COL864 - Assignment 1

Special Topics in AI

Gurarmaan S. Panjeta
2020CS50426

Feat. Radars and Localisation

September 19, 2023

September 19, 2023

Plots

The Plots have been uploaded on google drive, accessible via GDrive_plots.

If the above link doesn't work, kindly open :

"https://drive.google.com/drive/folders/1bMSKxHuA2BFccCWvGDtmowpGfYLcC5Oa?usp=sharing"

Q1

State Estimation Using Kalman Filters

Part A

The Motion Model

The laws of physics govern motion of an object given it's state variables. In outcase, these variables are the planes's coordinates x_t, y_t, z_t and velocity components $\dot{x}_t, \dot{y}_t, , \dot{z}_t,$.

After each time step, the location changes to $x_i + v_i * \Delta t$ in that direction, and the velocity changes to $v_t + \delta_t$ in that direction!

Based on this, the Linear Kalman Model for Motion is

$$X_{t+1} = A_t X_t + B_t u_t + \epsilon_t$$

, where,

$$X_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ z_{t+1} \\ \dot{x_{t+1}} \\ \dot{y_{t+1}} \\ \dot{z_{t+1}} \end{bmatrix}, A_t = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, X_t = \begin{bmatrix} x_t \\ y_t \\ z_t \\ \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \end{bmatrix}$$

$$B_t = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, u_t = \begin{bmatrix} \delta_{\dot{x}_t} \\ \delta_{\dot{y}_t} \\ \delta_{\dot{x}_t} \end{bmatrix} \text{ and } \epsilon_t = \mathcal{N}(0, R) \text{ with}$$

$$R = \begin{bmatrix} \sigma_{rx}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{ry}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{rx^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{r\dot{x}^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{r\dot{y}^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{r\dot{x}^2} \end{bmatrix} \text{ where } \sigma_{ri} = 1.0 \text{ and } \sigma_{r\dot{i}} = 0.008$$

The Observation Model

Our measurements give us x'_t, y'_t, z'_t, which are noisy observations of the positional coordinates, and hence the Observation model can be written as

$$Z_t = C_t X_t + d_t + \delta_t$$

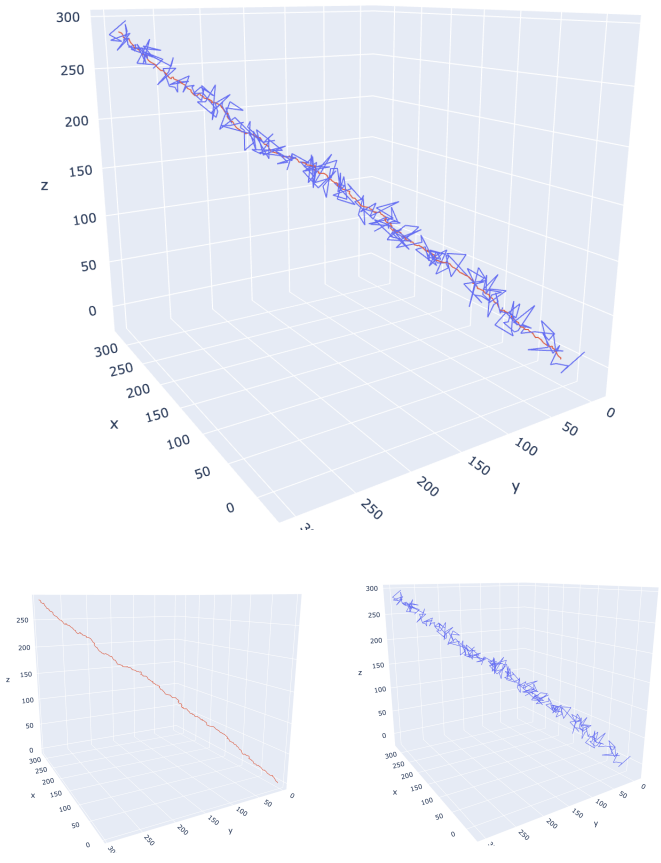
, where,

$$Z_t = \begin{bmatrix} x'_t \\ y'_t \\ z'_t \end{bmatrix}, C_t = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, X_t = \begin{bmatrix} x_t \\ y_t \\ z_t \\ \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \end{bmatrix}$$

$$d_t = \mathbf{0} \text{ and} \delta_t = \mathcal{N}(0, Q), \text{ where } Q = \sigma_s^2 I_{3*3} \text{ with } \sigma_s = 8.$$

because there is no default offset in the sensors, $d_t = \mathbf{0}$, and is hence ignored beyond this point.

For the given instantiation of the path variables, the actual trajectory, the actual path and the observations are plotted below



Part B

The Kalman Updates are carried out using the standard algorithm discussed in class,

Previous belief action observation

Algorithm Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)**:**

$$\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$$
$$\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$$
$$K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$$
$$\Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$$
return μ_t, Σ_t

Belief is gaussian

The initial μ_{init} is initialised to all 0's (because the object is at rest and at origin. The Prior belief (Σ_{init})is an isotroic gaussian (6*6) with diagonal values = $\sigma^2, \sigma = 0.008$.

