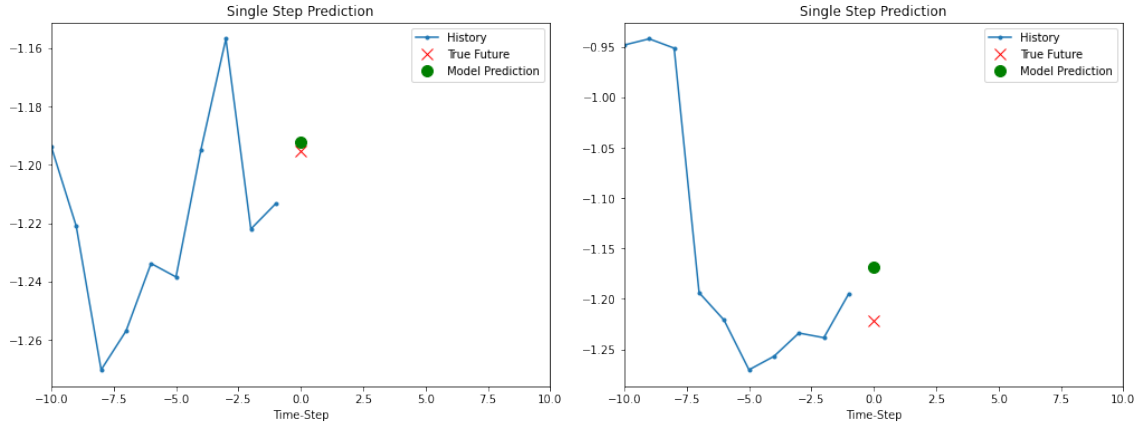
Figure 3: linear velocity prediction



I found that the velocity predictions were significantly less accurate than the position predictions, they generally offered a substantial improvement over the KF model (that was constant velocity).

However the predictions were much more erratic in terms of the rotational velocity and orientation. Which are understandably harder to predict.

Figure 4: angular velocity prediction



I think that use in combination with a Kalman Filter would smooth the variation here and provide better average results.

4.3 System Robustness

I have tested this system with up to 20 people being tracked at the same time. While it was able to handle the relatively large volume of people, I do not thing it can handle many more. Thinking about this, while I have in place a number of safeguards for the data input failure, I should have placed a maximum value to the people being tracked to prevent a memory overload. This is probably the largest area of vulnerability in the system as it is currently constructed.

I am very happy with the system's ability to handle a large data load. a number of design choices make sure to prevent memory overflow. Particularly the data is read in on a line by line basis, therefore the 1.0 gb data file does not actually have to be loaded, only 1 line at a time is in memory. Similarly the prediction data is written to log files making the theoretical limit on trial length (how long you can collect predictions) set by disk space and input data rather then RAM. Also by using the ROS framework I have been able to use the built in parallelism in ROS to make the code run more efficiently, particularly when adding in additional prediction methods. Making

it so detection, probabilistic prediction, and neural network prediction can all run on separate threads, but share data.

5 Discussion

5.1 goals completed

While I had hoped to be able to achieve both I found building a Machine Learning algorithm for prediction more interesting of a challenge particularly as I had access to a large data set of people tracks. This was coupled with the fact there did not appear to be any simple configuration for creating a people detection model with RGB-d data only models that use normal video, making the depth aspect of the camera less important. Therefore I did not end up completing the detection, in favor of exploring using machine learning for prediction.

My prediction integration with the NN was also not completed at the time of this writing, however I am hoping to have it implemented.

I was able to complete all aspects of my probabilistic prediction goals.

5.2 method effectiveness

I found it it surprising that a simple model for motion would yield such good results. In general the effectiveness of the predictions depended entirely on if a person turned. While walking in a direction without turning the predictions even out to 8 seconds often performed quite well, it was only in instances when a person would change direction or drastically change velocity that a prediction would nearly immediately loose all bearing on reality.

It was my hope that the inclusion of predictions from the neural network would help with the issues related to changing directions and velocities. based on my initial test of my model it does appear that the neural network makes an improvement from the basic model in the prediction, particularly in terms of velocity, and actually struggling the most with constant motion (where the KF is best) which leads me to believe that a combination of both methods could yield highly effective results.

I was also very happy that this system appeared to perform quite well with a (relatively) large number of people being tacked at the same time, and allowed for new people to be added or removed from the set. Due to keeping minimal data in memory (writing everything into log files) and using buffer type structures for retrieving data, along with referencing people tracks by id (rather than an un-indexed array) I was able to run the system for an arbitrarily long time, with a variable number of tracks without having any errors (in any of the test of the system that I did). this robustness of construction is one of the stronger points of this project in my opinion.

6 Conclusion

While I was not able to meet some of the goals I set out for myself I feel that I was able to create a robust people tracking and prediction system that generates data that can be used for meaningful analysis. While the prediction approach is very simplistic, it is surprisingly effective. I am very satisfied that an attention to data integrity and timeliness has allowed this project to reach a state of robustness that it can flexibly handle a large amount of input data.

7 Appendix

7.1 Project Code

The entire project can be found at `https://github.com/NickMcSweeney/hyperion`. The README contains instructions for building and running the application

