COL778

ASSIGNMENT 1

State Estimation

Shubh Goel *(2020EE10672)*
Link to Plots

Date of submission of Report: February 5, 2024

Plots

Click here to open drive folder containing plots.

1 State Estimation using Kalman Filters

(a)

Let the state of the plane at time t be defined as,

$$X_t = \begin{bmatrix} x_t \\ y_t \\ z_t \\ \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \end{bmatrix}$$

where $[x_t, y_t, z_t]$ represents the true position of the plane and $[\dot{x}_t, \dot{y}_t, \dot{z}_t]$ represents the true velocity of the plane at time t.
The motion equations of the plane is defined as follows:

$$x_{t+1} = x_t + \dot{x}_t \Delta t$$

$$y_{t+1} = y_t + \dot{y}_t \Delta t$$

$$z_{t+1} = z_t + \dot{z}_t \Delta t$$

$$\dot{x}_{t+1} = \dot{x}_t + \delta \dot{x}_t$$

$$\dot{y}_{t+1} = \dot{y}_t + \delta \dot{y}_t$$

$$\dot{x}_{t+1} = \dot{x}_t + \delta \dot{x}_t$$

Hence, the **motion model** of the plane is defined as follows:

$$X_{t+1} = AX_t + Bu_t + \mathcal{N}(0, Q)$$

where,

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad u_t = \begin{bmatrix} \delta \dot{x}_t \\ \delta \dot{y}_t \\ \delta \dot{z}_t \end{bmatrix}$$

Let $Z_t = [x'_t, y'_t, z'_t]$ be the noisy observations of X_t. Then the **observation model** of the plane is defined as:

$$Z_t = CX_t + \mathcal{N}(0, R)$$

where,

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Simulation:

- $T = 500$, $\Delta t = 1$

- $X_0 = [0,0,0,0,0,0]^T$
- $u_t = [\sin t, \cos t, \sin t]^T$
- $R = diag(\sigma_s^2, \sigma_s^2, \sigma_s^2)$ with $\sigma_s = 7$
- $Q = diag(\sigma_{rx}, \sigma_{ry}, \sigma_{rz}, \sigma_{r\dot{x}}, \sigma_{r\dot{y}}, \sigma_{r\dot{z}},)$ with $\sigma_{rx}, \sigma_{ry}, \sigma_{rz} = 1.2$ and $\sigma_{r\dot{x}}, \sigma_{r\dot{y}}, \sigma_{r\dot{z}} = 0.01$

```python
#1 a)
T = 500
sig_s = 7
sig_rx = 1.2
sig_ry = 1.2
sig_rz = 1.2
sig_rx_ = 0.01
sig_ry_ = 0.01
sig_rz_ = 0.01

t_values = np.arange(T)
u = np.vstack([np.sin(t_values), np.cos(t_values), np.sin(t_values)]).T

plane = Plane(sig_s,sig_rx,sig_ry,sig_rz,sig_rx_,sig_ry_,sig_rz_,u,init_state = np.array([0,0,0,0,0,0]), T = T)

for t in range(T-1):
    plane.simulate(t+1)
    plane.observe(t+1)
```

Figure 1: Simulations Code

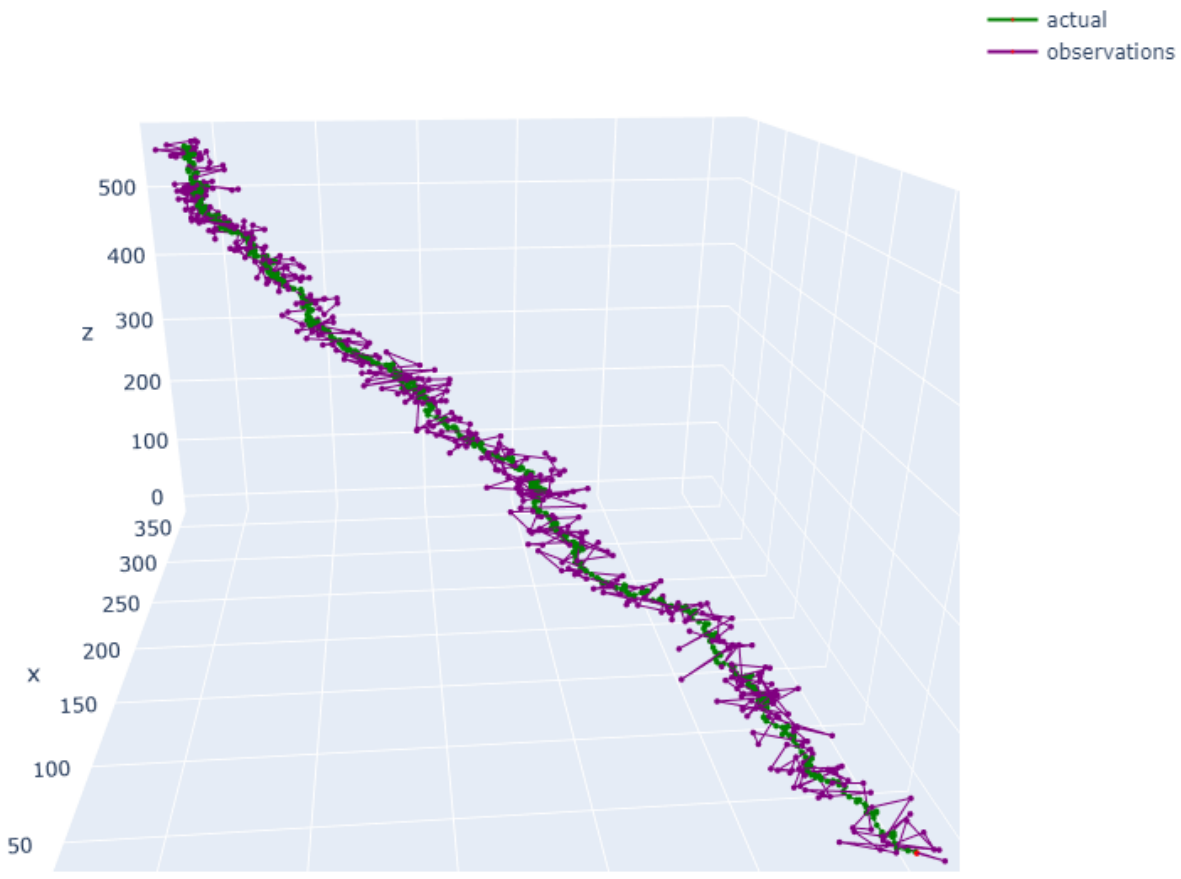


Figure 2: Plots of actual and observed trajectory

(b)

Let belief of the estimate at time t be,

$$X_{t|0:t} \sim \mathcal{N}(\mu_{t|0:t}, \Sigma_{t:0:t})$$

- **Action Update:** Using the motion model defined in $1(a)$ and marginalizing the joint distribution $(X_{t|0:t}, X_{t+1|0:t+1})$ we get,

$$X_{t+1|0:t+1} \sim \mathcal{N}(A\mu_{t|0:t} + Bu_t, A\Sigma_{t|0:t}A^T + Q)$$

- **Measurement Update:** Using the observation model defined in $1(a)$ and by conditioning the joint distrubtion $(X_{t+1|0:t}, Z_{t+1|0:t+1})$ on $Z_{t+1} = z_{t+1}$ we get,

$$X_{t+1|0:t+1} \sim \mathcal{N}(\mu_{t+1|0:t+1} + K(z_{t+1} - C\mu_{t+1|0:t+1}), (I - KC)\Sigma_{t+1|0:t+1})$$

where,

$$K = \Sigma_{t+1|0:t+1}C^T(C\Sigma_{t+1|0:t+1}C^T + R)^{-1}$$

```python
def kalman_filter(self,curr_mu, curr_cov, action, observation, flag = 'estimate'):
    if flag == 'predict':
        mu = np.dot(self.A,curr_mu) + np.dot(self.B,action)
        cov = np.dot(np.dot(self.A,curr_cov),self.A.T) + self.Q
        return mu, cov
    if flag == 'correct':
        K = np.dot(np.dot(curr_cov,self.C.T),np.linalg.inv(np.dot(np.dot(self.C,curr_cov),self.C.T) + self.R))
        mu = curr_mu + np.dot(K, (observation - np.dot(self.C,curr_mu)))
        cov = np.dot((np.eye(curr_cov.shape[0]) - np.dot(K,self.C)),curr_cov)
        return mu,cov
    mu_action = np.dot(self.A,curr_mu) + np.dot(self.B,action)
    cov_action = np.dot(np.dot(self.A,curr_cov),self.A.T) + self.Q
    K = np.dot(np.dot(cov_action,self.C.T),np.linalg.inv(np.dot(np.dot(self.C,cov_action),self.C.T) + self.R))
    mu = mu_action + np.dot(K, (observation - np.dot(self.C,mu_action)))
    cov = np.dot((np.eye(curr_cov.shape[0]) - np.dot(K,self.C)),cov_action)
```

Figure 3: Kalman Filter Implementation

(c)

Prior belief over the vehicle's initial state with a standard deviation of 0.01 for each state variable.