The controls are sinusoidal, hence for the ith iteration, the control vector

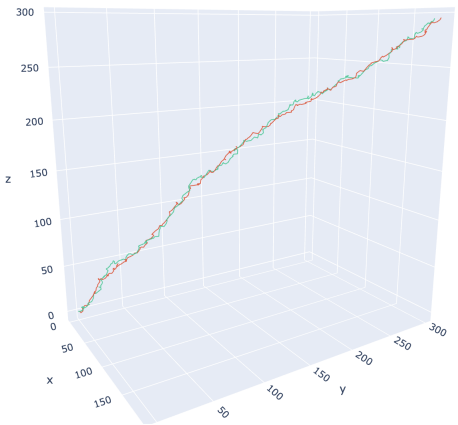
$$u_t = \begin{bmatrix} \delta_{\dot{x}_t} \\ \delta_{\dot{y}_t} \\ \delta_{\dot{x}_t} \end{bmatrix} = \begin{bmatrix} \cos i \\ \sin i \\ \sin i \end{bmatrix}$$

is made to act on the latent state.

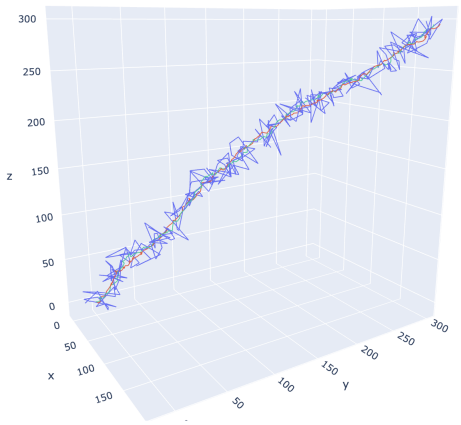
Part C

The Instantiation in Part B yields the following plots

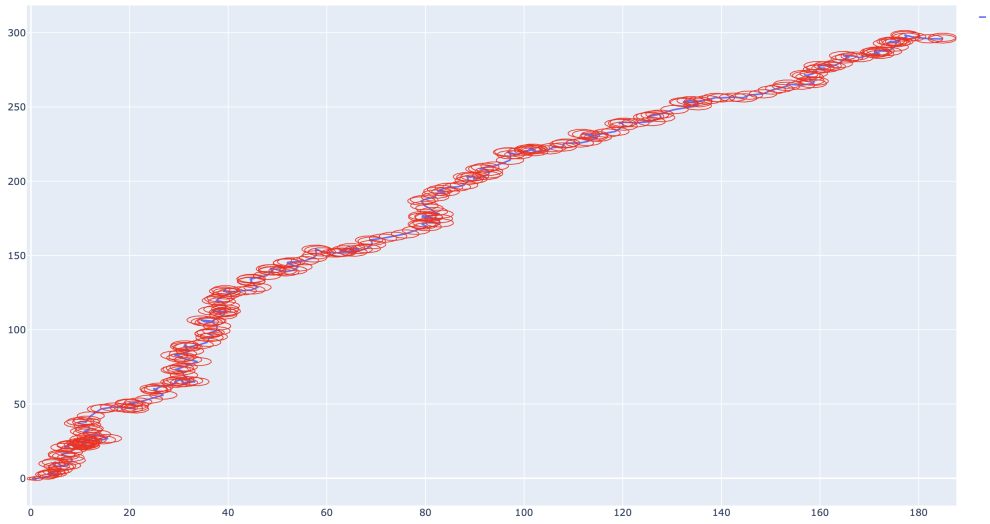
Actual and Estimated Trajectories!



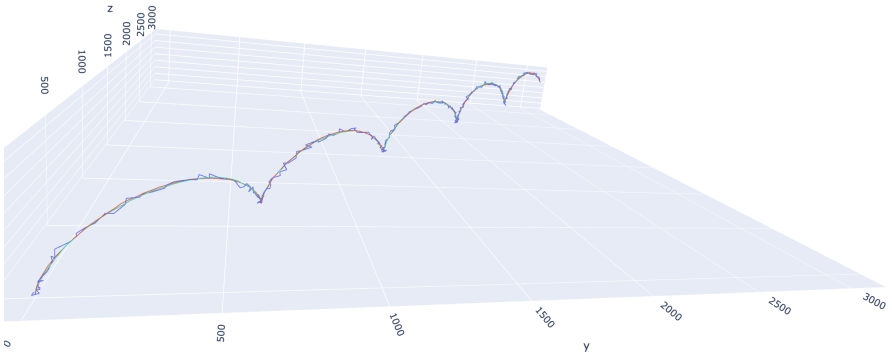
Actual and Estimated Trajectories, with Noisy Observations!



Motion Projected onto the XY plane, along with uncertainty ellipses!



Kindly note that for this and all subsequent parts, there is a parameter of Δt, the time after which the update happens. *I have assumed this to be 1.0,* since it gives nice linear-ish plots. An alternate timestamp could yield plots that are shaped like the following, and I also attach some of these in the reports as alternates.



Part D

Incrementing the 3 noise parameters increases the delta between the actual trajectory and the estimated trajectory.

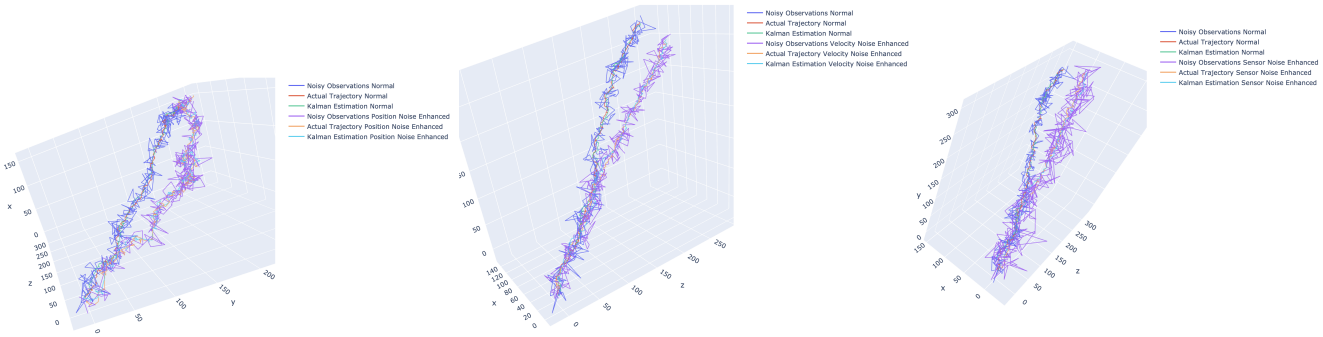
Sensor Noise (σ_s) - Increasing this noise feeds the estimator with samples that are more noisy and away from the actual trajectory. Since the noise's mean is still 0, the estimated trajectory still remains more or less the same, but our confidence in the estimation decreases (higher deviation) than the normal case.