7.2 Additional test trials

7.2.1 prediction data

Figure 5: trial 2

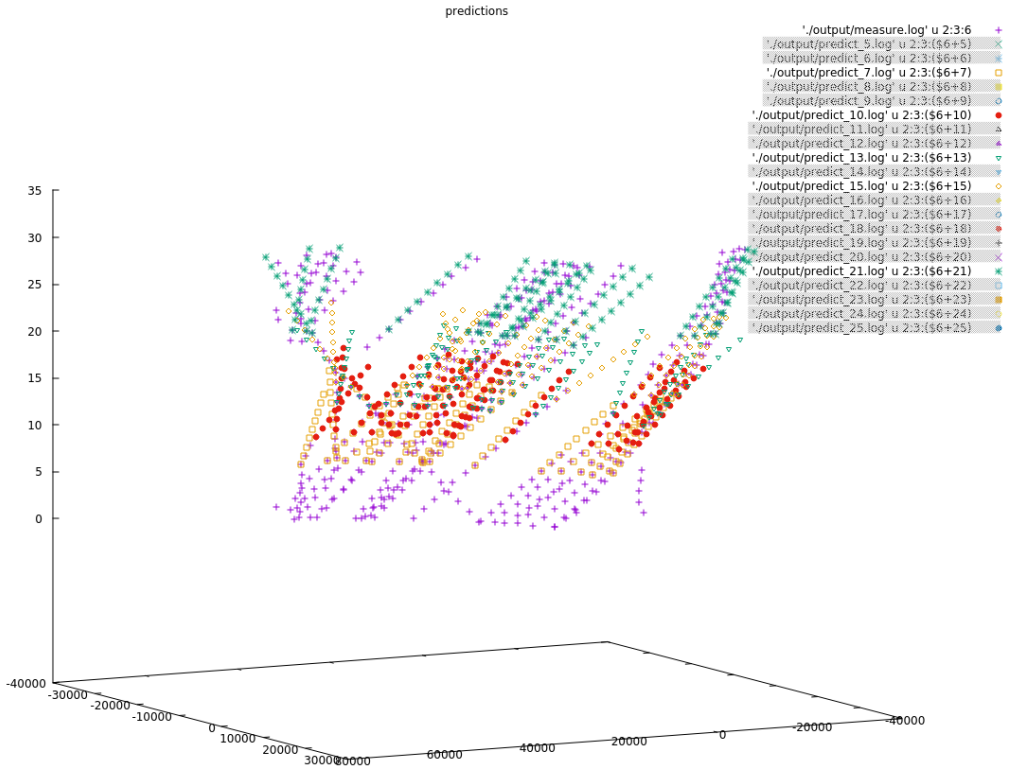
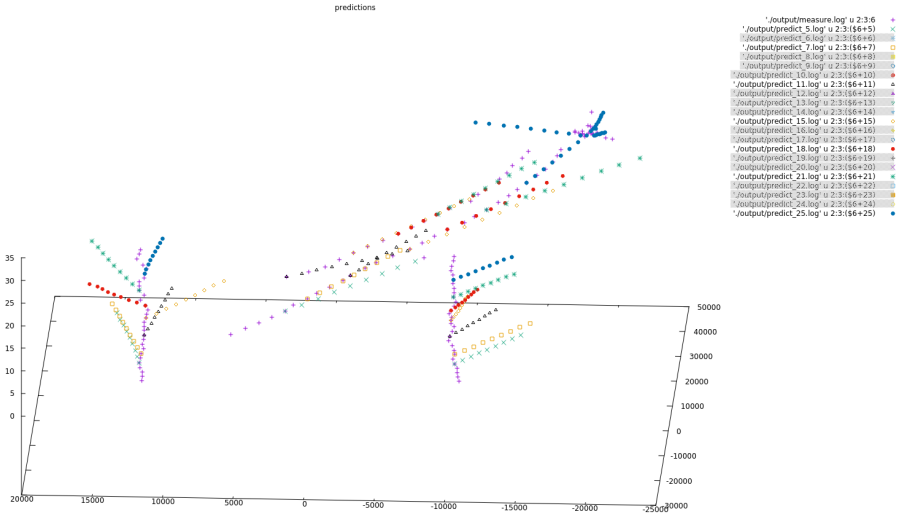