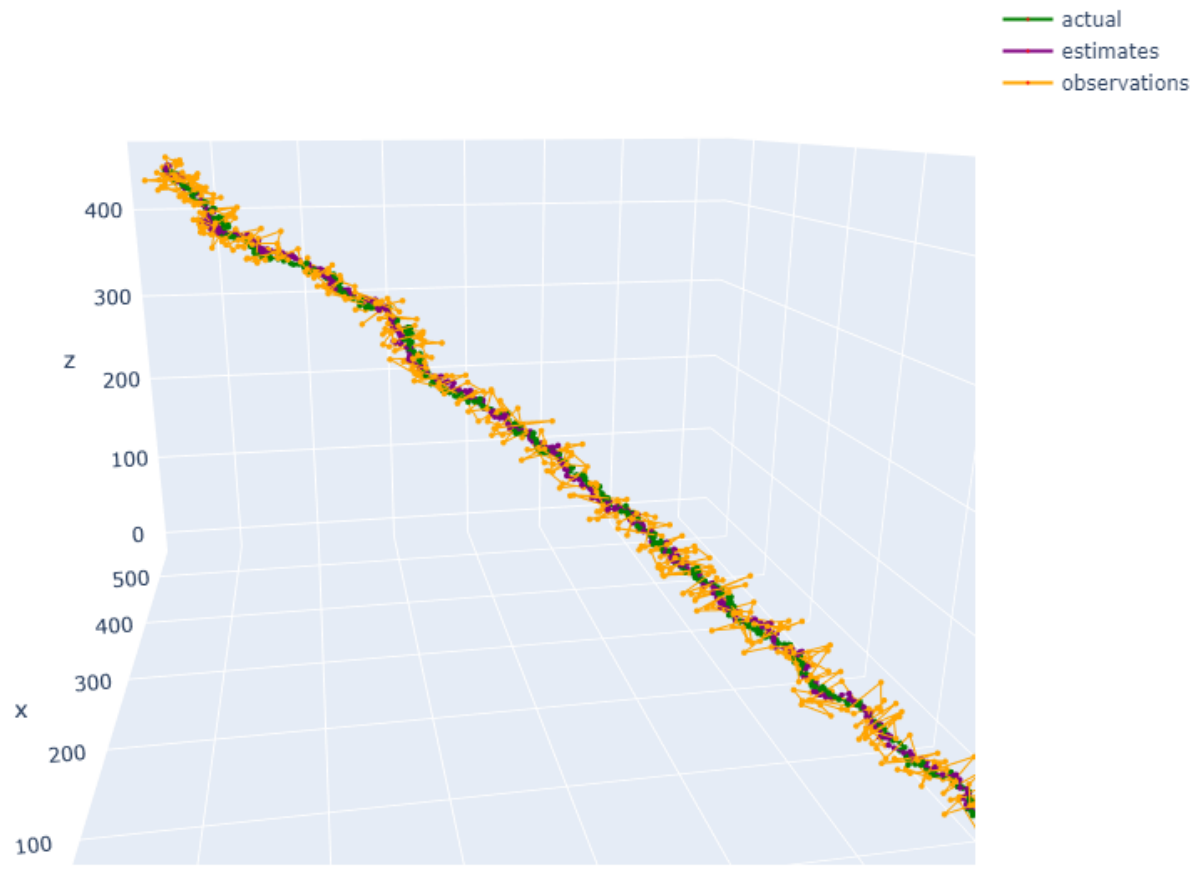


Figure 4: Plots of actual, observed and estimated trajectory

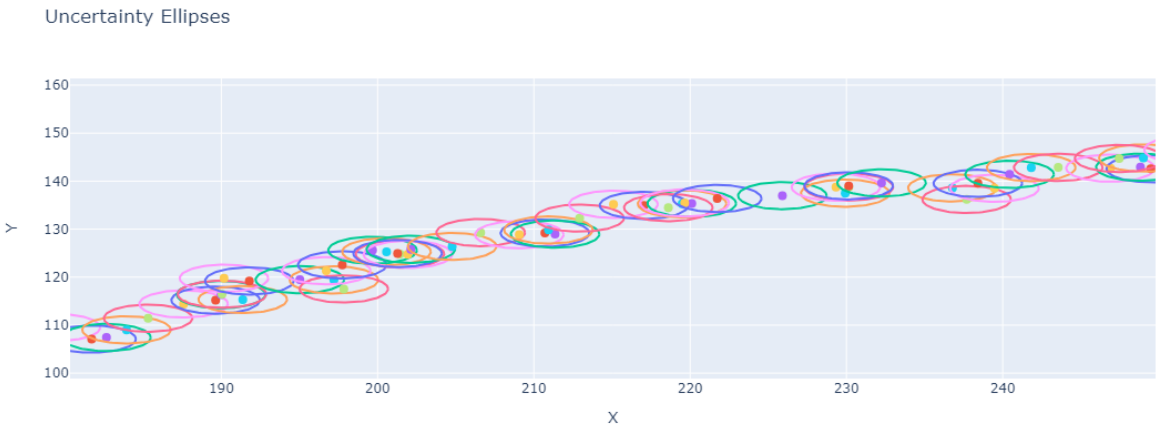


Figure 5: Uncertainty ellipses for the projection of the estimated trajectory on the XY plane

(d)

Kalman Filter equations:

$$\bar{\mu}_t = A\mu_{t-1} + Bu_t \tag{1}$$

$$\bar{\Sigma}_t = A\Sigma_{t-1}A^T + Q_t \tag{2}$$

$$K_t = \bar{\Sigma}_t C^T (C\bar{\Sigma}_t C^T + R_t)^{-1} \tag{3}$$

$$\mu_t = \bar{\mu}_t + K_t(z_t - C\bar{\mu}_t) \tag{4}$$

$$\Sigma_t = (I - K_t C)\bar{\Sigma}_t \tag{5}$$

- Increasing the noise in sensor measurements($\sigma_s = 11$) doesn't affect the actual trajectory of the plane. However, it increases the noise in the observations which in turn increases the uncertainty in the estimated trajectory(Fig.7a,Eq.3,Eq.5). The increased uncertainty in the estimated trajectory could also be inferred from the increased area of the uncertainty ellipses(7b).
- Increasing the noise in position updates($\sigma_{rx}(= \sigma_{ry} = \sigma_{rz}) = 2$) changes the actual trajectory of the plane(makes it a little noisier). This in turn increases the uncertainty in the estimated trajectory(Fig.8a) when the motion update is performed in the Kalman Filter(Eq.2,Eq.5). The increased uncertainty in the estimated trajectory could also be inferred from the increased area of the uncertainty ellipses(Fig.8b).
- Increasing the noise in velocity updates($\sigma_{r\dot{x}}(= \sigma_{r\dot{y}} = \sigma_{r\dot{z}}) = 0.1$) changes the actual trajectory of the plane (Fig.9a). This happens because the velocity at each time step becomes significantly different from the previous conditions. However, this noise doesn't get propagated to the estimated trajectory because the noise in the position updates remains unchanged and the observation model returns the measurements of positions only. Hence, the uncertainty remains unchanged which is evident from the uncertainty ellipses plot (Fig.9b).

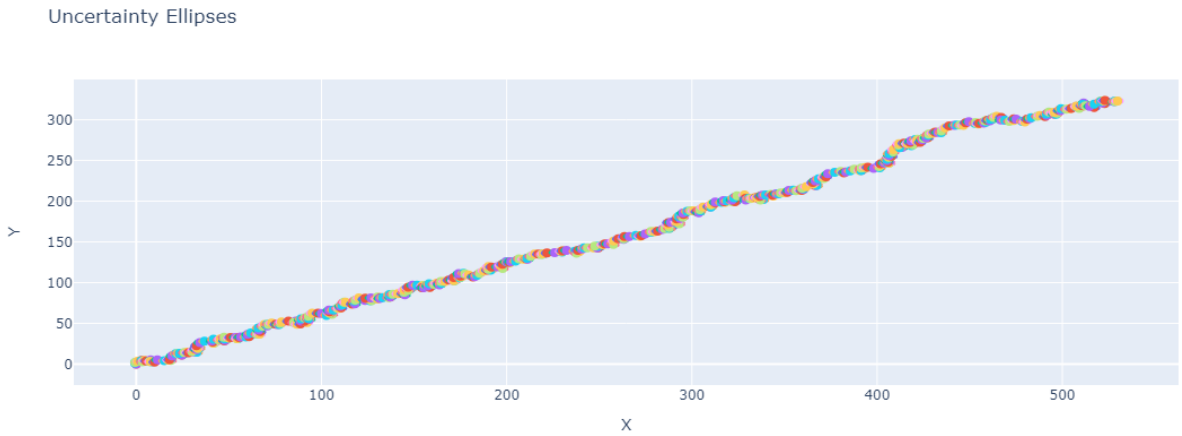
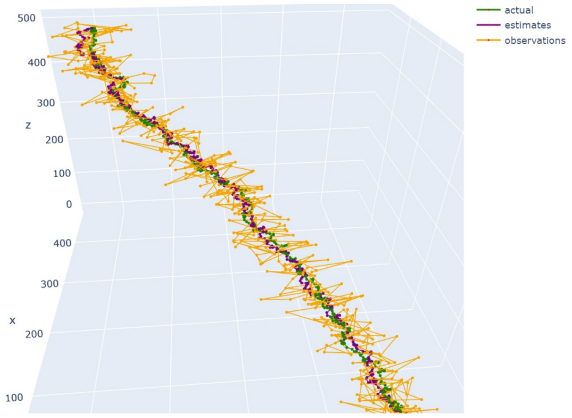
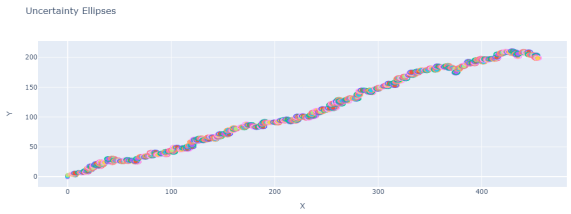