Positional Update Noise (σ_{rx_i}) - Increasing this flavour of noise makes the latent state variables of location more erratic. This is then fed into the estimator via the measurements, which finds it more difficult to model it than the normal case. If ranked, this affects the estimation the least when compared to other noise variations.

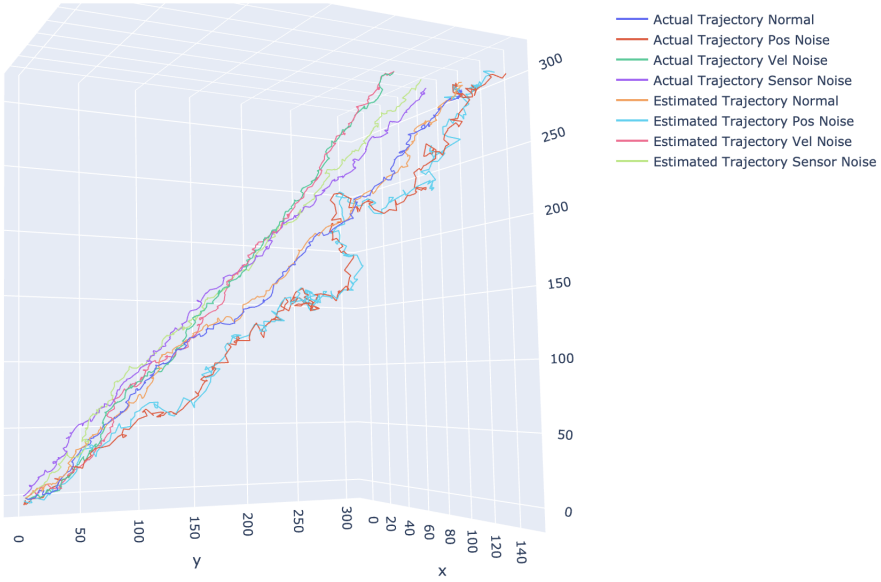
Velocity Update Noise (σ_{rx_i}) - Increasing this flavour of noise makes the latent state variables of velocity more erratic. This is not available to the estimator via the measurements, which is why it deviates significantly from the actual velocity (and hence the trajectory). If ranked, this affects the estimation the most when compared to other noise variations.

I have plotted 5 plots. 3 of them show the Normal Case vs Noise Enhanced, one for each variant. 1 Plot shows all three noise plots side by side, along with the normal one. One plot gives the uncertainty ellipses for all 4 trajectories.

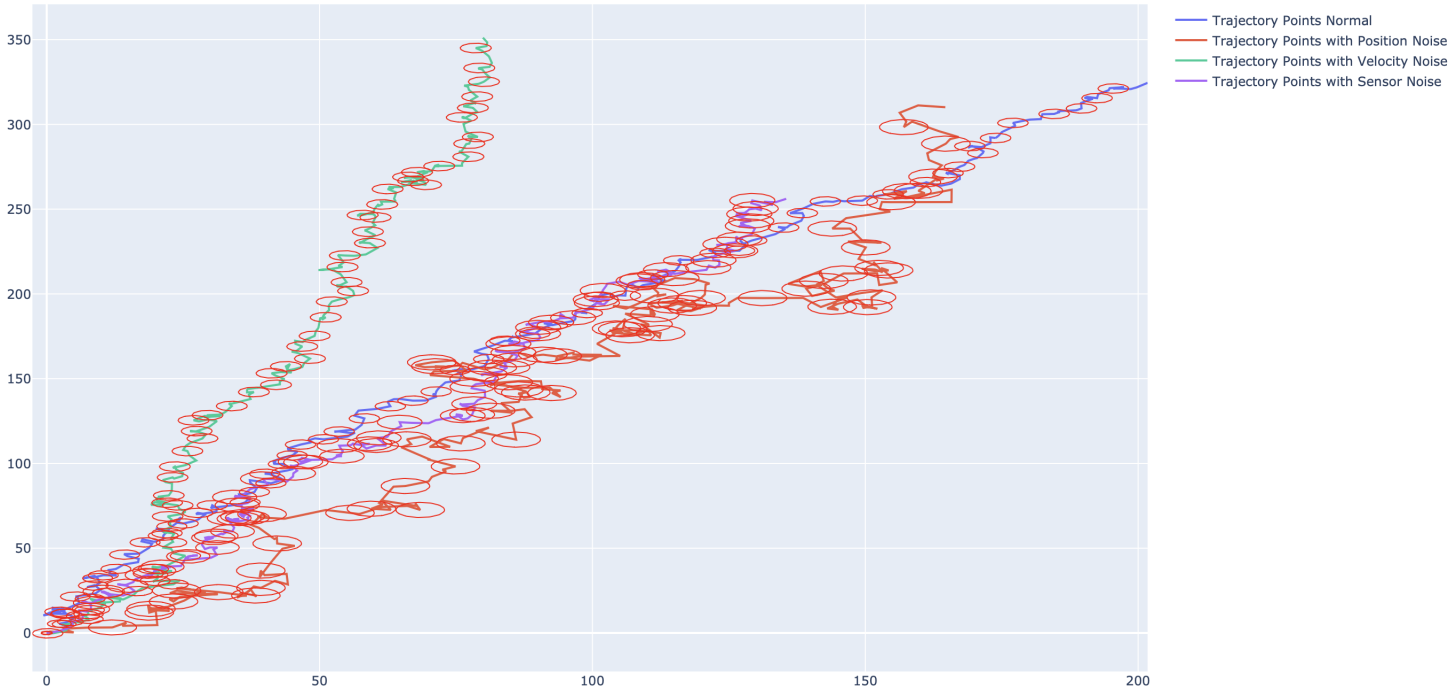
The Positional Noise Enhanced, Velocity Noise Enhanced and the Sensor Noise Enhanced Actual Paths, Estimations and Noisy Observations are captured below:



