

ENAE 441 - 0101
Final Project: Extended Kalman Filter

Due on December 16th, 2025 at 12:30 PM

Dr. Martin, 09:30 AM

Vai Srivastava

December 19th, 2025

Overview

You are working at NASA GSFC on the navigation team for an Earth orbiting science satellite launched into orbit on a Falcon-9. SpaceX designed its launch to deliver the payload into low Earth orbit with reference coordinates of

$$\mathbf{X}_{\infty} = \begin{bmatrix} a \\ e \\ i \\ \omega \\ \Omega \\ \theta \end{bmatrix} = \begin{bmatrix} 7 \times 10^3 \text{ km} \\ 0.2 \\ 45^\circ \\ 0^\circ \\ 270^\circ \\ 78.75^\circ \end{bmatrix}$$

however it's the navigation's team responsibility to verify this orbit and/or determine any errors in its delivery. To accomplish this, the navigation team has access to range and range-rate measurements from the spacecraft provided by the Deep Space Network (DSN). The latitude and longitude (ϕ, λ) of the three DSN ground stations are provided below:

- DSN # 0: Goldstone, USA: $(35.297^\circ, -116.914^\circ)$
- DSN # 1: Madrid, Spain: $(40.4311^\circ, -4.248^\circ)$
- DSN # 2: Canberra, Australia: $(-35.4023^\circ, 148.9813^\circ)$

These ground stations are positioned on a spherical Earth of radius 6378.137 km , with a rotation rate of $\omega_{E/N} = 7.292\ 115 \times 10^{-5} \frac{\text{rad}}{\text{s}}$ and a Local Sidereal Time of $\gamma_0 = 0^\circ$.

`Project-Measurements-Easy.npy` contains the range ρ and range-rate $\dot{\rho}$ measurements alongside additional information. Specifically, each row in the datafile is formatted as:

$$[t, i, \rho, \dot{\rho}]$$

where t corresponds with the time the measurement was received and i corresponds to the ground station index (as labeled above).

As best as you are aware, the spacecraft's motion is governed by the following differential equation:

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3} \cdot \mathbf{r}$$

where \mathbf{r} is the position of the spacecraft in the inertial frame. Similarly, the measurements provided can be computed using the following equations:

$$\begin{aligned} \boldsymbol{\rho} &= \mathbf{r} - \mathbf{R}_{\text{site},i} \\ \rho &= \|\mathbf{r} - \mathbf{R}_{\text{site},i}\| \\ \dot{\rho} &= \frac{\boldsymbol{\rho} (\dot{\mathbf{r}} - \dot{\mathbf{R}}_{\text{site},i})}{\rho} \\ \dot{\mathbf{R}}_{\text{site},i} &= \boldsymbol{\omega}_{E/N} \times \mathbf{R}_{\text{site},i} \end{aligned}$$

where $\mathbf{R}_{\text{site},i}$ is the location of the DSN station also in the inertial frame. Note that the measurements provided by the DSN have some intrinsic noise which can be modeled as Gaussian white noise. Explicitly, the noise in the range is characterized by a variance of 1 m^2 , and by $1 \frac{\text{cm}^2}{\text{s}^2}$ variance in the range-rate. Using your knowledge of the spacecraft's dynamics and the measurements provided by the DSN, generate a report which culminates in an estimate of the spacecraft's state over time. To help facilitate your progress, please generate your report in the following order, answering the following intermediate questions:

Problem 1: Problem Setup

- a. Express the non-linear system in continuous time state-space form, clearly defining the vectors $\mathbf{f}(X(t))$ and $\mathbf{h}(X(t))$
- b. Define the linearized dynamics and measurement matrices $A(t)$ and $C(t)$.
- c. Show how these matrices are converted to their discrete time forms F_k and H_k . Recall F_k is the state transition matrix $\Phi(t_j, t_i)$ which requires integration.
- d. Define your noise matrices Q_k and R_k , and discuss their relationship to the aforementioned system of equations.
- e. Plot the measurements as a function of time.

Solution

Part A

First, we use the inertial Cartesian state:

$$X(t) \equiv \begin{bmatrix} r(t) \\ v(t) \end{bmatrix} = [x \ y \ z \ \dot{x} \ \dot{y} \ \dot{z}]^T \in \mathbb{R}^6, \quad r, v \in \mathbb{R}^3$$

And the two-body point mass model:

$$\dot{r} = v, \quad \dot{v} = -\mu \frac{r}{\|r\|^3}$$

Thus,

$$f(X, t) = \begin{bmatrix} v \\ -\mu \frac{r}{\|r\|^3} \end{bmatrix}$$

Now, each station's ECEF position is:

$$R_{\text{ECEF},i} = R_E \begin{bmatrix} \cos(\phi_i) \cos(\lambda_i) \\ \cos(\phi_i) \sin(\lambda_i) \\ \sin(\phi_i) \end{bmatrix}$$