Figure 6: trial 3



7.2.2 error profile

Figure 7: trial 2

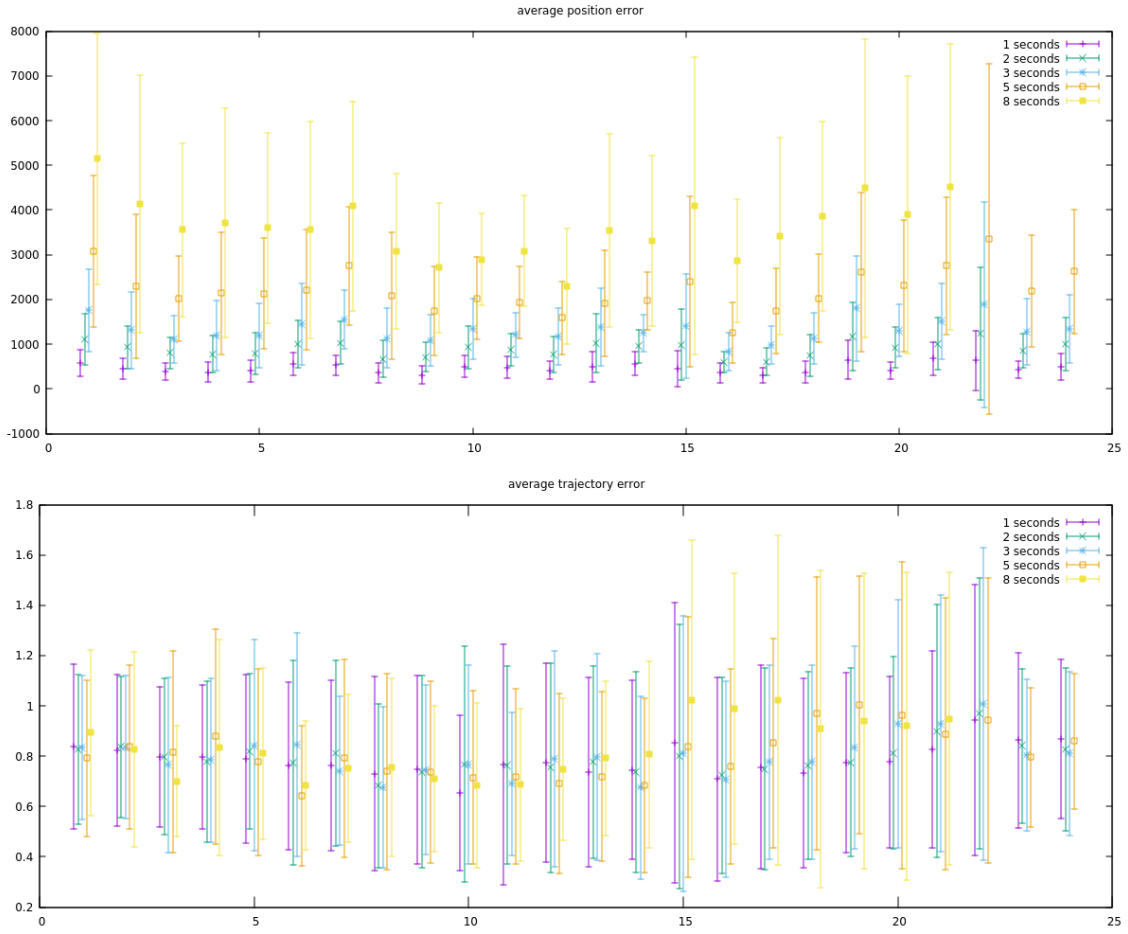
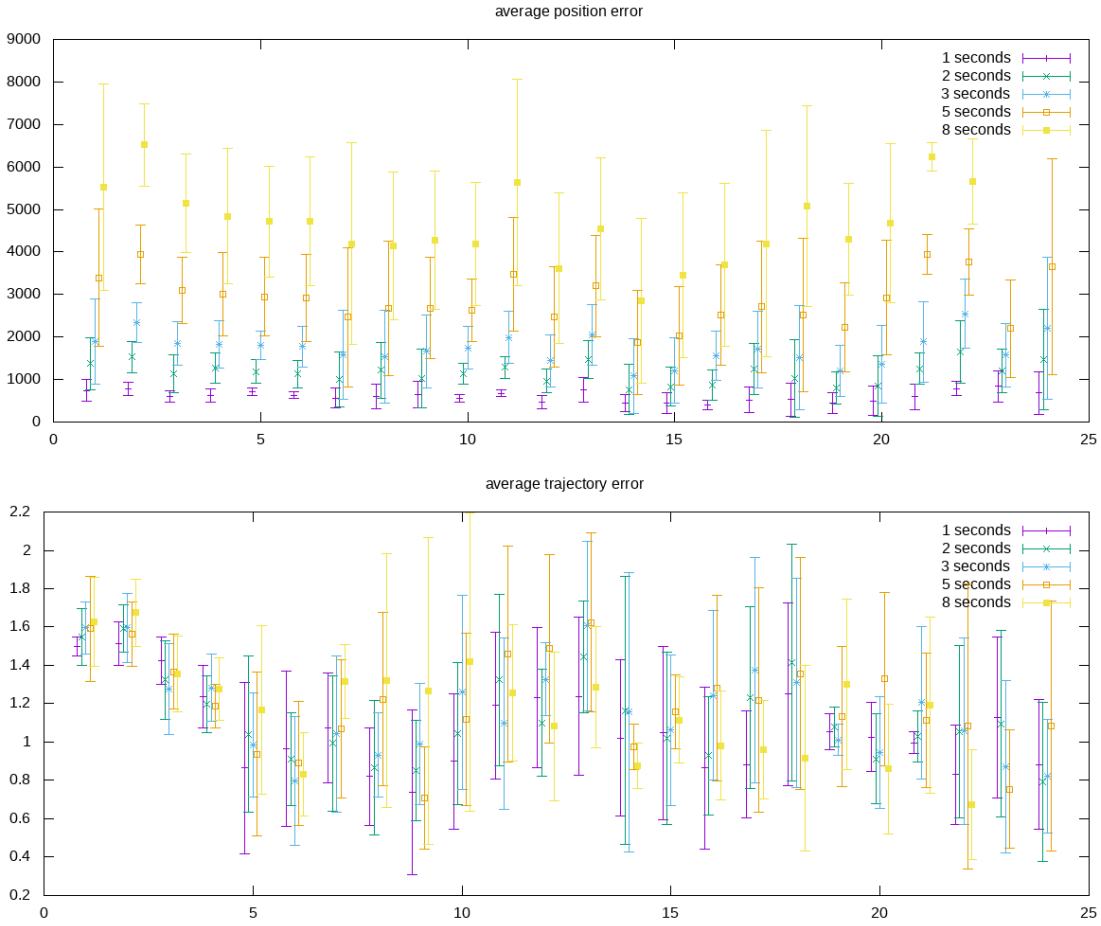
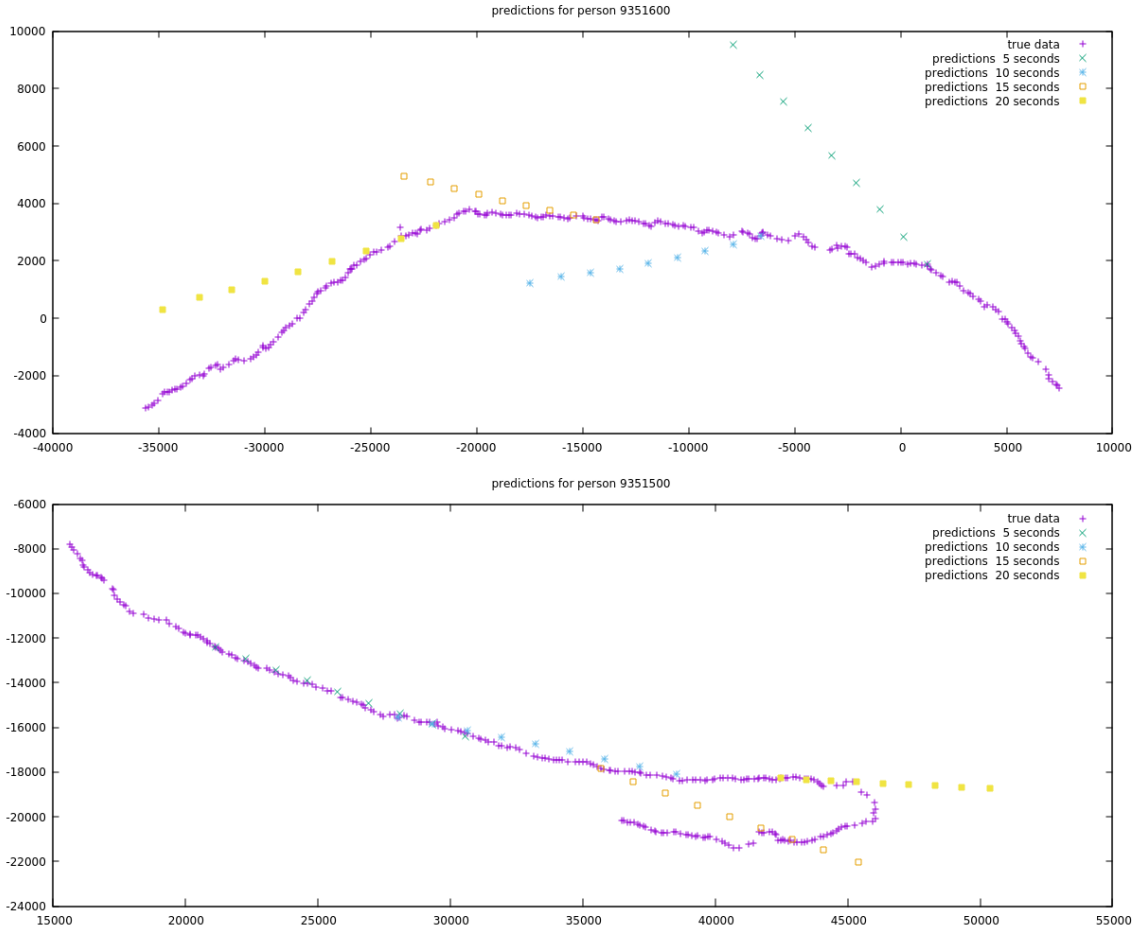