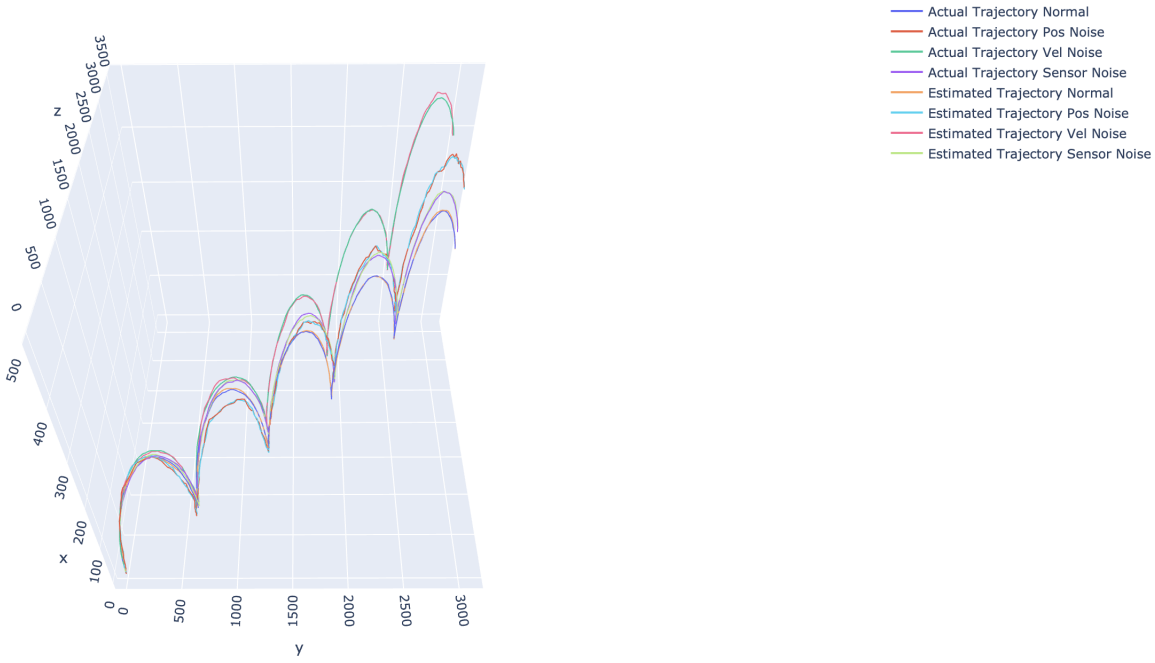
Plotting all of these together yields:



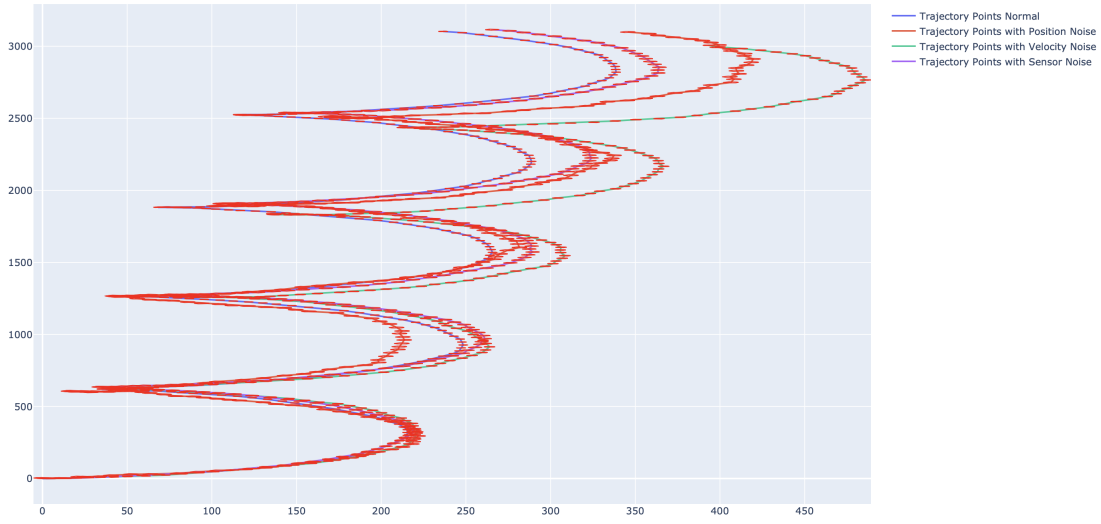
The Uncertainty Ellipses are Plotted. I have plotted these at selective points along the path to highlight the difference in sizes and for clarity. Note that the noisy ellipses are larger than the normal ones, accounting for the lower certainity because of noise:



Experimenting with the above variations on an alternate control input:



Which yields the uncertainty ellipses:

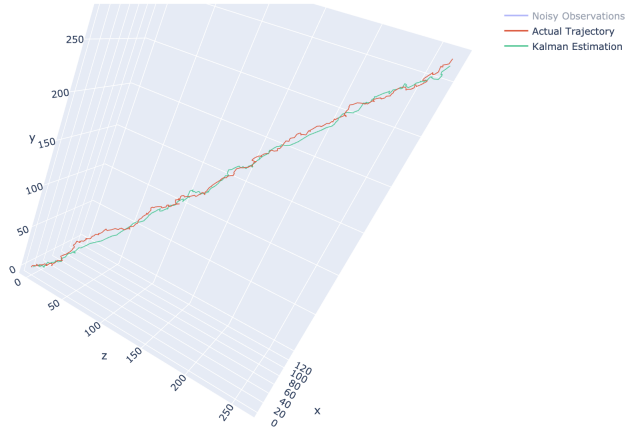


Part E

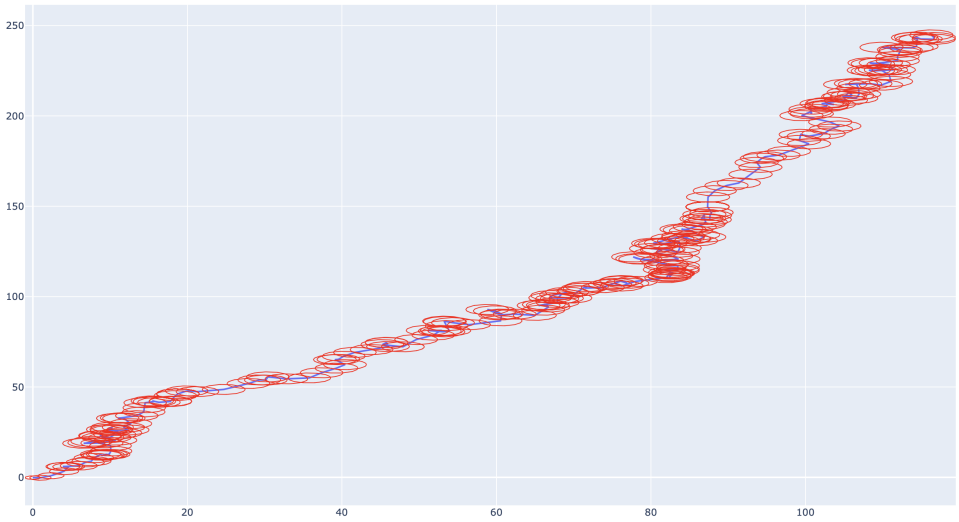
When the sensor measurements drop out for 30 seconds, the estimator can only perform the motion model update of evolving the state with time. Thus, the latest belief when the sensors turn off is propagated, and the actual trajectory diverges significantly from the estimated one.

When the estimator re-establishes contact with the sensors, it starts updating it's belief, and converges into the actual trajectory as more and more inputs flood in.

This Plot captures the estimates diverging from the actual ones near times 50 and 200 for a while, before converging back into the actual.



The uncertainty ellipses are plotted:



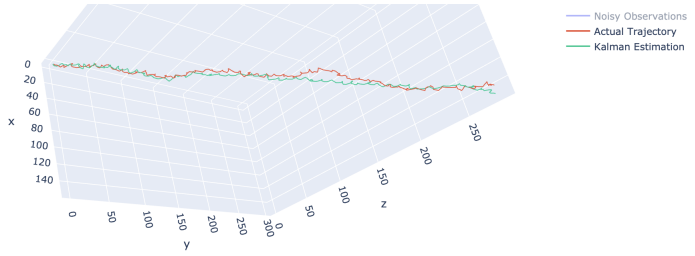
It can be observed that the uncertainty ellipses get slightly bigger, and follow a straighter line in periods where the sensor data is not available. This is because the control inputs are not available to the estimator and the previous beliefs are propagated.

Part F

Up until the point all the values are available to estimator, it models the actual trajectory fairly well.

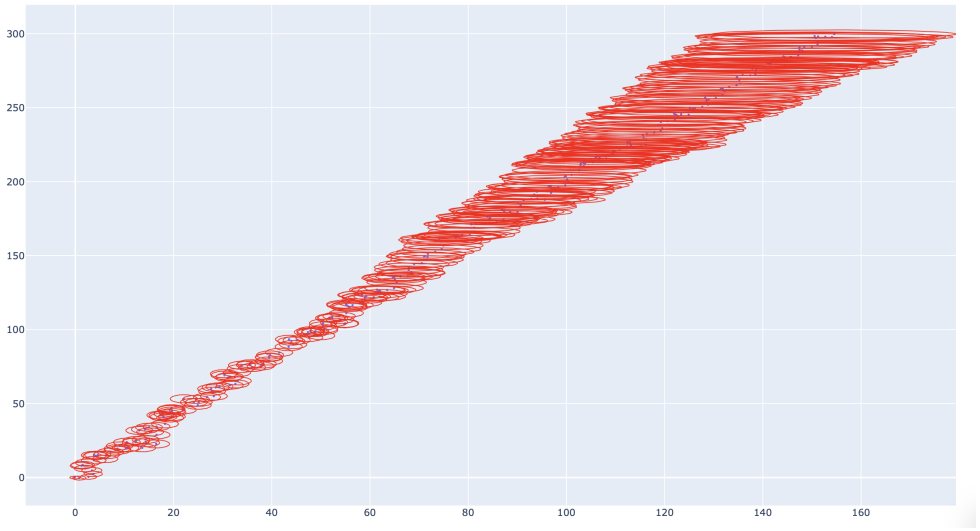
While the estimator loses the X sensor, it keeps receiving the Y and Z measurements. Thus, even the estimated trajectory is fairly accurate in it's y and z coordinates. However, it evolves the last measured update in X, which diverges with time since these is a velocity update happening in X as well.