As Earth rotates about inertial $+\hat{z}$ with $\omega_E = \omega_{\mathcal{E}/\mathcal{N}}$, siderial angle is:

$$\gamma(t) = \gamma_0 + \omega_E t$$

Rotating to get the Site positions:

$$R_{\text{site},i} = R_3(\gamma(t)) R_{\text{ECEF},i}, \quad \dot{R}_{\text{site},i}(t) = \omega \times R_{\text{site},i}(t), \quad \omega = \begin{bmatrix} 0 \\ 0 \\ \omega_E \end{bmatrix}$$

Now, each station's LOS vector is:

$$\rho_i(t) = r(t) - R_{\text{site},i}(t), \quad \rho(t) = \|\rho_i(t)\|, \quad \hat{\rho}(t) = \frac{\rho_i(t)}{\rho(t)}$$

Range-rate:

$$\dot{\rho}(t) = \frac{\rho_i^T(v - \dot{R}_{\text{site},i})}{\rho} = \hat{\rho}^T(v - \dot{R}_{\text{site},i})$$

Measurements including noise:

$$y(t) = h(X, t, i) + v(t), \quad v(t) \sim \mathcal{N}(0, R)$$

Part B

With δX as a small perturbation about a reference trajectory $\bar{X}(t)$,

$$\delta \dot{X}(t) = A(t) \delta X(t), \quad \delta y(t) = C(t) \delta X(t) + v(t)$$

$A(t)$ and $C(t)$ are Jacobians:

$$A(t) = \left. \frac{\partial f}{\partial X} \right|_{\bar{X}(t)}, \quad C(t) = \left. \frac{\partial h}{\partial X} \right|_{\bar{X}(t), t, i}$$

Dynamics Jacobian $A(t)$:

With $r = \bar{r}(t)$, $v = \bar{v}(t)$, $r = \|r\|$,

$$A(t) = \begin{bmatrix} \frac{\partial \dot{r}}{\partial r} & \frac{\partial \dot{r}}{\partial v} \\ \frac{\partial \dot{v}}{\partial r} & \frac{\partial \dot{v}}{\partial v} \end{bmatrix} = \begin{bmatrix} 0_{3 \times 3} & I_{3 \times 3} \\ -\mu \left(\frac{1}{r^3} I_{3 \times 3} - \frac{3}{r^5} rr^T \right) & 0_{3 \times 3} \end{bmatrix}$$

Measurement Jacobian $C(t)$:

For a measurement taken at time t from station i ,

$$\rho = r - R_{\text{site},i}(t), \quad \rho = \|\rho\|, \quad u = \hat{\rho} = \rho/\rho, \quad v_{\text{rel}} = v - \dot{R}_{\text{site},i}(t)$$

Range row:

$$\frac{\partial \rho}{\partial r} = u^T, \quad \frac{\partial \rho}{\partial v} = 0_{1 \times 3}$$

Range-rate row:

$$\dot{\rho} = u^T v_{\text{rel}}$$

Using $\frac{\partial u}{\partial r} = \frac{1}{\rho} (I - uu^T)$ (since $u = \rho/\|\rho\|$ and ρ depends linearly on r),

$$\frac{\partial \dot{\rho}}{\partial v} = u^T, \quad \frac{\partial \dot{\rho}}{\partial r} = v_{\text{rel}}^T \frac{\partial u}{\partial r} = \frac{1}{\rho} v_{\text{rel}}^T (I - uu^T)$$

Thus,

$$C(t) = \begin{bmatrix} u^T & 0_{1 \times 3} \\ \frac{1}{\rho} v_{\text{rel}}^T (I - uu^T) & u^T \end{bmatrix}$$

Part C

Measurements occur at times t_k (not necessarily uniform), meaning the EKF-style discrete model is:

$$X_{k+1} = \varphi(X_k, t_k, t_{k+1}) + w_k, \quad y_k = h(X_k, t_k, i_k) + v_k$$

The discrete linearized mapping for perturbations is:

$$\delta X_{k+1} = F_k \delta X_k, \quad F_k \equiv \Phi(t_{k+1}, t_k)$$

Φ is obtained by integrating the variational equation alongside the state:

$$\dot{\Phi}(t, t_k) = A(t) \Phi(t, t_k), \quad \Phi(t_k, t_k) = I_{6 \times 6},$$

and then setting:

$$F_k = \Phi(t_{k+1}, t_k)$$

At measurement time t_k and station index i_k ,

$$H_k \equiv C(t_k) = \frac{\partial h}{\partial X} \Big|_{\bar{X}(t_k), t_k, i_k}$$

Part D

Measurement noise is simply:

$$R_k = \begin{bmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\rho^2 \end{bmatrix}$$

Process noise is used to account for unmodeled accelerations. Here we choose additive white acceleration noise as our process noise model:

$$\dot{r} = v, \quad \dot{v} = -\mu \frac{r}{\|r\|^3} + w_a(t), \quad w_a(t) \sim \mathcal{N}(0, Q_c),$$

with $Q_c = q_a I_{3 \times 3}$, and simplifying to:

$$\dot{X} = f(X, t) + Gw_a(t), \quad G = \begin{bmatrix} 0_{3 \times 3} \\ I_{3 \times 3} \end{bmatrix}$$