Figure 8: trial 3



7.2.3 people tracks

Figure 9: trial 2



7.2.4 motion tracks

Figure 10: trial 2

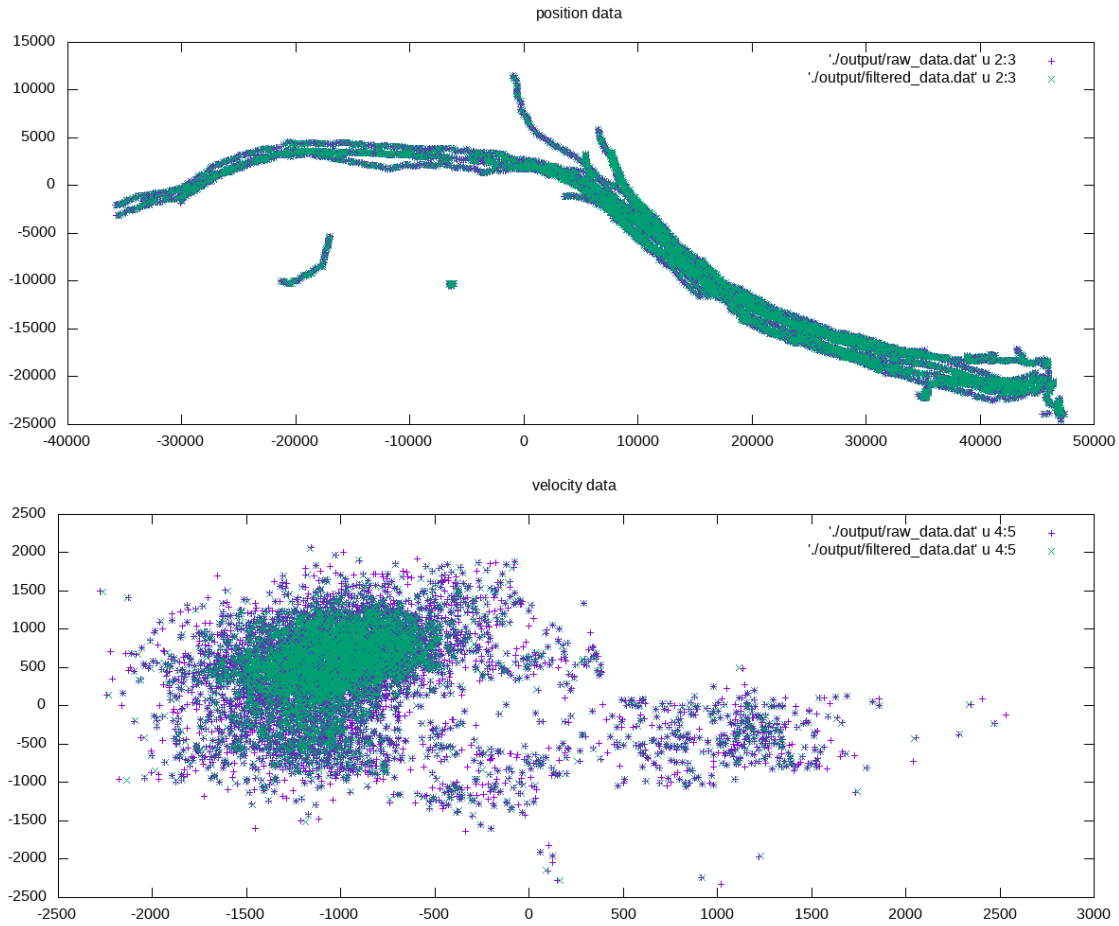
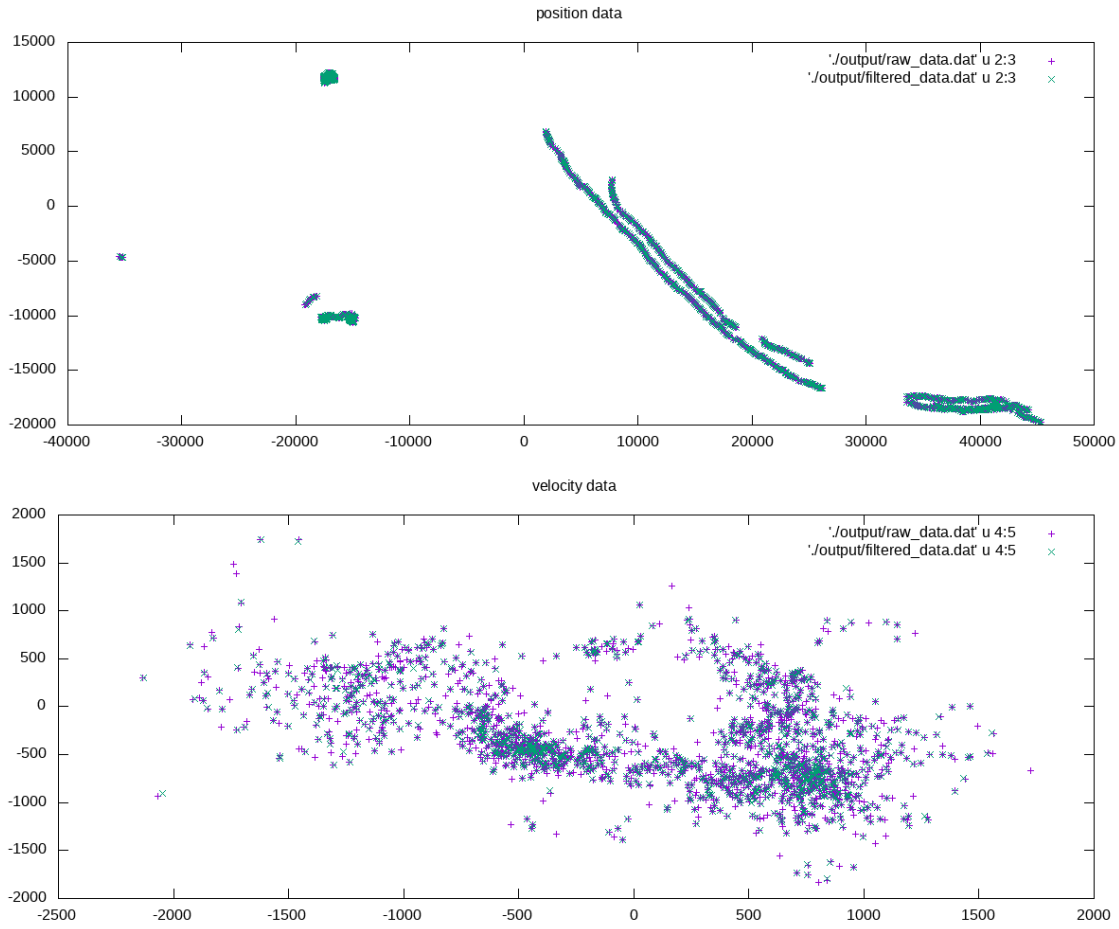