We can actually see the estimate diverging form the actual somewhere in the middle of the



plot below! :

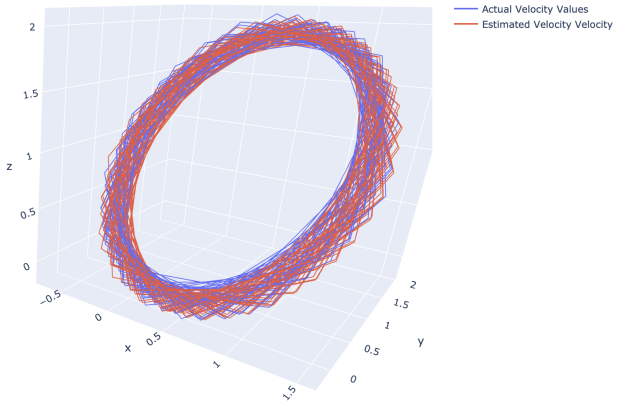
The uncertainty ellipses are plotted:



The X dimension of the uncertainty ellipses has stretched out by a lot, since we are no longer receiving any X inputs and there is no belief correction. The Y component of the ellipses is still small because the belief is corrected at each measurement.

Part G

Even though the estimator receives no explicit information about the velocity of the aeroplane, it can still track these values fairly accurately since it can use the difference between two simultaneous positional coordinates to model the velocity components. That is why the graphs for these are so strikingly similar!



Data Association

Part H

Figuring out which data point would belong to which trajectory requires a metric that evaluates it's association with a particular trajectory. Then, the values of this metric of points with trajectories can be ranked and an assignment of variables can be found.

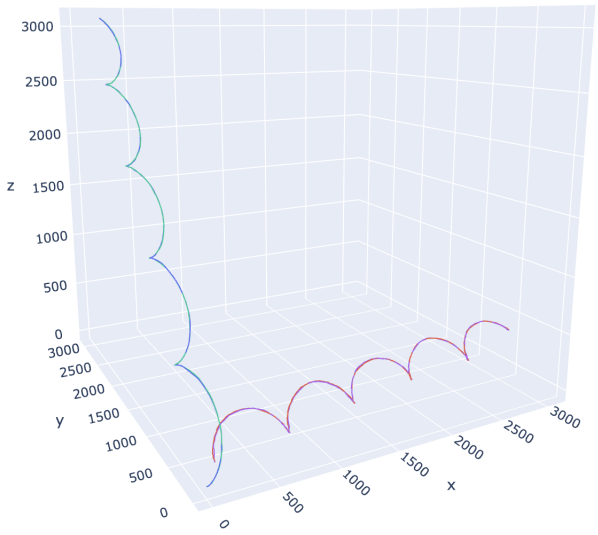
The first approach was to evolve the trajectory using the motion model, and use the mean to find the distance from the point. This could either be a euclidean or the manhattan distance. However, this approach would disregard the variance data, which is critical, the distance to be evaluated should factor in the spread of the trajectory as well.

The **Mahalanobis** distance is a metric that calculates the distance of a point from a distribution!

Given two airplanes and their (jumbled) sensor data, I find the 4 (2*2) distances, and use compare these values to figure out which point based approach to figure which point belongs to which trajectory.

If the points are closer to separate trajectories, I allocate them to the ones they are closer to. If they are closer to the same trajectory, I allocate the closer one to that trajectory and the other point to the other trajectory.

My code is also able to handle situations in which no observation or a single measurement is made available. In that case. It simply allocates the only measurement to the closer trajectory, and if no measurement is available, simply performs the state evolution.



Part I

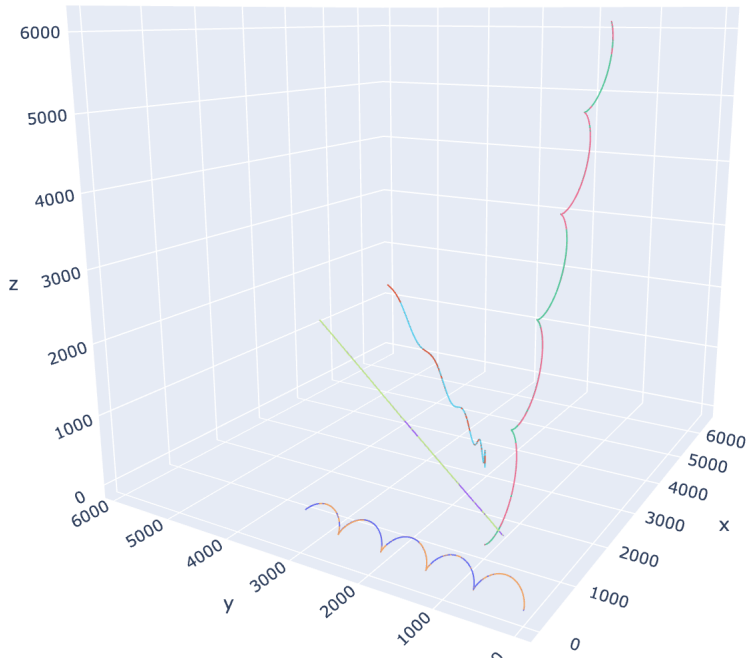
Generalising the approach from the previous part, I now had to select the correct mapping from trajectories to their measurements for a general n.