Then the discrete process noise over $[t_k, t_{k+1}]$ is

$$Q_k = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau) G Q_c G^T \Phi(t_{k+1}, \tau)^T d\tau$$

Part E

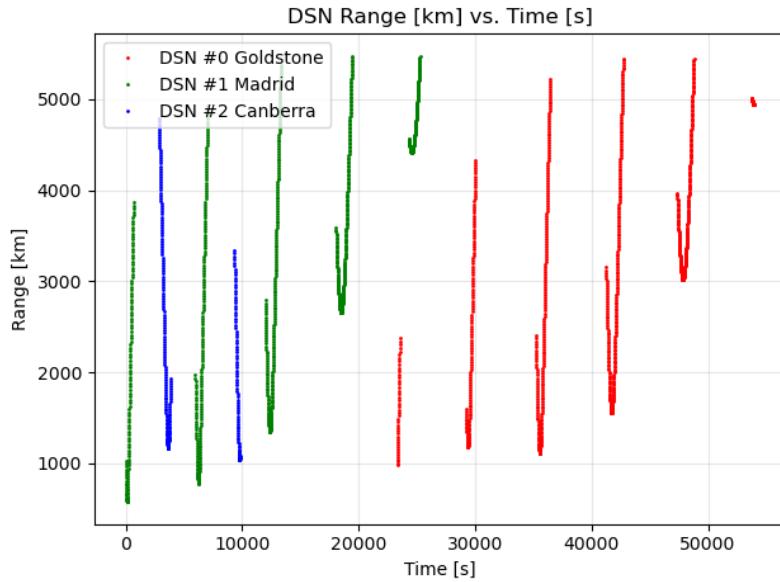


Figure 1: DSN Range [km] vs. Time [s]

Problem 2: Plan Filter Implementation

Provide pseudocode from which you will base your extended Kalman filter implementation. Highlight the major steps in your algorithm and any noteworthy modifications or subtle details required for this problem that you want the grades to be aware of. Be comprehensive, as this is what the grading team will primarily reference if the results/plots don't quite look right.

Solution

```

1 Inputs:
2   data.t[0..N-1], data.i[0..N-1], data.rho[0..N-1], data.drho[0..N-1]
3   station lat/lon: (phi[i], lam[i]), i=0..2
4   constants: mu, RE, omegaE, gamma0
5   noise: R = diag(sig_rho^2, sig_drho^2)  (in FILTER UNITS)
6       Q-model params: q_a (white accel PSD) OR set Qk = 0
7   initial: X_plus (6x1), P_plus (6x6)
8
9 Precompute:
10  for each station i:
11    R_ecef[i] = RE * [cos(phi_i)cos(lam_i), cos(phi_i)sin(lam_i), sin(phi_i)]^T
12
13 Helpers:
14  site_eci(i, t):
15    gamma = gamma0 + omegaE*t
16    R_site = R3(gamma) * R_ecef[i]
17    Rdot_site = omega x R_site # (omega = [0,0,omegaE]^T)
18    return (R_site, Rdot_site)
19
20 dynamics(X):
21   r = X[0:3], v = X[3:6]
22   rnorm = ||r||
23   rdot = v
24   vdot = -mu * r / rnorm^3
25   return [rdot; vdot]
26
27 A_matrix(X):
28   r = X[0:3]; rnorm = ||r||
29   I3 = identity(3)
30   dadr = -mu*( (1/rnorm^3)*I3 - (3/rnorm^5)*(r*r^T) )
31   return [[0, I3],
32           [dadr, 0]]
33
34 propagate_state_and_stm(X0, t0, t1):
35   Integrate coupled ODEs from t0 -> t1:
36   Xdot = dynamics(X)
37   Phidot = A_matrix(X) * Phi
38   with initial Phi(t0)=I6
39   return (X_minus, Phi) where Phi = phi(t1,t0)
40

```

```

41     meas_and_jacobian(X, i, t):
42         (R_site, Rdot_site) = site_eci(i, t)
43         r = X[0:3]; v = X[3:6]
44         rho_vec = r - R_site
45         rho = ||rho_vec||
46         u = rho_vec / rho
47         v_rel = v - Rdot_site
48         drho = u^T * v_rel
49
50         yhat = [rho; drho]
51
52         # Jacobian H = ∂h/∂X (2x6)
53         H_rho_r = u^T
54         H_rho_v = [0 0 0]
55
56         H_drho_v = u^T
57         H_drho_r = (1/rho) * (v_rel^T * (I3 - u*u^T))
58
59         H = [[H_rho_r, H_rho_v],
60               [H_drho_r, H_drho_v]]
61         return (yhat, H)
62
63     process_noise_discrete(dt, q_a):
64         if q_a == 0:
65             return 0_(6x6)
66         else:
67             I3 = identity(3)
68             Q = q_a * [[[dt^3/3)*I3, (dt^2/2)*I3],
69                         [(dt^2/2)*I3, (dt)*I3]]
70             return Q
71
72 Main EKF:
73     t_prev = data.t[0]
74
75     for k in 0..N-1:
76         t_k = data.t[k]
77         i_k = data.i[k]
78         y_k = [data.rho[k], data.drho[k]]^T
79
80         # Predict / propagate to t_k
81         dt = t_k - t_prev
82         if dt > 0:
83             (X_minus, Phi) = propagate_state_and_stm(X_plus, t_prev, t_k)
84             Qk = process_noise_discrete(dt, q_a)
85             P_minus = Phi * P_plus * Phi^T + Qk
86         else:
87             X_minus = X_plus
88             P_minus = P_plus