Figure 11: trial 3



7.3 Kalman Filter Full Evaluation Log

```
----- KF Movement Prediction Report -----
Test note: initial test for kf prediction


EVALUATION:

---- PREDICT DATA ----
time(1)[0] person[9315400] POS: 36906 -20147  -774.335  607.28
time(1)[0] person[9330400] POS: 28257 -16303  -1167.31  1138.96
time(1)[0] person[9330600] POS: -13990  10734 -1153.57  -136.789
time(1)[1] person[9315400] POS: 35976.8 -19418.3  -774.335  607.28
time(1)[1] person[9330400] POS: 26856.2 -14936.3  -1167.31  1138.96
time(1)[1] person[9330600] POS: -15374.3  10569.9 -1153.57  -136.789
time(1)[2] person[9315400] POS: 35047.6 -18689.5  -774.335  607.28
time(1)[2] person[9330400] POS: 25455.5 -13569.5  -1167.31  1138.96
time(1)[2] person[9330600] POS: -16758.6  10405.7 -1153.57  -136.789
time(1)[3] person[9315400] POS: 34118.4 -17960.8  -774.335  607.28
time(1)[3] person[9330400] POS: 24054.7 -12202.8  -1167.31  1138.96
time(1)[3] person[9330600] POS: -18142.8  10241.6 -1153.57  -136.789
time(1)[4] person[9315400] POS: 33189.2 -17232.1  -774.335  607.28
time(1)[4] person[9330400] POS: 22653.9 -10836  -1167.31  1138.96
time(1)[4] person[9330600] POS: -19527.1  10077.4 -1153.57  -136.789
time(1)[5] person[9315400] POS: 32260 -16503.3  -774.335  607.28
time(1)[5] person[9330400] POS: 21253.2 -9469.26  -1167.31  1138.96
time(1)[5] person[9330600] POS: -20911.4  9913.26 -1153.57  -136.789
time(1)[6] person[9315400] POS: 31330.8 -15774.6  -774.335  607.28
time(1)[6] person[9330400] POS: 19852.4 -8102.51  -1167.31  1138.96
```

```
time(1)[6] person[9330600] POS: -22295.7  9749.12 -1153.57  -136.789
time(1)[7] person[9315400] POS: 30401.6 -15045.8  -774.335  607.28
time(1)[7] person[9330400] POS: 18451.6 -6735.77  -1167.31  1138.96
time(1)[7] person[9330600] POS: -23680  9584.97 -1153.57  -136.789
time(1)[8] person[9315400] POS: 29472.4 -14317.1  -774.335  607.28
time(1)[8] person[9330400] POS: 17050.9 -5369.02  -1167.31  1138.96
time(1)[8] person[9330600] POS: -25064.3  9420.82 -1153.57  -136.789

EVALUATION:
time(1)[0] person[9315400] diff: 0.000000 -0.000000 -0.000000 0.000000
time(1)[1] person[9315400] diff: 0.004117 -0.005992 -0.329577 0.030570
time(1)[2] person[9315400] diff: 0.008010 -0.019472 -0.001658 0.757093
time(1)[3] person[9315400] diff: 0.015221 -0.045498 -0.497826 0.146725
time(1)[4] person[9315400] diff: 0.022400 -0.063216 -0.178072 0.390334
time(1)[5] person[9315400] diff: 0.035267 -0.077731 -0.228801 0.086278
time(1)[6] person[9315400] diff: 0.053307 -0.091259 -0.449132 0.233751
time(1)[7] person[9315400] diff: 0.063343 -0.117263 -0.297337 0.375995
time(1)[8] person[9315400] diff: 0.070403 -0.155019 -0.299739 0.752560
time(1)[0] person[9330400] diff: 0.000000 -0.000000 -0.000000 0.000000
time(1)[1] person[9330400] diff: 0.008299 -0.067268 -0.115760 0.559705
time(1)[2] person[9330400] diff: 0.002380 -0.112759 -0.021316 0.580629
time(1)[3] person[9330400] diff: 0.006105 -0.180167 -0.128768 0.424982
time(1)[4] person[9330400] diff: 0.004892 -0.263809 -0.050845 0.471546
time(1)[5] person[9330400] diff: 0.002904 -0.347803 -0.142188 0.092263
time(1)[6] person[9330400] diff: 0.007857 -0.450875 -0.003527 0.828070
time(1)[7] person[9330400] diff: 0.030390 -0.589794 -0.225928 1.064049
time(1)[8] person[9330400] diff: 0.011841 -0.729829 -0.407044 0.322966
time(1)[0] person[9330600] diff: -0.000000  0.000000  -0.000000 -0.000000
time(1)[1] person[9330600] diff: -0.025513  0.050454  -0.029164 3.254099
time(1)[2] person[9330600] diff: -0.067138  0.054528  -0.605353 -1.146307
time(1)[3] person[9330600] diff: -0.143791  0.113243  -1.206060 2.832688
time(1)[4] person[9330600] diff: -0.215961  0.224434  -2.626902 2.479397
time(1)[5] person[9330600] diff: -0.297426  0.334484  -3.291302 2.528438
time(1)[6] person[9330600] diff: -0.376706  0.426681  -5.482370 2.707220
time(1)[7] person[9330600] diff: -0.450838  0.501586  -4.529768 2.703739
time(1)[8] person[9330600] diff: -0.515719  0.592953  -2.351571 2.522798

---- PREDICT DATA ----
time(2)[0] person[9315400] POS: 35829 -19535  -1079.89  626.133
time(2)[0] person[9330400] POS: 27080 -15976  -1310.74  640.87
time(2)[0] person[9330600] POS: -14987  11117 -1187.71  573.083
time(2)[1] person[9315400] POS: 34647.5 -18842.1  -1074.01  625.77
time(2)[1] person[9330400] POS: 25632.4 -15236.1  -1307.98  650.452
time(2)[1] person[9330600] POS: -16304.6  11721.3 -1187.05  559.427
time(2)[2] person[9315400] POS: 33477.1 -18148.8  -1068.25  625.414
time(2)[2] person[9330400] POS: 24188.7 -14473.8  -1305.28  659.85
time(2)[2] person[9330600] POS: -17622.9  12296.1 -1186.41  546.033
time(2)[3] person[9315400] POS: 32317.6 -17455.3  -1062.59  625.065
time(2)[3] person[9330400] POS: 22748.8 -13689.8  -1302.62  669.067
time(2)[3] person[9330600] POS: -18941.7  12842.3 -1185.78  532.897
time(2)[4] person[9315400] POS: 31168.6 -16761.5  -1057.05  624.723
time(2)[4] person[9330400] POS: 21312.4 -12884.6  -1300.02  678.107
time(2)[4] person[9330600] POS: -20261  13360.6 -1185.16  520.014
time(2)[5] person[9315400] POS: 30029.9 -16067.4  -1051.61  624.388
time(2)[5] person[9330400] POS: 19879.6 -12058.8  -1297.47  686.973
time(2)[5] person[9330600] POS: -21581  13851.8 -1184.55  507.378
time(2)[6] person[9315400] POS: 28901 -15372.9  -1046.27  624.058
time(2)[6] person[9330400] POS: 18450.1 -11213.1  -1294.96  695.668
time(2)[6] person[9330600] POS: -22901.5  14316.9 -1183.95  494.986
time(2)[7] person[9315400] POS: 27781.8 -14678.2  -1041.04  623.736
time(2)[7] person[9330400] POS: 17023.9 -10348  -1292.51  704.196
time(2)[7] person[9330600] POS: -24222.6  14756.4 -1183.37  482.832
time(2)[8] person[9315400] POS: 26532.5 -13929.7  -1041.04  623.736
time(2)[8] person[9330400] POS: 15472.9 -9502.94  -1292.51  704.196
time(2)[8] person[9330600] POS: -25642.7  15335.8 -1183.37  482.832

EVALUATION:
time(2)[0] person[9315400] diff: 0.000000 -0.000000 -0.000000 0.000000
time(2)[1] person[9315400] diff: 0.003472 -0.011341 -0.322648 0.782641
time(2)[2] person[9315400] diff: 0.003754 -0.035089 -0.186194 0.175957
```

```
time(2)[3] person[9315400] diff: 0.004212 -0.050357 -0.137697 0.418021
time(2)[4] person[9315400] diff: 0.000854 -0.062226 -0.528217 0.114524
time(2)[5] person[9315400] diff: 0.010912 -0.072899 -0.150560 0.206306
time(2)[6] person[9315400] diff: 0.012745 -0.095817 -0.001425 0.402216
time(2)[7] person[9315400] diff: 0.011359 -0.130237 -0.006043 0.775395