I thus needed a metric to **compare** two data assignments. Realising that making any assignment wrong would increase the **product** of the distances large, distance could be a good metric.

Thus, I go through all the permutations of assignments, find the product of distances, and declare the one with the minimum product to be the correct one.

The above approach would now work for higher values of n because n! scales very fast. In that case, I would randomly select a trajectory, find it's distances from all the data points available, **allocate** the point with the minimum distance to this trajectory, and **remove** (so that it isn't selected for some other trajectory as well) this point from the set of points. This step repeatedly would give a good approximation of the permutations approach. Since we are asked to simulate this for n = 4, the permutations are scalable (24) and it is thus a better bet. The working code for greedy approach has been commented out, and can be made to work by un-commenting it.

The following figure shows 4 close trajectories, and my algorithm is able to figure out which data point belong to which trajectory, producing estimates very close to their actual values!



Q2 - Landmark Localisation

Part A

Note that the action model remains similar to Q1, and it is the observation model that incurs a non-linearity here.

Thus, the Linear Kalman Model for Motion is

$$X_{t+1} = A_t X_t + B_t u_t + \epsilon_t$$

, where,

$$X_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \dot{x}_{t+1} \\ \dot{y}_{t+1} \end{bmatrix}, A_t = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, X_t = \begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix}$$

$$B_t = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, u_t = \begin{bmatrix} \delta_{\dot{x}_t} \\ \delta_{\dot{y}_t} \end{bmatrix} \text{ and } \epsilon_t = \mathcal{N}(0, R) \text{ with}$$

$$R = \begin{bmatrix} \sigma_{mm}^2 & 0 & 0 & 0 \\ 0 & \sigma_{mm}^2 & 0 & 0 \\ 0 & 0 & \sigma_{mm}^2 & 0 \\ 0 & 0 & 0 & \sigma_{mm}^2 \end{bmatrix} \text{ where } \sigma_{mm} = 0.01$$

The Observation model takes an interesting turn.

In case there are no landmarks accessible to the airplane, the updates are carried out linearly, just like Q1 above. However, the non-linearity is introduced in the case where the aeroplane finds itself in the range of one of the landmarks. I describe the update equations for one of the landmarks (150, 0), and the others are very similar.

$$z_t = h(X_t) + \delta_t$$

where

$$z_t = \begin{bmatrix} x'_t \\ y'_t \\ r_{l_1 t} \end{bmatrix}, X_t = \begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix}, h(X_t) = \begin{bmatrix} x_t \\ y_t \\ ((x_t - 150)^2 + (y_t)^2)^{1/2} \end{bmatrix}, \delta_t = \mathcal{N}(0, Q) \text{ with}$$

$$Q = \begin{bmatrix} \sigma_{gps}^2 & 0 & 0 & 0 \\ 0 & \sigma_{gps}^2 & 0 & 0 \\ 0 & 0 & \sigma_{gps}^2 & 0 \\ 0 & 0 & 0 & \sigma_{lm}^2 \end{bmatrix} \text{ where } \sigma_{gps} = 10 \text{ and } \sigma_{lm} = 1$$

Part B

When both the updates are linear (when the airplane does not lie in any landmark-spanned region), both the evolve and correct steps of the KF are linear and hance are evaluated exactly like in Q1. In the landmark presence scenario, the measurement update is carried out after a linearisation step,

$$h(x_t) \approx h(\bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t)}{\partial x_t}(x_t - \bar{\mu}_t)$$

$$h(x_t) \approx h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t)$$

Jacobian matrices

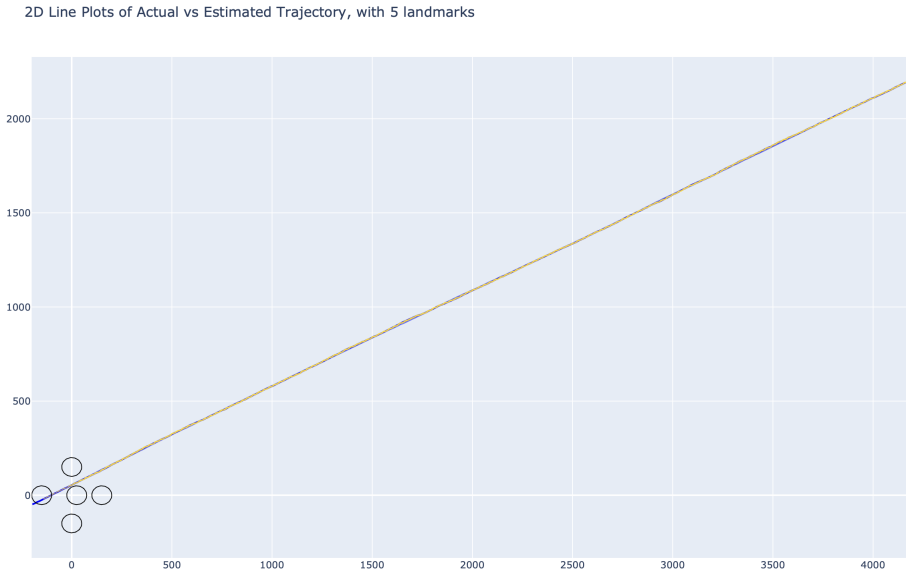
$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + R_t)^{-1} \quad \longleftarrow \quad K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + R_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t)) \quad \longleftarrow \quad \mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t)$$
$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t \quad \longleftarrow \quad \Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$$

For the considered Landmark at (150, 0), the Jacobian matrix H_t, is evaluated to