Figure 6: Uncertainty ellipse for $\sigma_s = 7, \sigma_{rx}(= \sigma_{ry} = \sigma_{rz}) = 1.2$ and $\sigma_{r\dot{x}}(= \sigma_{r\dot{y}} = \sigma_{r\dot{z}}) = 0.01$ (Zoomed out version of Fig. 5)

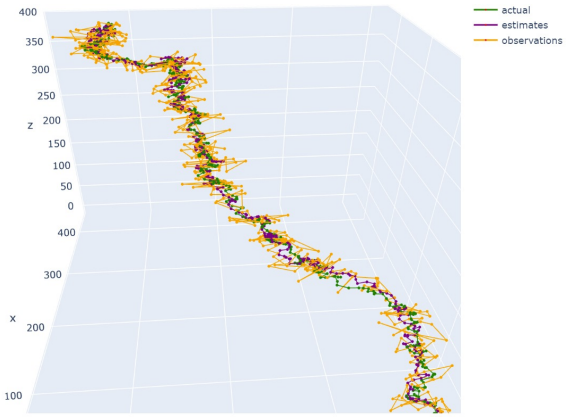


(a) Actual, Observerd and Estimated trajectory

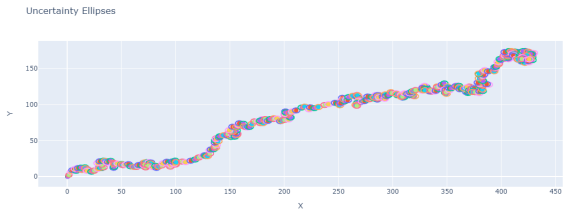


(b) Uncertainty Ellipses

Figure 7: $\sigma_s = 11, \sigma_{rx}(= \sigma_{ry} = \sigma_{rz}) = 1.2$ and $\sigma_{r\dot{x}}(= \sigma_{r\dot{y}} = \sigma_{r\dot{z}}) = 0.01$

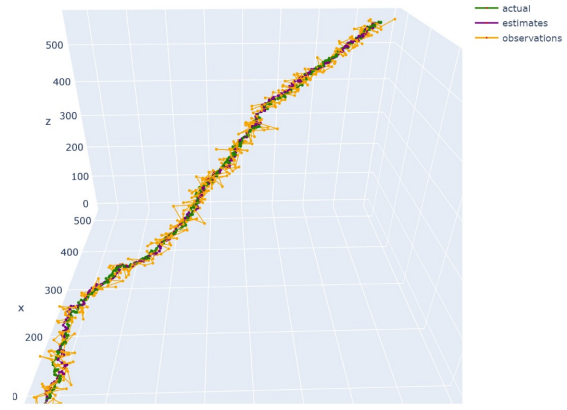


(a) Actual, Observerd and Estimated trajectory

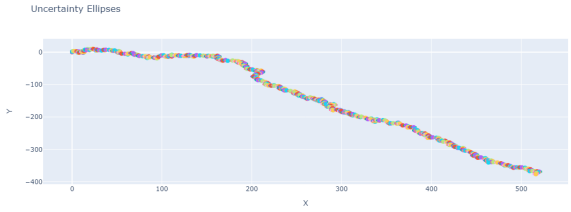


(b) Uncertainty Ellipses

Figure 8: $\sigma_s = 7, \sigma_{rx}(= \sigma_{ry} = \sigma_{rz}) = 2$ and $\sigma_{r\dot{x}}(= \sigma_{r\dot{y}} = \sigma_{r\dot{z}}) = 0.01$



(a) Actual, Observerd and Estimated trajectory



(b) Uncertainty Ellipses

Figure 9: $\sigma_s = 7, \sigma_{rx}(= \sigma_{ry} = \sigma_{rz}) = 1.2$ and $\sigma_{r\dot{x}}(= \sigma_{r\dot{y}} = \sigma_{r\dot{z}}) = 0.1$

(e)

We observe periodic deviation of estimated trajectory from the actual trajectory in the Fig. 10. This happens because during the intervals of radio silence, we propagate the current belief

using only the motion model(Eq.1, Eq.2). However, as soon as we get the measurements in the subsequent intervals, we use the observation model(Eq.4, Eq.5) to "correct" the belief, therefore decreasing the uncertainty. The periodic increase and decrease in uncertainty is also visible in the uncertainty ellipse plot(Fig. 11).

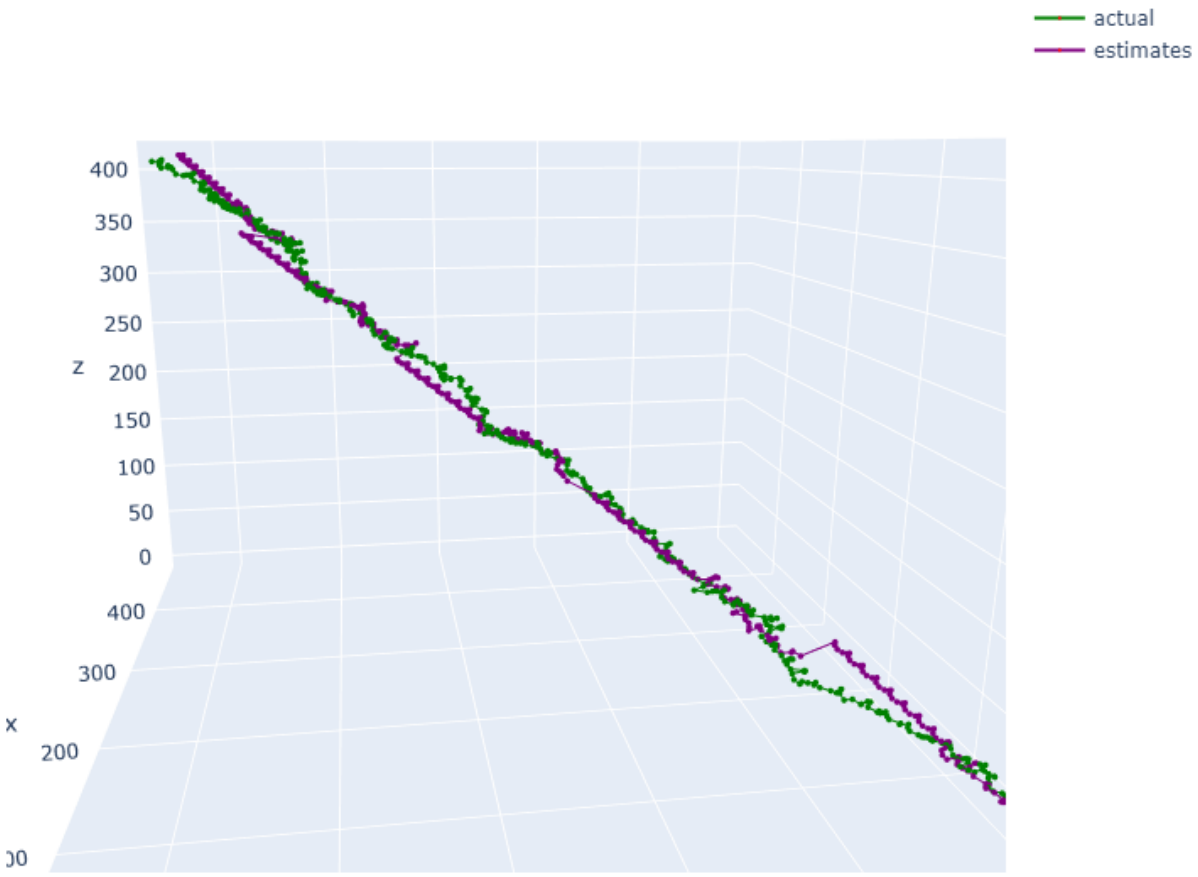


Figure 10: Plots of actual and estimated trajectory

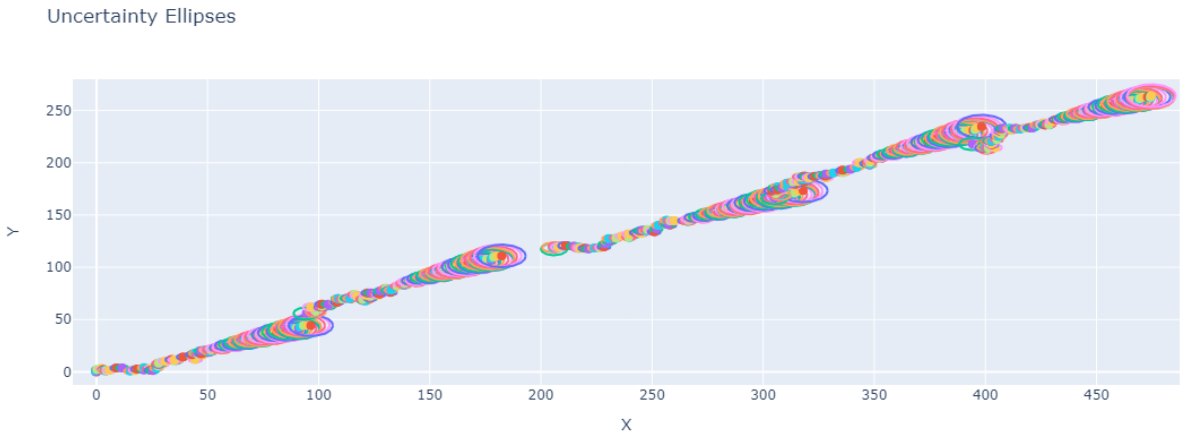


Figure 11: Plot of uncertainty ellipses of estimated trajectory

(f)

From Fig.12 we can see that the estimator cannot track the true values of velocity properly. This is because the belief over velocity estimates is propagated only through the action model since the sensors on the plane captures only its positions. Hence the observation model does not 'correct' the belief over velocity estimates at each time step.