```

```

89     Phi = I6
90
91     # Measurement prediction at (t_k, i_k)
92     (yhat_k, Hk) = meas_and_jacobian(X_minus, i_k, t_k)
93     nu = y_k - yhat_k # innovation (2x1)
94
95     # Gain computation
96     S = Hk * P_minus * Hk^T + R # (2x2)
97     K = P_minus * Hk^T * inv(S) # (6x2)
98
99     # Update (state + Joseph covariance)
100    X_plus = X_minus + K * nu
101    P_plus = (I6 - K*Hk)*P_minus*(I6 - K*Hk)^T + K*R*K^T
102
103    # Diagnostics
104    log residual nu, NIS = nu^T * inv(S) * nu, etc.
105    t_prev = t_k
106
107 Outputs:
108     filtered state estimates X_plus and X_minus over time
109     residual history, NIS history, covariance history

```

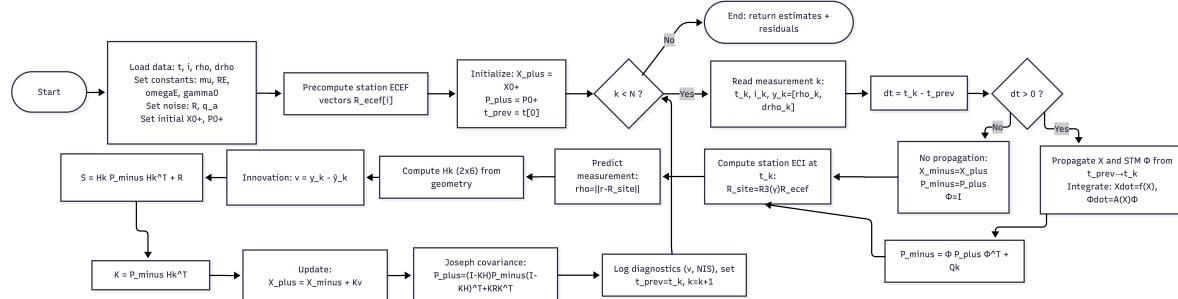


Figure 2: (Pseudo) Code Flowchart

Problem 3: Pure Prediction

- Specify your choice of \mathbf{x}_0 , P_0 , Q_0 , and R_0 . Explain your reasoning. (For \mathbf{x}_0 , express in cartesian coordinates.)
- Implement just the prediction step of the extended Kalman Filter (i.e. do not perform measurement updates).
- Plot the covariance of the state estimate as a function of time, centered about a mean of zero — i.e. $\pm 3\sigma$ bounds taken from $P_{\hat{x},k}^- (t)$. Choose appropriate y-limits for your plot to maximize readability. Explain what you see and if it makes sense.

Solution

Part A

\mathbf{x}_0 is the nominal spacecraft orbit:

$$\mathbf{X}_0^+ \equiv \begin{bmatrix} r_0 \\ v_0 \end{bmatrix}$$

P_0 is chosen to be a diagonal covariance matrix, with the first 3 terms being the range noise uncertainty scaled by the tuning ratio:

$$\sigma_r^2 = \left(\frac{\sigma_\rho}{\sigma_a} \right)^2 = \left(\frac{1 \times 10^{-6}}{1 \times 10^{-9}} [\text{km}] \right)^2$$

and the final 3 terms being the range-rate noise uncertainty scaled by the tuning ratio:

$$\sigma_v^2 = \left(\frac{\sigma_{\dot{\rho}}}{\sigma_a} \right)^2 = \left(\frac{1 \times 10^{-10}}{1 \times 10^{-9}} \left[\frac{\text{km}}{\text{s}} \right] \right)^2$$

Q_0 is chosen to be the short-step approximation:

$$Q_k \approx q_a \begin{bmatrix} \frac{\Delta t^3}{3} I_3 & \frac{\Delta t^2}{2} I_3 \\ \frac{\Delta t^2}{2} I_3 & \Delta t I_3 \end{bmatrix}$$