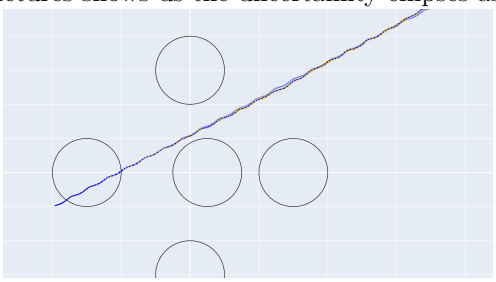
$$H_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{x_t - 150}{((x_t-150)^2 + y_t^2)^{0.5}} & \frac{y_t}{((x_t-150)^2 + y_t^2)^{0.5}} & 0 & 0 \end{bmatrix}$$

Part C

Based on the Initial Conditions and the control variables, the following trajectory was realised

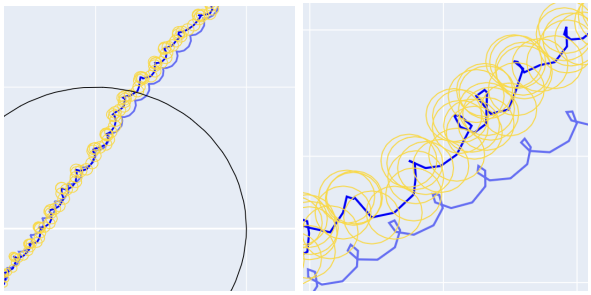


A more zoomed in pictures shows us the uncertainity ellipses as well.



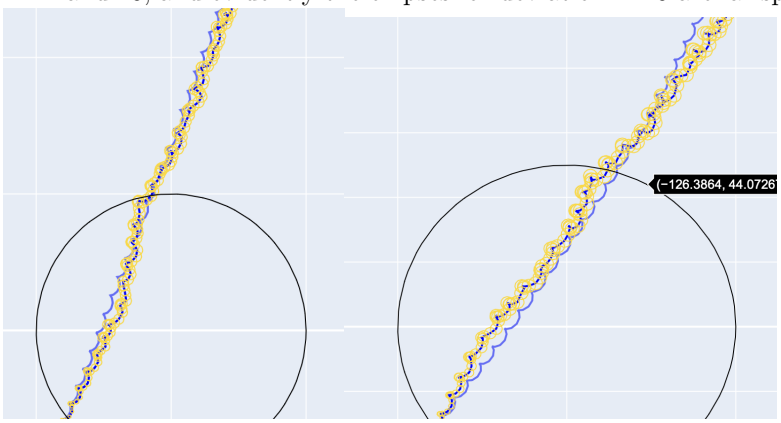
Near the boundary in the landmark range, we can see how the actual and estimated trajectories are very similar, but as we move outside the range, they diverge. This is because measurements are now less and less certain.

Also depicted is a later timestamp, outside the landmark ranges, where the paths have now diverged significantly.



Part D

Increasing Noise from Landmark Observation is expected to make the estimates less certain, which is exactly what is observed. The Screenshots below depict uncertainity ellipses for deviation = 1 and 20, and evidently the ellipses for deviation = 20 are far spread out ().

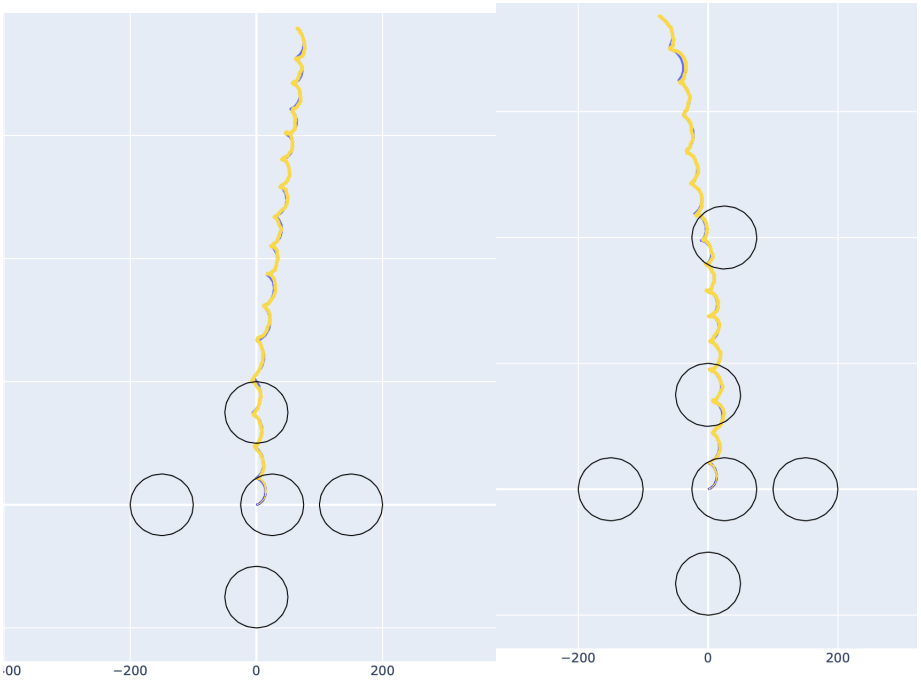


Further we can se that the left image tries to keep the actual and estimated trajectories together, while the Right Hand side does a poorer job at it.

Part E

I added an additional Landmark at (25, 400). I experimented with the controls until the actual trajectory passed through some of these landmarks.

I had to edit some motion updates to make this more coherent. The comparison with and without the more additional landmark creates a trajectory:



We can see that the estimation is pretty accurate in the landmark region, and that this is expected to make the estimated trajectory more accurate, though it is difficult to directly observe it from the figure.