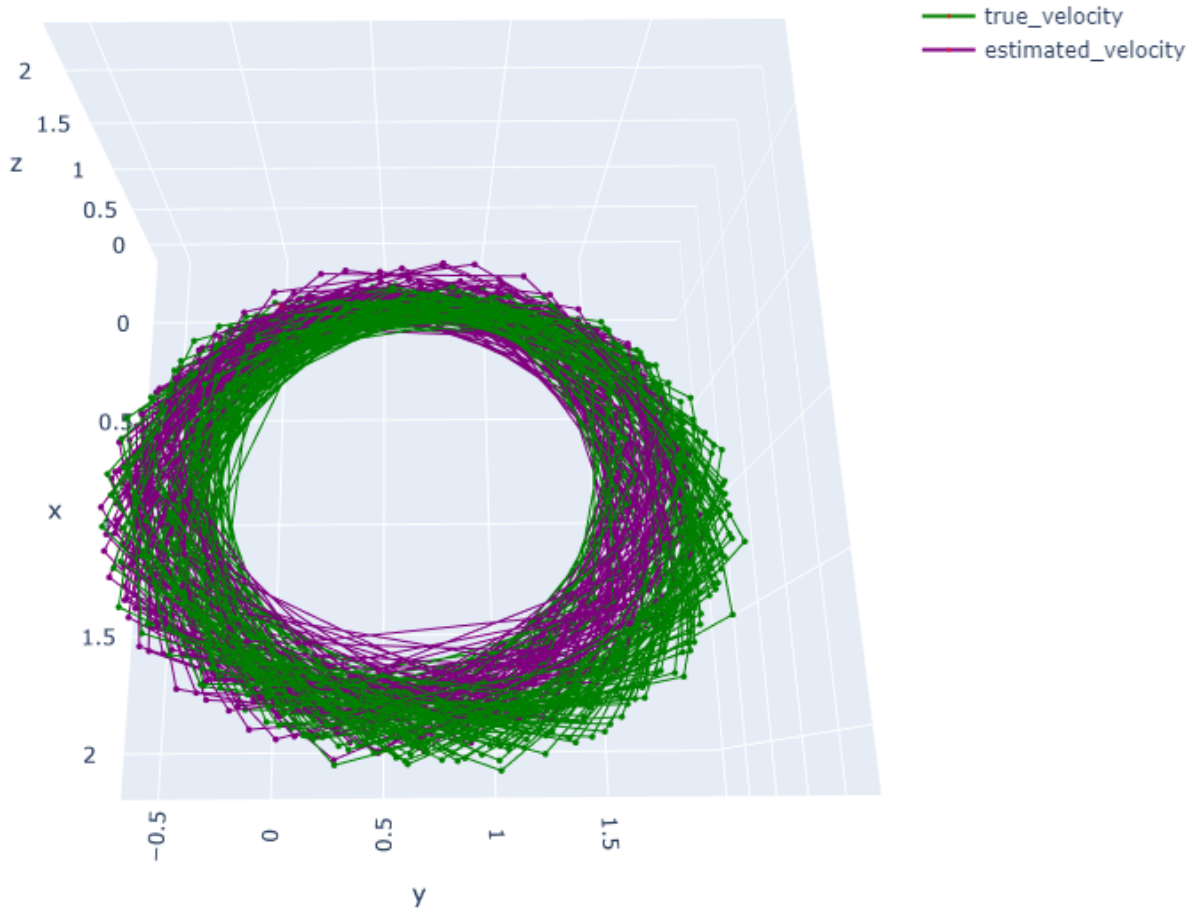


Figure 12: Plot of actual and estimated velocities

Data Association

Let $\{(\bar{\mu}_i, \bar{\Sigma}_i)\}_{i=1}^4$ be the believes of the planes after applying action update i.e.,

$$p(x; \bar{\mu}_i, \bar{\Sigma}_i) = \frac{1}{(2\pi)^{n/2}|\bar{\Sigma}_i|^{1/2}} exp\left(-\frac{1}{2}(x-\bar{\mu}_i)^T\bar{\Sigma}_i^{-1}(x-\bar{\mu}_i)\right) \tag{6}$$

Let $\{z_i\}_{i=1}^4$ be the set of observations received. Let the suitability of an observation z, to belong to a particular trajectory $(\bar{\mu}_i, \bar{\Sigma}_i)$ be defined as the likelihood of z being generated by the distribution $\mathcal{N}(\bar{\mu}_i, \bar{\Sigma}_i)$. Thus, for a particular plane, the observation that maximizes the Eq.6 or minimizes $\sqrt{(x-\bar{\mu}_i)^T\bar{\Sigma}_i^{-1}(x-\bar{\mu}_i)}$ would be the most suitable observation for it. The latter is also called the **Mahalanobis Distance**. More formally, let the distance of an observation z from a trajectory $(\bar{\mu}_i, \bar{\Sigma}_i)$ be defined as follows:

$$\mathcal{M}(z; \bar{\mu}_i, \bar{\Sigma}_i) = \sqrt{(z-\bar{\mu}_i)^T\bar{\Sigma}_i^{-1}(z-\bar{\mu}_i)} \tag{7}$$

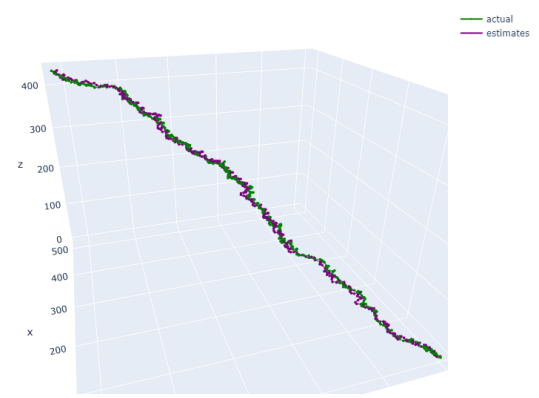
Let σ be an association of observations w.r.t the estimated trajectories. Then the goodness of this association $\mathcal{F}(\sigma)$ is defined as:

$$\mathcal{F}(\sigma) = \Sigma_{i=1}^{4} \mathcal{M}(\sigma(i); \bar{\mu}_i, \bar{\Sigma}_i)^2 \tag{8}$$

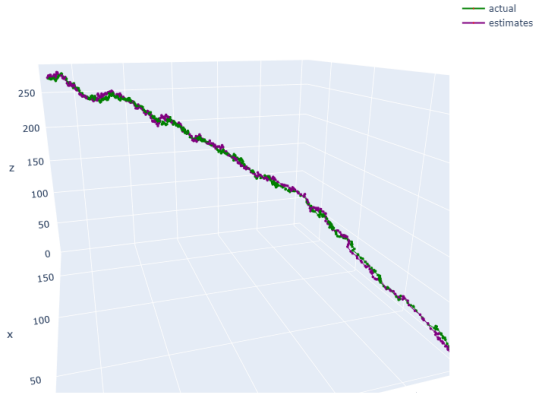
(g) Brute Force

```
cnt = 0 #For checking the accuracy of the strategy
for t in range(T-1):
    observations = []
    predictions = []
    pred_cov = []
    cnt+=1
    for i in range(4):
        planes[i].simulate(t+1)
        observations.append((planes[i].observe(t+1),i))
        predictions.append(planes[i].estimate(t+1,'predict')[:3])
        pred_cov.append(planes[i].curr_cov[:3,:3])
    np.random.shuffle(observations) #Randomize the obtained observations
    perms = get_permutations(observations) #get all the permutations
    best_score = 1e16
    best_perm = []
    for i in range(len(perms)):
        curr_score = metric(perms[i],predictions,pred_cov)
        if(curr_score < best_score):
            best_score = curr_score
            best_perm = perms[i]

    flag = True
    for i,plane in enumerate(planes):
        plane.estimate(t+1, 'correct', best_perm[i][0]) #Correct the believes using the best permutations
        if(i != best_perm[i][1] and flag):
            cnt-=1
            flag = False
print(cnt)
```

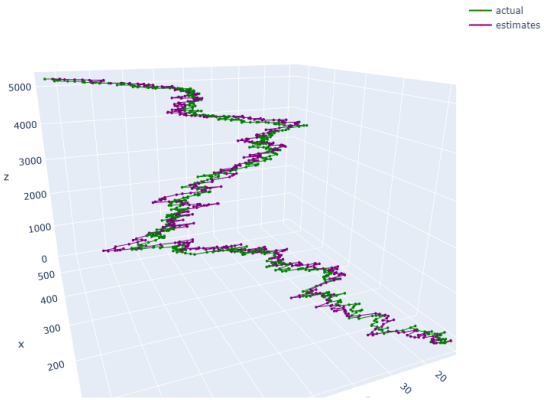
Figure 13: Simulation of brute force strategy