R_0 is chosen to be a diagonal covariance matrix, with the first 3 terms being the range noise uncertainty:

$$\sigma_\rho^2 = (1 \times 10^{-3} [\text{km}])^2$$

and the final 3 terms being the range-rate noise uncertainty

$$\sigma_{\dot{\rho}}^2 = \left(1 \times 10^{-10} \left[\frac{\text{km}}{\text{s}} \right] \right)^2$$

```

1 | X₀⁺ =
2 | [ 6.34337103e+03 -8.92209622e+02 -8.92209622e+02 3.04285069e+00
3 |   5.34125353e+00 5.34125353e+00]
4 | P₀⁺ =
5 | [[1.e+06 0.e+00 0.e+00 0.e+00 0.e+00 0.e+00]
6 |   [0.e+00 1.e+06 0.e+00 0.e+00 0.e+00 0.e+00]
7 |   [0.e+00 0.e+00 1.e+06 0.e+00 0.e+00 0.e+00]
8 |   [0.e+00 0.e+00 0.e+00 1.e-02 0.e+00 0.e+00]
9 |   [0.e+00 0.e+00 0.e+00 0.e+00 1.e-02 0.e+00]
10 |   [0.e+00 0.e+00 0.e+00 0.e+00 0.e+00 1.e-02]]
11 | R₀ =
12 | [[1.e-06 0.e+00]
13 |   [0.e+00 1.e-10]]]
```

Part B

Implementation can be viewed in the [Python files](#) embedded at the end of the report.

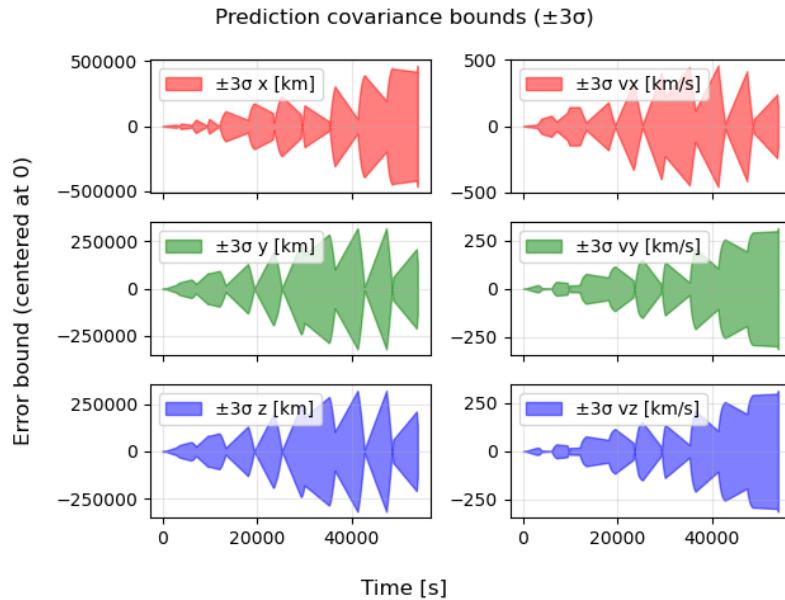
Part C

Figure 3: $\pm 3\sigma$ Bounds on Covariance $P_{\hat{x},k}^-$ vs. Time t

Answer

Problem 4: Measurement Updates

- Implement the measurement update step of the extended Kalman Filter.
- Plot the pre- and post-measurement update $\pm 3\sigma$ bounds on the same plot. Explain any differences.
- Plot the difference between the pre-and post-measurement update state estimates, $\mu_{\hat{x},k}^+(t_k) - \mu_{\hat{x},k}^-(t_k)$, inside of the pre-measurement covariance $\pm 3\sigma$ bounds $P_{\hat{x},k}^+(t_k)$.

Solution**Part A**

Implementation can be viewed in the [Python files](#) embedded at the end of the report.

Part B

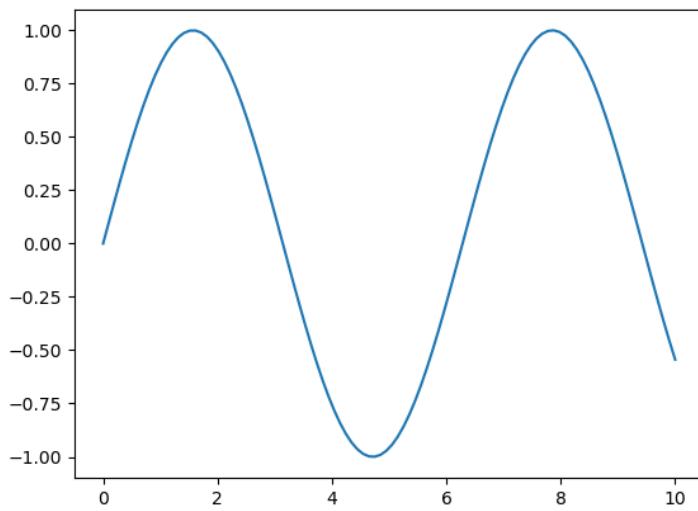


Figure 4: Example

Answer

Part C

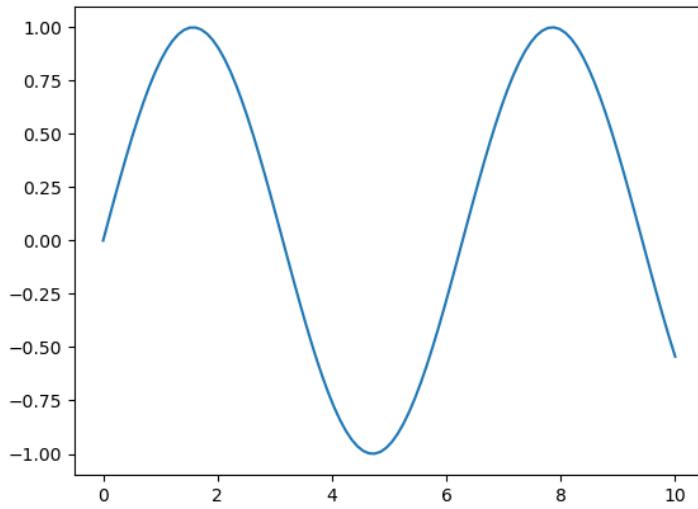


Figure 5: Example

Problem 5: Filter Solutions

- a. With your EKF implemented, run the simulation through to completion and compute the post-fit measurement residuals as a function of time via:

$$\delta \mathbf{Y} (t_k) = \mathbf{Y} (t_k) - h \left(\boldsymbol{\mu}_{\dot{\mathbf{x}},k}^+ (t_k) \right)$$

- b. Do your measurement residuals give you confidence that you estimated the state correctly? Why or why not?
- c. Plot the estimated state $\boldsymbol{\mu}_{\dot{\mathbf{x}},k}^+ (t_k)$ and its associated $\pm 3\sigma$ bounds taken from $P_{\dot{\mathbf{x}},k}^+ (t)$ as a function of time.
- d. Report your best state estimate at the final time-step and its associated uncertainty.

Solution

Part A

Answer

Part B

Answer

Part C

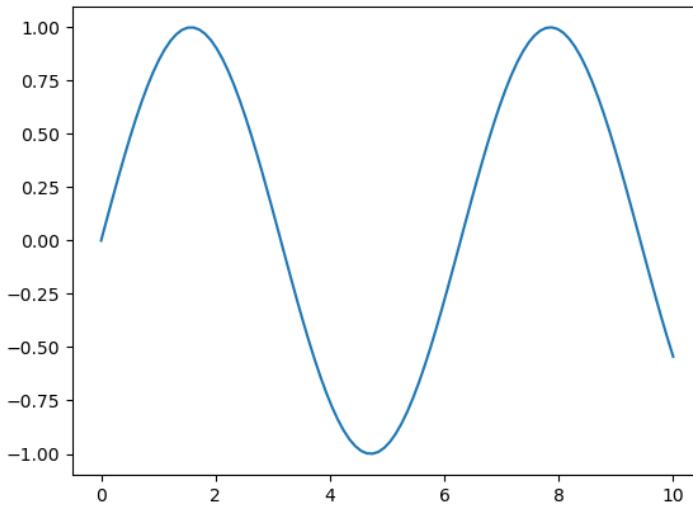


Figure 6: Example

Answer

Part D

Answer

Problem 6: Debugging Efforts (*Optional*)

Use this section to outline any of your debugging efforts for if things aren't going your way. This is a good place to earn some partial credit. This should be a “research” log of what experiments you performed and why. A list of guiding questions if you’re stuck include:

1. Consider how process noise matrix is used in the filter. What happens if there are large gaps between measurements?
2. Consider if the values used in your measurement noise matrix are appropriate. Should these values only reflect the uncertainty in the sensor?
3. If your filter is diverging, does the divergence start at the beginning or mid-way through? What possible reasons exist for either outcome?
4. If you had to define a single scalar metric to evaluate your filter’s quality, what would it be, and can you use this to help you determine optimal tuning values?
5. Does plotting your best estimate in a different reference frame or element description help?

Solution

Answer

Problem 7: Challenge Orbit (*Bonus*)

Perform a second analysis with the more difficult dataset `Project-Measurements-Hard.npy`. Use

$$\begin{bmatrix} a \\ e \\ i \\ \omega \\ \Omega \\ \theta \end{bmatrix} = \begin{bmatrix} 7000 \text{ km} \\ 0.6 \\ 45^\circ \\ 180^\circ \\ 0^\circ \\ 45^\circ \end{bmatrix}$$

as your initial guess. Your analysis can amount to your debugging process, and points will be awarded based on how thoughtful your experimentation is and the quality of your solution.

Solution

Answer

Code

See the [full source code](#) for this project.

`./src/final.py`

```

1 # libs
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import os
5
```

```

6  # src
7  from measurements import *
8  from helpers import *
9  import p01
10 import p03
11 import p04
12
13 def main() -> int:
14     # constants
15     EPS = 1e-12 # [-]
16     MU = 398600.4418 # [km^3/s^2]
17     RE = 6378.137 # [km]
18     OMEGA_E = 7.292115e-5 # [rad/s]
19     GAMMA0 = 0.0 # [rad]
20
21     # givens
22     oe = [
23         7e3, # [km]
24         0.2, # [-]
25         np.deg2rad(45), # [deg -> rad]
26         np.deg2rad(0), # [deg -> rad]
27         np.deg2rad(270), # [deg -> rad]
28         np.deg2rad(78.75) # [deg -> rad]
29     ]
30     σ = [
31         1e-6, # [km^2]
32         1e-10 # [km^2/s^2]
33     ]
34     stations = {
35         0: (35.297, -116.914), # [lat, long]
36         1: (40.4311, -4.248), # [lat, long]
37         2: (-35.4023, 148.9813), # [lat, long]
38     }
39
40     # process noise tuning
41     σ_a = 1e-9 # [km/s^2]
42
43     # load numpy measurements
44     data = numpy_data.load("./data/Project-Measurements-Easy.npy")
45
46     #####
47     # p01 #
48     #####
49     # p01e
50     p01.e(data).savefig("./outputs/figures/s01e.png")
51
52     #####
53     # p03 #