time(2)[8] person[9315400] diff: 0.011810 -0.153949 -0.003136 0.933116
time(2)[0] person[9330400] diff: 0.000000 -0.000000 -0.000000 0.000000
time(2)[1] person[9330400] diff: 0.009305 -0.002964 -0.092400 0.037612
time(2)[2] person[9330400] diff: 0.000550 -0.009982 -0.239507 0.114177
time(2)[3] person[9330400] diff: 0.000712 -0.031576 -0.160187 0.051388
time(2)[4] person[9330400] diff: 0.000122 -0.043385 -0.034747 0.419899
time(2)[5] person[9330400] diff: 0.006488 -0.061115 -0.109133 0.371021
time(2)[6] person[9330400] diff: 0.030471 -0.098113 -0.327694 0.666475
time(2)[7] person[9330400] diff: 0.013426 -0.108832 -0.308443 0.154687
time(2)[8] person[9330400] diff: 0.021931 -0.096265 -0.343370 0.400641
time(2)[0] person[9330600] diff: -0.000000  0.000000  -0.000000 0.000000
time(2)[1] person[9330600] diff: -0.039694  0.064491  -0.631227 38.473665
time(2)[2] person[9330600] diff: -0.114839  0.069432  -1.223772 0.369951
time(2)[3] person[9330600] diff: -0.185831  0.017065  -2.607288 0.822995
time(2)[4] person[9330600] diff: -0.266458  0.039214  -3.246160 0.770815
time(2)[5] person[9330600] diff: -0.345187  0.082053  -5.315797 0.568588
time(2)[6] person[9330600] diff: -0.418992  0.111158  -4.425492 0.595061
time(2)[7] person[9330600] diff: -0.483694  0.162192  -2.341961 0.840887
time(2)[8] person[9330600] diff: -0.556428  0.208215  -2.343998 0.887805

---- PREDICT DATA ----
time(3)[0] person[9315400] POS: 34768 -19057  -775.62 273.764
time(3)[0] person[9330400] POS: 25395 -15191  -1192.46  626.439
time(3)[0] person[9330600] POS: -15670  10989 -617.507  -504.139
time(3)[0] person[9331400] POS: -35809  -3400 1178.06 308.588
time(3)[1] person[9315400] POS: 33904.3 -18743  -781.25 280.529
time(3)[1] person[9330400] POS: 24082.8 -14500.1  -1194.63  627.081
time(3)[1] person[9330600] POS: -16376  10471.3 -628.452  -483.936
time(3)[1] person[9331400] POS: -34395.3  -3029.69  1178.06 308.588
time(3)[2] person[9315400] POS: 33028.9 -18414.4  -786.662  287.157
time(3)[2] person[9330400] POS: 22765.8 -13806.7  -1196.71  627.889
time(3)[2] person[9330600] POS: -17105.7  9995.86 -639.173  -464.374
time(3)[2] person[9331400] POS: -32981.7  -2659.39  1178.06 308.588
time(3)[3] person[9315400] POS: 32142.4 -18071.6  -791.864  293.651
time(3)[3] person[9330400] POS: 21444.3 -13110.3  -1198.69  628.855
time(3)[3] person[9330600] POS: -17858.2  9560.61 -649.677  -445.437
time(3)[3] person[9331400] POS: -31568  -2289.08  1178.06 308.588
time(3)[4] person[9315400] POS: 31245.5 -17715  -796.861  300.013
time(3)[4] person[9330400] POS: 20118.5 -12410.3  -1200.59  629.973
time(3)[4] person[9330600] POS: -18633  9163.58 -659.967  -427.106
time(3)[4] person[9331400] POS: -30154.3  -1918.78  1178.06 308.588
time(3)[5] person[9315400] POS: 30338.7 -17345.1  -801.659  306.247
time(3)[5] person[9330400] POS: 18788.7 -11706.3  -1202.41  631.237
time(3)[5] person[9330600] POS: -19429.4  8802.9  -670.047  -409.367
time(3)[5] person[9331400] POS: -28740.6  -1548.47  1178.06 308.588
time(3)[6] person[9315400] POS: 29422.6 -16962.2  -806.264  312.355
time(3)[6] person[9330400] POS: 17455 -10997.8  -1204.14  632.641
time(3)[6] person[9330600] POS: -20246.8  8476.72 -679.922  -392.203
time(3)[6] person[9331400] POS: -27327  -1178.17  1178.06 308.588
time(3)[7] person[9315400] POS: 28495.2 -16565.7  -810.781  318.345
time(3)[7] person[9330400] POS: 16115.3 -10285.2  -1205.84  634.017
time(3)[7] person[9330600] POS: -21086.4  8186.47 -689.608  -375.37
time(3)[7] person[9331400] POS: -25913.3  -807.861  1178.06 308.588
time(3)[8] person[9315400] POS: 27522.2 -16183.7  -810.781  318.345
time(3)[8] person[9330400] POS: 14668.3 -9524.35  -1205.84  634.017
time(3)[8] person[9330600] POS: -21913.9  7736.03 -689.608  -375.37
time(3)[8] person[9331400] POS: -24499.6  -437.555  1178.06 308.588

EVALUATION:
time(3)[0] person[9315400] diff: 0.000000 -0.000000 -0.000000 0.000000
time(3)[1] person[9315400] diff: 0.008926 -0.002877 -0.489477 0.605714
time(3)[2] person[9315400] diff: 0.017559 -0.003122 -0.162392 0.349917
time(3)[3] person[9315400] diff: 0.031616 -0.013010 -0.250864 0.619255
time(3)[4] person[9315400] diff: 0.050583 -0.024687 -0.421816 0.876389
time(3)[5] person[9315400] diff: 0.061274 -0.024812 -0.263340 0.301755
```

```
time(3)[6] person[9315400] diff: 0.068714 -0.014202 -0.260125 0.126419
time(3)[7] person[9315400] diff: 0.083127 -0.019056 -0.251771 0.335530
time(3)[8] person[9315400] diff: 0.095684 -0.023045 -0.460655 0.113156
time(3)[0] person[9330400] diff: 0.000000 -0.000000 -0.000000 0.000000
time(3)[1] person[9330400] diff: 0.004937 -0.008166 -0.151789 0.164863
time(3)[2] person[9330400] diff: 0.000035 -0.023074 -0.075694 0.114794
time(3)[3] person[9330400] diff: 0.006048 -0.026025 -0.115771 0.491380
time(3)[4] person[9330400] diff: 0.005458 -0.032399 -0.031636 0.286759
time(3)[5] person[9330400] diff: 0.012288 -0.055133 -0.255125 0.578725
time(3)[6] person[9330400] diff: 0.011582 -0.048028 -0.377175 0.260659
time(3)[7] person[9330400] diff: 0.018746 -0.017234 -0.275630 0.500271
time(3)[8] person[9330400] diff: 0.001274 -0.028146 -0.265940 0.586824
time(3)[0] person[9330600] diff: -0.000000  0.000000  -0.000000 -0.000000
time(3)[1] person[9330600] diff: -0.041577  0.091121  -0.749979 8.245363
time(3)[2] person[9330600] diff: -0.084364  0.232453  -3.294728 4.282620
time(3)[3] person[9330600] diff: -0.141579  0.369582  -5.058695 4.451588
time(3)[4] person[9330600] diff: -0.201116  0.485395  -19.447694  5.534512
time(3)[5] person[9330600] diff: -0.259403  0.580458  -10.011778  5.243018
time(3)[6] person[9330600] diff: -0.311622  0.687698  -2.635386 3.982884
time(3)[7] person[9330600] diff: -0.371497  0.791061  -2.629036 3.709530
time(3)[8] person[9330600] diff: -0.411440  0.890058  -3.079115 3.533593
time(3)[0] person[9331400] diff: -0.000000  -0.000000 0.000000  0.000000
time(3)[1] person[9331400] diff: -0.003068  -0.092642 0.070104  8.595660
time(3)[2] person[9331400] diff: -0.009094  -0.155800 0.199839  0.807021
time(3)[3] person[9331400] diff: -0.023263  -0.328028 0.676954  0.658705
time(3)[4] person[9331400] diff: -0.019074  -0.369025 0.202341  0.041003
time(3)[5] person[9331400] diff: -0.013698  -0.810966 0.018510  0.347557
time(3)[6] person[9331400] diff: -0.026753  -2.539411 0.213901  0.891262
time(3)[7] person[9331400] diff: -0.014318  -32.427973  0.265020  0.538026
time(3)[8] person[9331400] diff: -0.024966  5.268720  0.219734  0.224595
```