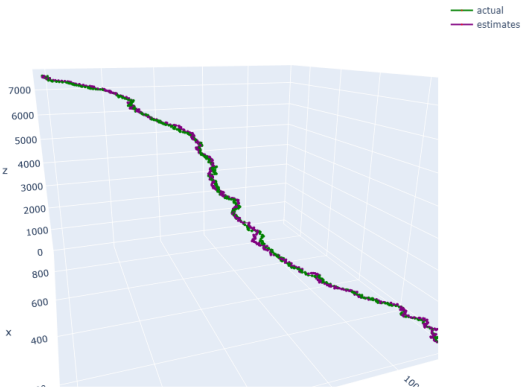
(a) $\sigma_s = 7, \sigma_{rx} = 1.2, \sigma_{r\dot{x}} = 0.01$, initial belief std = 0.01 and initial state = $[0, 0, 0, 0, 0, 0]$



(b) $\sigma_s = 8, \sigma_{rx} = 1.4, \sigma_{r\dot{x}} = 0.03$, initial belief std = 0.03 and initial state = $[5, 5, 5, 0, 5, 0]$



(c) $\sigma_s = 9, \sigma_{rx} = 1.6, \sigma_{r\dot{x}} = 0.05$, initial belief std = 0.05 and initial state = $[10, 10, 10, 0, 0, 10]$



(d) $\sigma_s = 10, \sigma_{rx} = 2, \sigma_{r\dot{x}} = 0.07$, initial belief std = 0.07 and initial state = $[15, 15, 15, 0, 0, 15]$

Figure 14: Simulation of strategy with different parameter settings

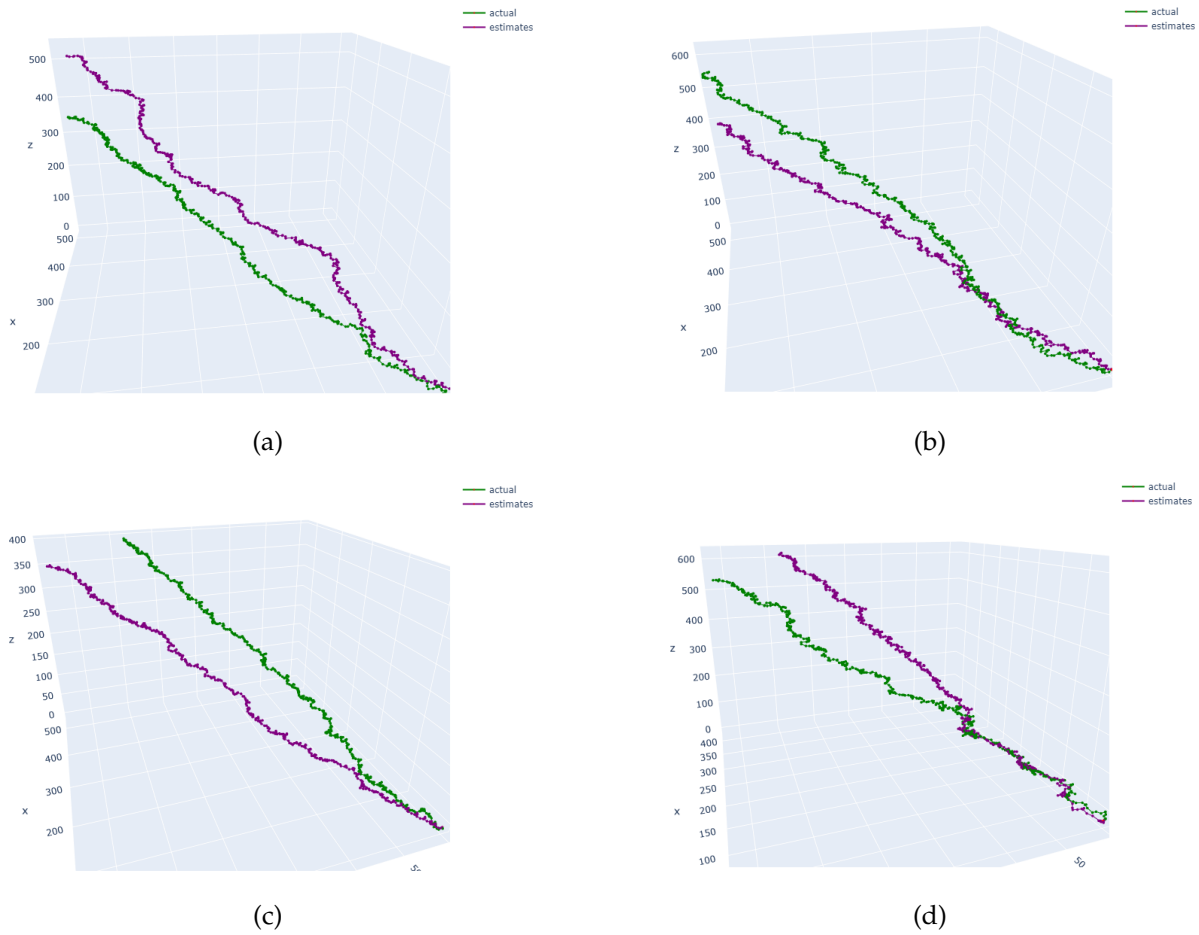


Figure 15: Simulation of strategy with same parameter settings

We observe that the brute force strategy works well when the planes have different parameter settings (Fig.14). However, when the planes have identical parameters, the estimated trajectory deviates from the actual trajectory(Fig.15).

(h) Hungarian Algorithm

The cost matrix $\mathcal{C}(i,j)$ is defined as follows:

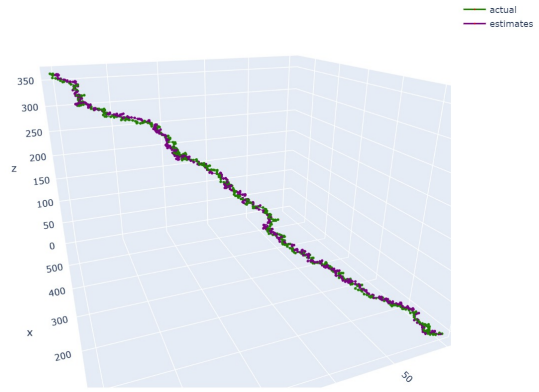
$$\mathcal{C}(i,j) = \mathcal{M}(z_j; \bar{\mu}_i, \bar{\Sigma}_i)^2 \tag{9}$$

```
def min_covered(cost_matrix):
    n = cost_matrix.shape[0]
    k = 2*n
    mask = (2**n) - 1
    rows = np.zeros(n)
    cols = np.zeros(n)
    covered_rows = 0
    covered_cols = 0
    num_lines = 2*n
    for p in range(1,2**k):
        r = p&mask
        c = (p&(mask<<4))>>4
        zeros_covered = True
        for i in range(n):
            for j in range(n):
```

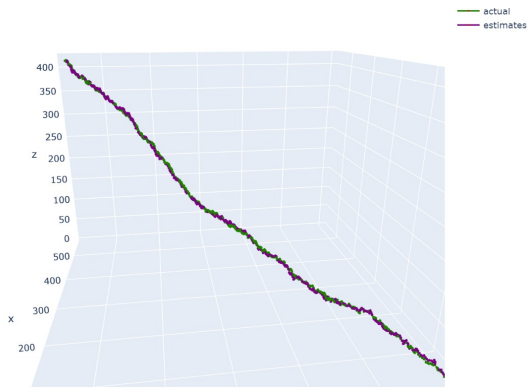
```
                if(cost_matrix[i,j] == 0):
                    if((not((r&(1<<i))>>i)) and (not((c&(1<<j))>>j))):
                        zeros_covered = False
                        break
            if(not zeros_covered):
                break
        if(zeros_covered):
            tr = bin(r)
            tc = bin(c)
            curr_lines = tr.count('1') + tc.count('1')
            if(curr_lines < num_lines):
                covered_rows = r
                covered_cols = c
                num_lines = curr_lines
    for i in range(n):
        if(covered_rows&(1<<i)):
            rows[i] = 1
        if(covered_cols&(1<<i)):
            cols[i] = 1

    return rows,cols

def hungarian_algorithm(cost_matrix):
    n = cost_matrix.shape[0]
    cost_matrix = cost_matrix - np.min(cost_matrix, axis=1, keepdims=
        True) #Step 1
    cost_matrix = cost_matrix - np.min(cost_matrix, axis=0, keepdims=
        True) #Step 2
    rows,cols = min_covered(cost_matrix) #Step 3
    num_covered = np.sum(rows) + np.sum(cols)
    while int(num_covered) < n: #Step 4
        min_num = 1e16
        for i in range(n):
            for j in range(n):
                if ((not rows[i]) and (not cols[j])):
                    min_num = min(min_num, cost_matrix[i,j])
        for i in range(n):
            if(int(rows[i]) == 1):
                continue
            for j in range(n):
                cost_matrix[i,j] -= min_num
        for j in range(n):
            if(int(cols[j]) == 0):
                continue
            for i in range(n):
                cost_matrix[i,j] += min_num
        rows,cols = min_covered(cost_matrix)
        num_covered = np.sum(rows) + np.sum(cols)
    assignment = np.zeros(n)
    for k in range(n):
        idx = 0
        for i in range(n):
            cnt = 0
            for j in range(n):
                cnt+=(cost_matrix[i,j] == 0)
            if(cnt == 1):
                idx = i
                break
```

```
        idx2 = 0
        for j in range(n):
            if(cost_matrix[idx,j] == 0):
                idx2 = j
                break
        assignment[idx] = idx2
        for i in range(n):
            cost_matrix[i,idx2]+=1
    return assignment
```