```

```

54 #####
55 # p03a
56 X0_plus, P0_plus, R0 = p03.a(oe, σ, σ_a, MU)
57 with open("./outputs/text/s03a.txt", "w", encoding="utf-8") as f:
58     f.write(f"X₀⁺ =\n{X0_plus}\n")
59     f.write(f"P₀⁺ =\n{P0_plus}\n")
60     f.write(f"R₀ =\n{R0}")
61 # p03c
62 p03.c(data, X0_plus, P0_plus, σ_a, MU).savefig("./outputs/figures/s03c.png")
63
64 return 0
65
66
67 if __name__ == "__main__":
68     main()

```

./src/p01.py

```

1 # libs
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # src
6 from measurements import *
7
8 def e(data: measurements.Measurement) -> plt.Figure:
9     t = np.asarray(data.t)
10    i = np.asarray(data.i).astype(int)
11    p = np.asarray(data.p)
12
13    labels = {0: "DSN #0 Goldstone", 1: "DSN #1 Madrid", 2: "DSN #2 Canberra"}
14    colors = {0: "r", 1: "g", 2: "b"}
15
16    order = np.argsort(t)
17    t, i, p = t[order], i[order], p[order]
18
19    fig, ax = plt.subplots()
20
21    for stn in np.unique(i):
22        mask = (i == stn)
23        ax.plot(t[mask], p[mask],
24                 linestyle="--", linewidth=0,
25                 marker=".", markersize=2,
26                 color=colors.get(int(stn)),
27                 label=labels.get(int(stn), f"DSN #{int(stn)}"))
28    ax.grid(True, alpha=0.3)
29    ax.legend(loc = "upper left")
30
31    ax.set_xlabel("Time [s]")

```

```

32     ax.set_ylabel("Range [km]")
33     ax.set_title("DSN Range [km] vs. Time [s]")
34
35     fig.tight_layout()
36
37     return fig

```

./src/p03.py

```

1  # libs
2  from scipy.integrate import solve_ivp
3  import matplotlib.pyplot as plt
4  import numpy as np
5
6  # src
7  from measurements import *
8  from helpers import *
9
10 def a(oe: list, σ: list, σ_a: float, μ: float) -> [np.ndarray, np.ndarray, float,
11    ← np.ndarray]:
12    # choose x₀ from starting OE
13    r₀, v₀ = system.coe2rv(oe, μ)
14    X₀_plus = np.hstack([r₀, v₀])
15
16    # choose P₀ from starting range variance
17    σ_r, σ_v = [float(x)/σ_a for x in σ]
18    P₀_plus = np.diag([σ_r**2]*3 + [σ_v**2]*3)
19
20    # choose R₀ from measurement noise
21    R₀ = np.diag(σ)
22
23    return [X₀_plus, P₀_plus, R₀]
24
25 def c(data: measurements.Measurement, X₀_plus: np.ndarray, P₀_plus: np.ndarray, σ_a: float,
26    ← μ: float) -> plt.Figure:
27    t_pred, X_minus_hist, P_minus_hist = propagators.predict_only(data, X₀_plus, P₀_plus,
28    ← σ_a, μ)
29
30    os = np.sqrt(np.clip(np.stack([np.diag(P) for P in P_minus_hist]), 0.0, np.inf)) # Nx6
31    bounds = 3.0 * os # Nx6
32
33    state_names = ["x [km]", "y [km]", "z [km]", "vx [km/s]", "vy [km/s]", "vz [km/s]"]
34    colors = {0: "r", 1: "g", 2: "b"}
35
36    fig, axs = plt.subplots(nrows=3, ncols=2, sharex=True, sharey=False)
37
38    for j in range(len(state_names)):
39        ax = axs[j%3, int(j/3)]

```

```

38     ax.fill_between(t_pred,  bounds[:, j], -bounds[:, j],
39                      color = colors.get(j%3),
40                      label=f"\u00b13\u03c3 {state_names[j]}",
41                      alpha=0.5)
42     ax.grid(True, alpha=0.3)
43     ax.legend(loc="upper left")
44
45 fig.suptitle("Time [s]")
46 fig.suptitle("Error bound (centered at 0)")
47 fig.suptitle("Prediction covariance bounds (\u00b13\u03c3)")
48 fig.tight_layout()
49
50 return fig