***

```
--------------------------------- RESULT: ---------------------------------

+++ prediction @ 1seconds:

person track 9315400:
pos X :
Mean Absolute Error: 1026.35
Mean Absolute Percent Error: 4.29186%
stdev: 251.447
pos Y :
Mean Absolute Error: 696.525
Mean Absolute Percent Error: 4.85975%
stdev: 348.148
vel X :
Mean Absolute Error: 5.27708
Mean Absolute Percent Error: 0.567964%
stdev: 6.69137
vel Y :
Mean Absolute Error: 3.82604
Mean Absolute Percent Error: 0.608618%
stdev: 4.87357

person track 9330400:
pos X :
Mean Absolute Error: 1287.77
Mean Absolute Percent Error: 9.54769%
stdev: 319.96
pos Y :
Mean Absolute Error: 939.373
Mean Absolute Percent Error: 11.9606%
stdev: 401.646
vel X :
Mean Absolute Error: 5.2515
Mean Absolute Percent Error: 0.4503%
stdev: 6.70564
```

```
vel Y :
Mean Absolute Error: 5.33967
Mean Absolute Percent Error: 0.630789%
stdev: 6.67641

person track 9330600:
pos X :
Mean Absolute Error: 515.455
Mean Absolute Percent Error: 3.42652%
stdev: 655.361
pos Y :
Mean Absolute Error: 997.927
Mean Absolute Percent Error: 6.93557%
stdev: 630.786
vel X :
Mean Absolute Error: 4.05393
Mean Absolute Percent Error: 0.891954%
stdev: 4.91074
vel Y :
Mean Absolute Error: 7.62464
Mean Absolute Percent Error: 0.832511%
stdev: 10.7305

person track 9331400:
pos X :
Mean Absolute Error: 1283.94
Mean Absolute Percent Error: 4.29959%
stdev: 293.266
pos Y :
Mean Absolute Error: 505.585
Mean Absolute Percent Error: 32.0295%
stdev: 282.655
vel X :
Mean Absolute Error: 5.5369
Mean Absolute Percent Error: 0.479001%
stdev: 7.32529
vel Y :
Mean Absolute Error: 5.7022
Mean Absolute Percent Error: 1.23853%
stdev: 7.39782

person track 9331600:
pos X :
Mean Absolute Error: 1235.7
Mean Absolute Percent Error: 4.23514%
stdev: 327.224
pos Y :
Mean Absolute Error: 567.883
Mean Absolute Percent Error: 3.21499%
stdev: 219.081
vel X :
Mean Absolute Error: 4.94256
Mean Absolute Percent Error: 0.444329%
stdev: 7.25941
vel Y :
Mean Absolute Error: 3.62567
Mean Absolute Percent Error: 0.708799%
stdev: 4.54539

person track 9332600:
pos X :
Mean Absolute Error: 1101.9
Mean Absolute Percent Error: 28.2084%
stdev: 800.053
pos Y :
Mean Absolute Error: 598.626
Mean Absolute Percent Error: 9.24286%
stdev: 377.629
vel X :
Mean Absolute Error: 9.53945
```

```
Mean Absolute Percent Error: 0.950992%
stdev: 11.6988
vel Y :
Mean Absolute Error: 4.23236
Mean Absolute Percent Error: 0.794242%
stdev: 5.59359

person track 9333700:
pos X :
Mean Absolute Error: 552.5
Mean Absolute Percent Error: 3.31334%
stdev: 0
pos Y :
Mean Absolute Error: 1104.6
Mean Absolute Percent Error: 8.68601%
stdev: 0
vel X :
Mean Absolute Error: 0
Mean Absolute Percent Error: 0%
stdev: 0
vel Y :
Mean Absolute Error: 0
Mean Absolute Percent Error: 0%
stdev: 0

person track 9333701:
pos X :
Mean Absolute Error: 1173.9
Mean Absolute Percent Error: 7.3705%
stdev: 0
pos Y :
Mean Absolute Error: 220.8
Mean Absolute Percent Error: 1.90164%
stdev: 0
vel X :
Mean Absolute Error: 0
Mean Absolute Percent Error: 0%
stdev: 0
vel Y :
Mean Absolute Error: 0
Mean Absolute Percent Error: 0%
stdev: 0

person track 9334200:
pos X :
Mean Absolute Error: 0
Mean Absolute Percent Error: -nan%
stdev: -nan
pos Y :
Mean Absolute Error: 0
Mean Absolute Percent Error: -nan%
stdev: -nan
vel X :
Mean Absolute Error: 0
Mean Absolute Percent Error: -nan%
stdev: -nan
vel Y :
Mean Absolute Error: 0
Mean Absolute Percent Error: -nan%
stdev: -nan
```