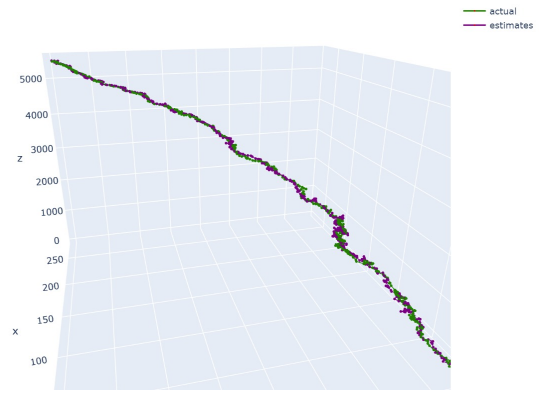
Listing 1: Hungarian algorithm Implementation



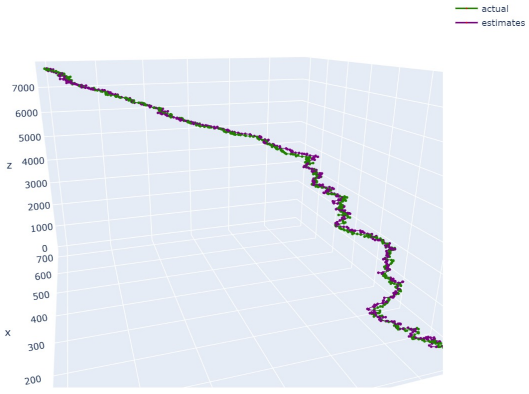
(a) $\sigma_s = 7, \sigma_{rx} = 1.2, \sigma_{r\dot{x}} = 0.01$, initial belief std = 0.01 and initial state = $[0, 0, 0, 0, 0, 0]$



(b) $\sigma_s = 8, \sigma_{rx} = 1.4, \sigma_{r\dot{x}} = 0.03$, initial belief std = 0.03 and initial state = $[5, 5, 5, 0, 5, 0]$



(c) $\sigma_s = 9, \sigma_{rx} = 1.6, \sigma_{r\dot{x}} = 0.05$, initial belief std = 0.05 and initial state = $[10, 10, 10, 0, 0, 10]$



(d) $\sigma_s = 10, \sigma_{rx} = 2, \sigma_{r\dot{x}} = 0.07$, initial belief std = 0.07 and initial state = $[15, 15, 15, 0, 0, 15]$

Figure 16: Simulation of strategy with different parameter settings

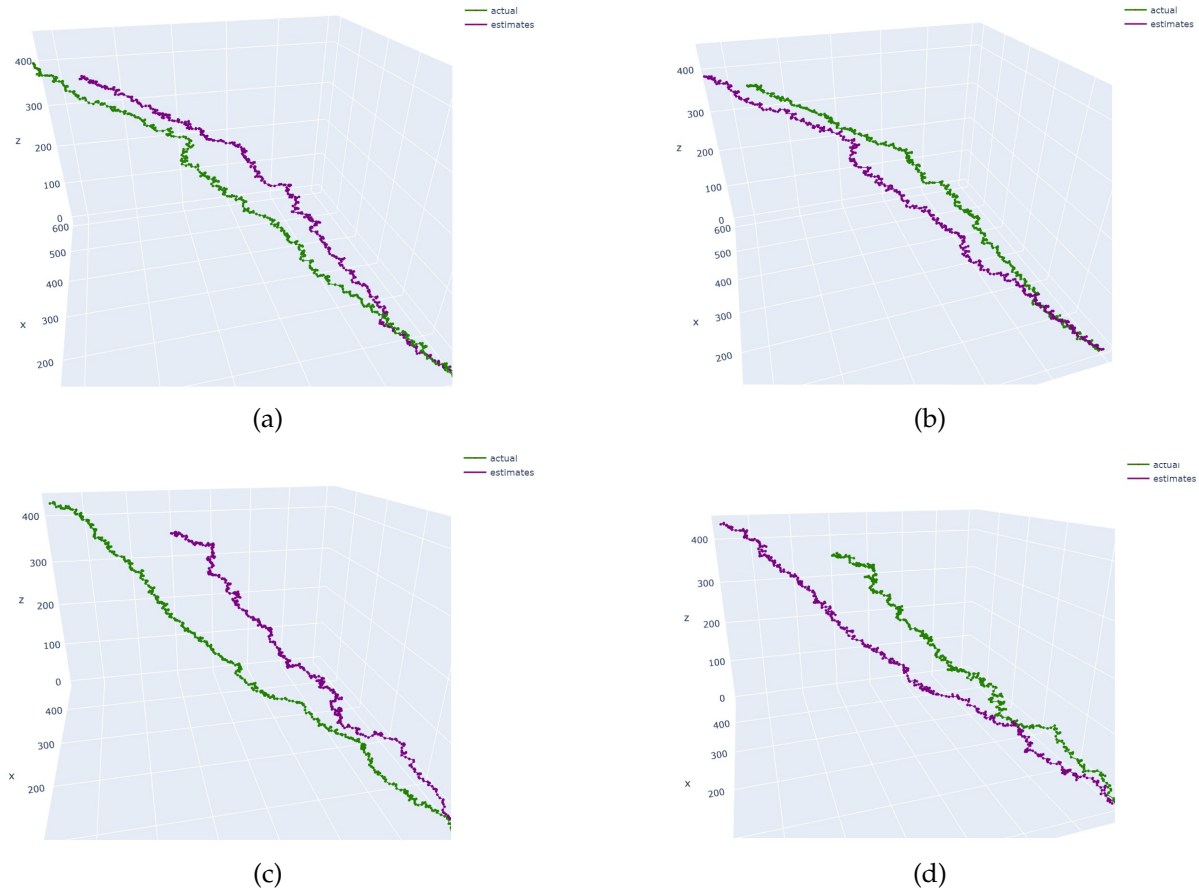


Figure 17: Simulation of strategy with same parameter settings

We observe that the Hungarian algorithm strategy also works well when the planes have different parameter settings (Fig.16). However, when the planes have identical parameters, the estimated trajectory deviates from the actual trajectory(Fig.17).

2 Landmark Localization

(a)

Currently we have the motion model and observation model defined as follows:

$$X_{t+1} = AX_t + Bu_t + \mathcal{N}(0, Q) \tag{10}$$

$$Z_t = CX_t + \mathcal{N}(0, R) \tag{11}$$

Let W_t be the distance of the plane from the landmark D at time t. The landmark-observation model is defined as follows:

$$W_t = h(X_t, D) + \mathcal{N}(0, S) \tag{12}$$

where $h(X_t, D)$ is the L_2 norm of the vector $CX_t - D$, i.e.,

$$h(X_t, D) = \sqrt{(CX_t - D)^T(CX_t - D)} \tag{13}$$

Since the above function is non-linear in X_t, we apply EKF linearization for X_t close to μ_t for a particular landmark D,

$$h(X_t, D) = h(\mu_t, D) + H(\mu_t)(X_t - \mu_t) \tag{14}$$

where,

$$H(\mu_t) = \frac{\partial h(X_t, D)}{\partial X_t}(\mu_t) = \frac{1}{h(\mu_t, D)} C^T (C\mu_t - D) \tag{15}$$

Therefore, to incorporate the landmark-observation we first check whether the plane is within a range of R of a particular landmark positioned at D at each time step. If we find one, we perform the EKF correction of the current belief$(\bar{\mu}_t, \bar{\Sigma}_t)$ as follows:

$$K_t = \bar{\Sigma}_t H(\bar{\mu}_t)^T (H(\bar{\mu}_t)\bar{\Sigma}_t H(\bar{\mu}_t)^T + S)^{-1} \tag{16}$$

$$\mu_t = \bar{\mu}_t + K_t(w_t - h(\bar{\mu}_t, D)) \tag{17}$$

$$\Sigma_t = (I - K_t H(\bar{\mu}_t))\bar{\Sigma}_t \tag{18}$$

If there are multiple landmarks, we take the one which is the nearest.

(b)

```
T = 200
#Standard Deviation in GPS measurement
sig_s = 10

#Standard Deviation in motion
sig_rx = 0.01
sig_ry = 0.01
sig_rz = 0.01
sig_rx_ = 0.01
sig_ry_ = 0.01
sig_rz_ = 0.01

#Standard Deviation in landmark-observations
sig_dist = 1

init_bel_sig = 0.01 #Initial belief standard deviation
init_state = np.array([100,0,0,0,4,0]) # Initial State of the Plane
landmarks = [np.array([150,0,100]), np.array([-150,0,100]), np.array([0,150,100]), np.array([0,-150,100])] #Landmark positions
rng = 90 #Range
t_values = np.arange(T)

#Velocity increments
u = np.vstack([-0.128*np.cos(0.032*t_values), -0.128*np.sin(0.032*t_values), np.array([0.01]*t_values)]).T

plane = Plane(sig_s,sig_rx,sig_ry,sig_rz,sig_rx_,sig_ry_,sig_rz_,sig_dist,init_bel_sig,u,init_state, T)
```

Figure 18: Simulation parameters

```
for t in range(T-1):
    X = plane.simulate(t+1) #simulate the plane
    plane.observe(t+1) #get GPS observation
    plane.estimate(t+1) #Propagate the belief

    #Find the nearest landmark
    best_dist = 1e16
    nearest_landmark = np.zeros(3)
    for landmark in landmarks:
        curr_dist = plane.dist(X,landmark)
        if(curr_dist < best_dist):
            best_dist = curr_dist
            nearest_landmark = landmark
    #Check whether the landmark is within range
    if(best_dist <= rng):
        plane.landmark_update(t+1,nearest_landmark)
    plane.update_cov_list()
```

Figure 19: Simulation accounting for landmark observations

(c)

Using the simulation parameters defined in Fig. 18, we obtain plots shown in Fig.20 and 21. In these plots, we observe that the uncertainty of the estimated trajectory is very low in the regions influenced by landmarks. This is due to the fact that we are accounting for another set of measurements, i.e., landmark-observations while updating the belief. (Eq.18).