***

```
+++ prediction @ 5seconds:

person track 9315400:
pos X :
Mean Absolute Error: 5129.8
```

```
Mean Absolute Percent Error: 21.4512%
stdev: 1159.44
pos Y :
Mean Absolute Error: 3480.17
Mean Absolute Percent Error: 24.2816%
stdev: 1700.49
vel X :
Mean Absolute Error: 24.8089
Mean Absolute Percent Error: 2.67014%
stdev: 31.5293
vel Y :
Mean Absolute Error: 18.4406
Mean Absolute Percent Error: 2.93339%
stdev: 23.388

person track 9330400:
pos X :
Mean Absolute Error: 6423.95
Mean Absolute Percent Error: 47.628%
stdev: 1495.08
pos Y :
Mean Absolute Error: 4694.66
Mean Absolute Percent Error: 59.7751%
stdev: 1913.72
vel X :
Mean Absolute Error: 25.036
Mean Absolute Percent Error: 2.14676%
stdev: 31.8748
vel Y :
Mean Absolute Error: 25.6723
Mean Absolute Percent Error: 3.03273%
stdev: 31.9409

person track 9330600:
pos X :
Mean Absolute Error: 2593.75
Mean Absolute Percent Error: 17.2422%
stdev: 3283.26
pos Y :
Mean Absolute Error: 4856.33
Mean Absolute Percent Error: 33.7514%
stdev: 3065.96
vel X :
Mean Absolute Error: 20.028
Mean Absolute Percent Error: 4.4066%
stdev: 24.011
vel Y :
Mean Absolute Error: 36.5413
Mean Absolute Percent Error: 3.98983%
stdev: 50.7596

person track 9331400:
pos X :
Mean Absolute Error: 6422.52
Mean Absolute Percent Error: 21.5074%
stdev: 1362.73
pos Y :
Mean Absolute Error: 2482.96
Mean Absolute Percent Error: 157.299%
stdev: 1304.84
vel X :
Mean Absolute Error: 26.0749
Mean Absolute Percent Error: 2.25576%
stdev: 34.7407
vel Y :
Mean Absolute Error: 26.7501
Mean Absolute Percent Error: 5.81017%
stdev: 35.0917

person track 9331600:
```

```
pos X :
Mean Absolute Error: 6201.62
Mean Absolute Percent Error: 21.2549%
stdev: 1562.57
pos Y :
Mean Absolute Error: 2845.36
Mean Absolute Percent Error: 16.1086%
stdev: 1041.18
vel X :
Mean Absolute Error: 23.3333
Mean Absolute Percent Error: 2.09764%
stdev: 34.1429
vel Y :
Mean Absolute Error: 17.2202
Mean Absolute Percent Error: 3.36647%
stdev: 21.6201

person track 9332600:
pos X :
Mean Absolute Error: 5430.56
Mean Absolute Percent Error: 139.022%
stdev: 3905.21
pos Y :
Mean Absolute Error: 3010.79
Mean Absolute Percent Error: 46.4869%
stdev: 1857.08
vel X :
Mean Absolute Error: 46.3017
Mean Absolute Percent Error: 4.61584%
stdev: 56.8231
vel Y :
Mean Absolute Error: 20.2544
Mean Absolute Percent Error: 3.80092%
stdev: 27.0037

person track 9333700:
pos X :
Mean Absolute Error: 2762.7
Mean Absolute Percent Error: 16.5679%
stdev: 0
pos Y :
Mean Absolute Error: 5523.19
Mean Absolute Percent Error: 43.4315%
stdev: 0
vel X :
Mean Absolute Error: 0
Mean Absolute Percent Error: 0%
stdev: 0
vel Y :
Mean Absolute Error: 0
Mean Absolute Percent Error: 0%
stdev: 0

person track 9333701:
pos X :
Mean Absolute Error: 5869.7
Mean Absolute Percent Error: 36.8538%
stdev: 0
pos Y :
Mean Absolute Error: 1103.9
Mean Absolute Percent Error: 9.50736%
stdev: 0
vel X :
Mean Absolute Error: 0
Mean Absolute Percent Error: 0%
stdev: 0
vel Y :
Mean Absolute Error: 0
Mean Absolute Percent Error: 0%
stdev: 0
```

```
person track 9334200:
pos X :
Mean Absolute Error: 0
Mean Absolute Percent Error: -nan%
stdev: -nan
pos Y :
Mean Absolute Error: 0
Mean Absolute Percent Error: -nan%
stdev: -nan
vel X :
Mean Absolute Error: 0
Mean Absolute Percent Error: -nan%
stdev: -nan
vel Y :
Mean Absolute Error: 0
Mean Absolute Percent Error: -nan%
```