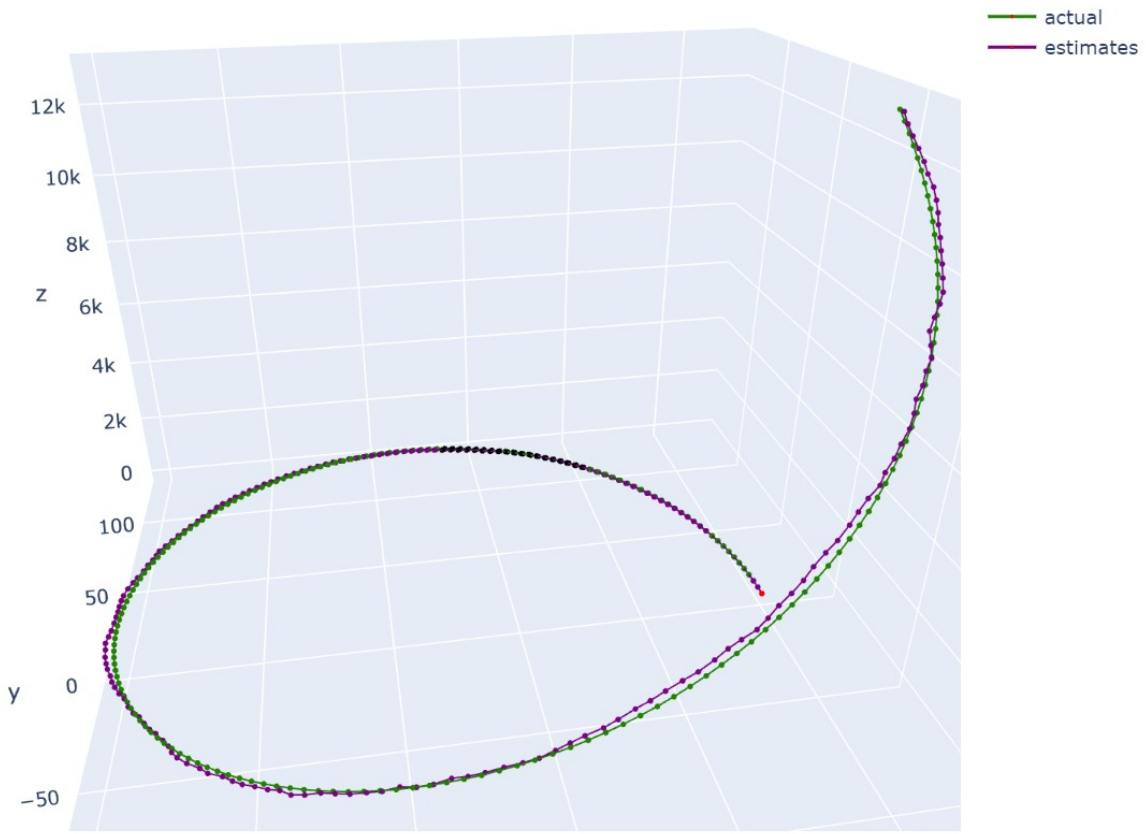


Figure 20: Plots of actual and estimated trajectory (the black points represents regions of influence landmarks)

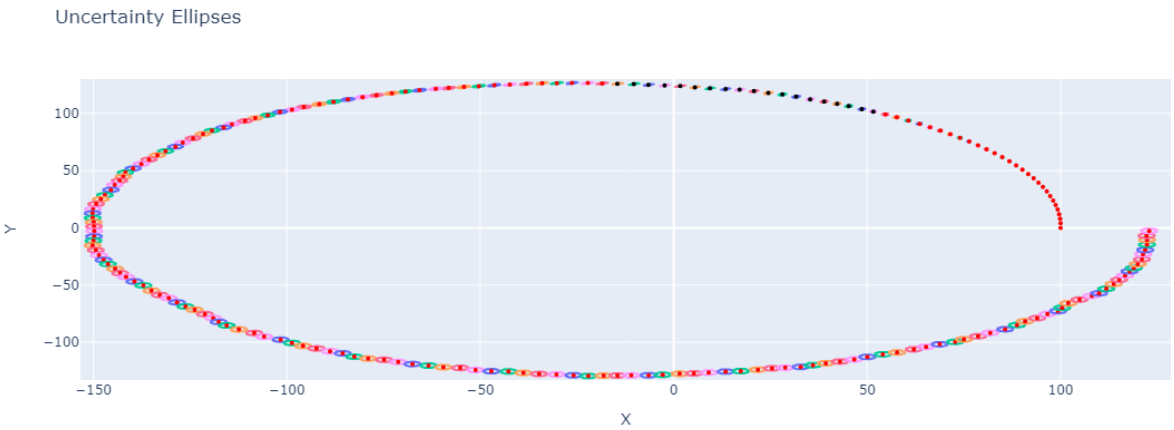


Figure 21: Uncertainty ellipses (the black points represents regions of influence landmarks)

(d)

- On decreasing the landmark distance standard deviation from 1 to 0.1 the estimated trajectory follows the actual trajectory more precisely(Fig.23). This is because we are now providing the plane with more accurate measurements of the distance leading to decrease in uncertainty in estimates(as is evident in uncertainty plot in Fig.24)
- On increasing the landmark distance standard deviation from 1 to 20 the estimated trajectory deviates more from the actual trajectory (Fig.25). This is because we are now providing the plane with more inaccurate measurements of the distance leading to increase in uncertainty in estimates(as is evident in uncertainty plot in Fig.26)

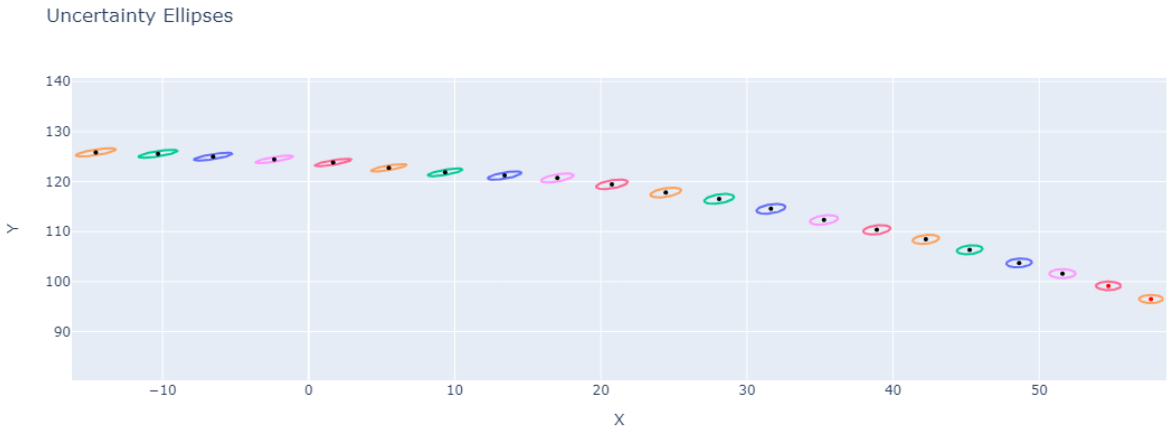


Figure 22: Uncertainty ellipses with $\sqrt{S} = 1$(Focusing on regions of influence of landmarks, zoomed in version of Fig. 21)

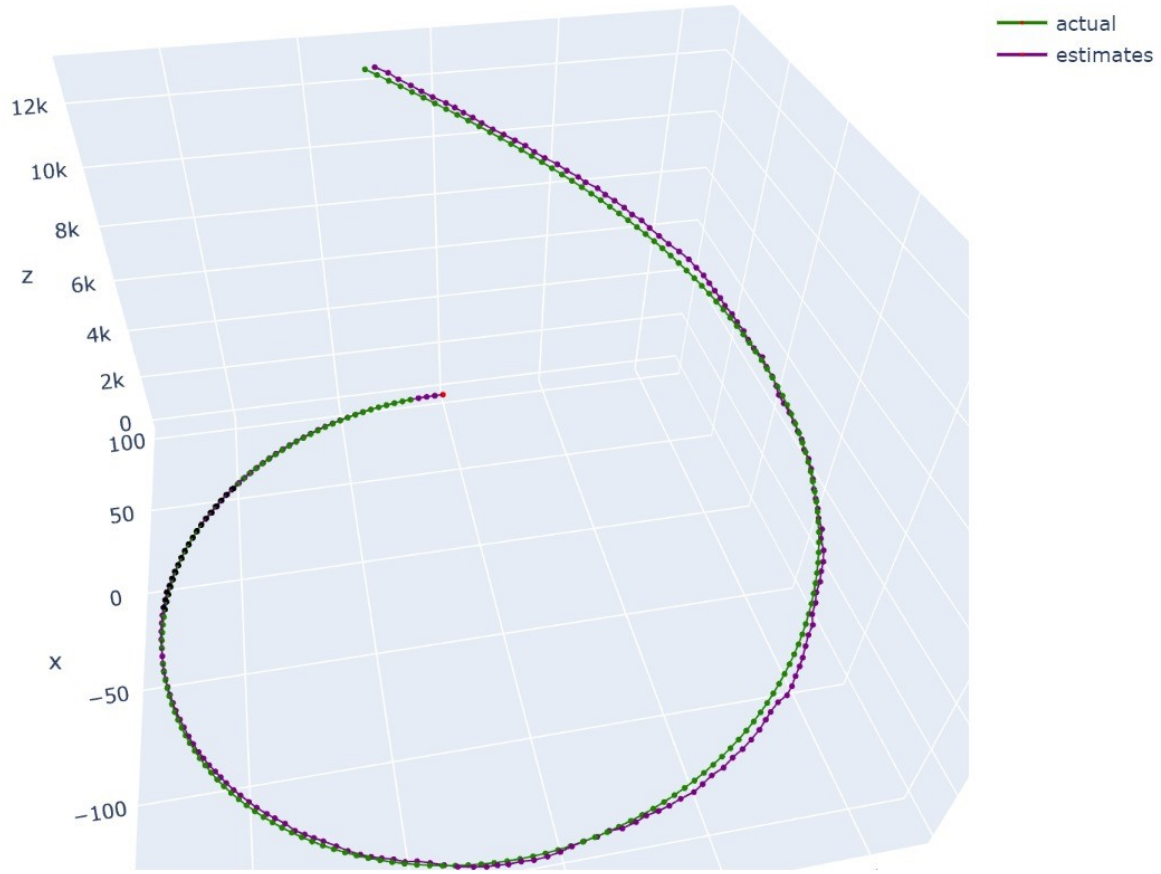


Figure 23: Plots of actual and estimated trajectory with $\sqrt{S} = 0.1$

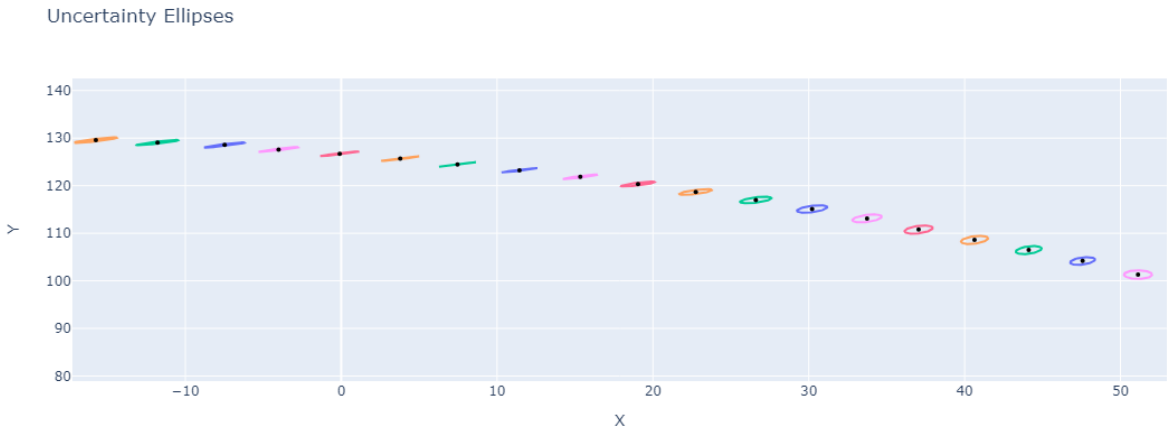


Figure 24: Uncertainty ellipses with $\sqrt{S} = 0.1$(Focusing on regions of influence of landmarks)

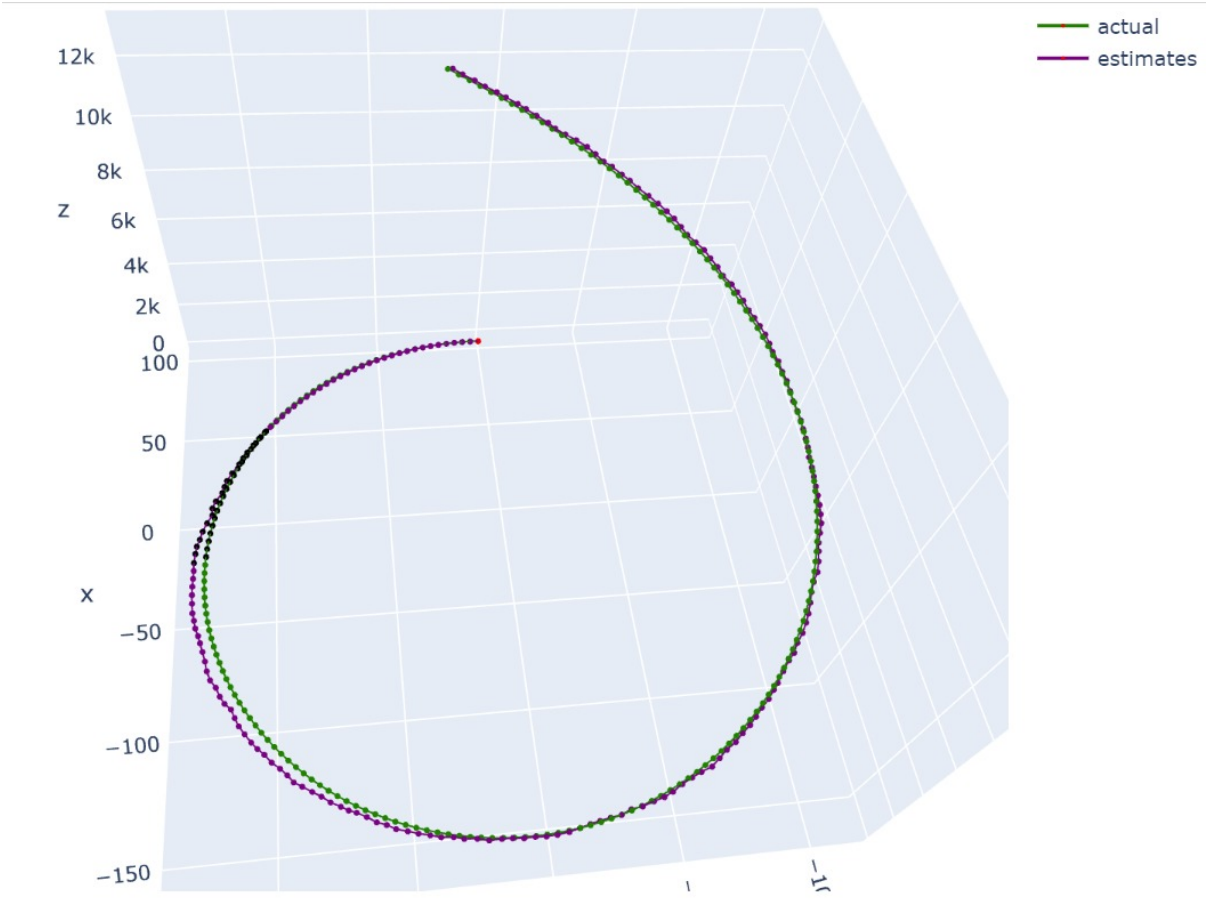


Figure 25: Plots of actual and estimated trajectory with $\sqrt{S} = 20$

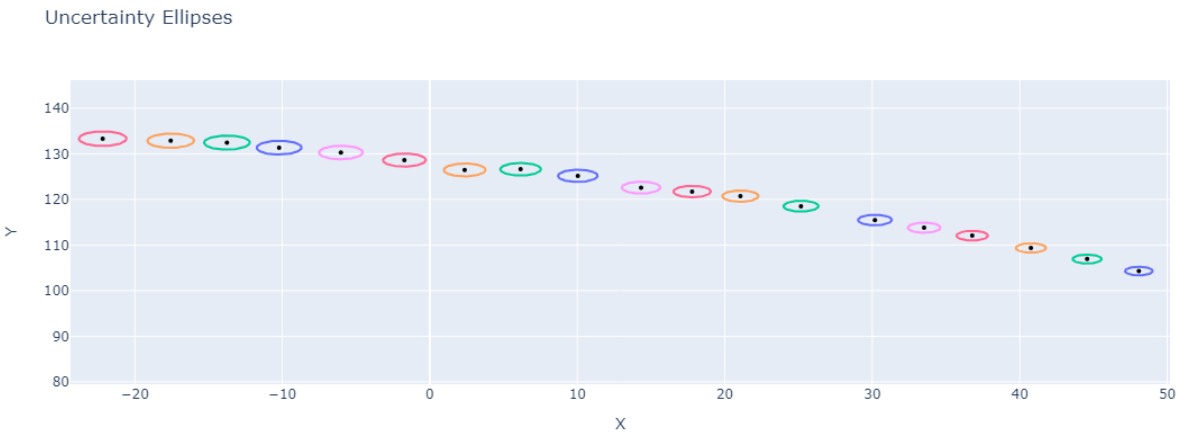


Figure 26: Uncertainty ellipses with $\sqrt{S} = 20$ (Focusing on regions of influence of landmarks)