```

./src/helpers/propagators.py

```

1 # libs
2 import numpy as np
3 import scipy as sp
4 import matplotlib.pyplot as plt
5
6 # src
7 from measurements import *
8 from .system import *
9
10 def predict_only(data: measurements.Measurement, X0_plus: np.ndarray, P0_plus: np.ndarray,
11 → σ_a: float, μ: float) -> [list, np.ndarray, np.ndarray]:
12     # extract arrays from data
13     t = np.asarray(data.t, dtype=float)
14
15     # shift epoch so t[0] = 0
16     t0 = t[0]
17     t = t - t0
18
19     # sort by time (for propagation)
20     order = np.argsort(t)
21     t = t[order]
22
23     # prediction storage
24     X_minus_hist = np.zeros((len(t), 6))
25     P_minus_hist = np.zeros((len(t), 6, 6))
26
27     X_plus = X0_plus.copy()
28     P_plus = P0_plus.copy()
29     t_prev = 0.0
30
31     for k, t_k in enumerate(t):
32         dt = t_k - t_prev

```

```

33     if dt > 0:
34         Φθ = np.eye(6)
35         yθ = np.concatenate([X_plus, Φθ.reshape(-1)])
36
37         sol = sp.integrate.solve_ivp(
38             fun=lambda tt, yy: eom_with_stm(tt, yy, μ),
39             t_span=(t_prev, t_k),
40             y0=yθ,
41             t_eval=[t_k],
42             rtol=1e-10,
43             atol=1e-12,
44             method="DOP853",
45         )
46         y_k = sol.y[:, -1]
47         X_minus = y_k[0:6]
48         Φ = y_k[6:].reshape(6,6)
49
50         # prediction-only, so no process update:
51         Q_k = Q_discrete(dt, σ_a)
52         P_minus = Φ @ P_plus @ Φ.T + Q_k
53     else:
54         # multiple measurements at same time tag: no propagation
55         X_minus = X_plus
56         P_minus = P_plus
57
58         X_minus_hist[k, :] = X_minus
59         P_minus_hist[k, :, :] = P_minus
60
61         # prediction-only, so no measurement update:
62         X_plus = X_minus
63         P_plus = P_minus
64         t_prev = t_k
65
66     return t, X_minus_hist, P_minus_hist

```

`./src/helpers/system.py`

```

1 # libs
2 import numpy as np
3 import scipy as sp
4 import matplotlib.pyplot as plt
5
6 # src
7 from measurements import *
8
9 def coe2rv(oe: list, μ:float) -> [np.ndarray, np.ndarray]:
10     # a in km, angles in rad
11     a, e, i, Ω, ω, ν = [float(x) for x in oe]
12

```

```

13 # Semi-latus rectum
14 p = a * (1 - e**2)
15
16 # Perifocal coordinates (PQW frame)
17 r_pf = (p / (1 + e*np.cos(v))) * np.array([np.cos(v), np.sin(v), 0.0])
18 v_pf = np.sqrt(mu/p) * np.array([-np.sin(v), e + np.cos(v), 0.0])
19
20 # Rotation from PQW -> ECI (3-1-3 sequence)
21 R_pqw_eci = sp.spatial.transform.Rotation.from_euler("Zxz", [-omega, -i, -Omega])
22 r = r_pf @ R_pqw_eci.as_matrix()
23 v = v_pf @ R_pqw_eci.as_matrix()
24
25 return r, v
26
27 def A_matrix(r: np.ndarray, mu: float) -> np.ndarray:
28     rnorm = np.linalg.norm(r)
29     I3 = np.eye(3)
30     dadr = -mu * (I3/(rnorm**3) - 3.0*np.outer(r, r)/(rnorm**5))
31     A = np.block([
32         [np.zeros((3,3)), I3],
33         [dadr, np.zeros((3,3))]
34     ])
35
36     return A
37
38 def Q_discrete(dt: float, sigma_a: float) -> np.ndarray:
39     if dt <= 0.0:
40         return np.zeros((6,6))
41
42     q_a = sigma_a**2
43     I3 = np.eye(3)
44
45     Q = q_a * np.block([
46         [(dt**3/3.0)*I3, (dt**2/2.0)*I3],
47         [(dt**2/2.0)*I3, (dt)*I3]
48     ])
49
50     return Q
51
52 def eom_with_stm(t: float, y: list, mu: float) -> np.ndarray:
53     # y = [r(3), v(3), Phi:flat(36)]
54     r = y[0:3]
55     v = y[3:6]
56     phi = y[6:6].reshape(6,6)
57
58     rnorm = np.linalg.norm(r)
59     a = -mu * r / (rnorm**3)
60

```

```
61 |     A = A_matrix(r, μ)
62 |     dΦ = A @ Φ
63 |
64 |     return np.concatenate([v, a, dΦ.reshape(-1)])
```

./src/measurements/measurements.py

```
1 | # libs
2 | from typing import NamedTuple
3 | import numpy as np
4 |
5 | class Measurement(NamedTuple):
6 |     t: np.ndarray
7 |     i: np.ndarray
8 |     p: np.ndarray
9 |     dp: np.ndarray
```

./src/measurements/numpy_data.py

```
1 | # libs
2 | import numpy as np
3 |
4 | # src
5 | from .measurements import *
6 |
7 | def raw_load(file_path: str) -> np.ndarray:
8 |     try:
9 |         raw_data = np.load(file_path, allow_pickle=True)
10 |     except NameError:
11 |         print(f"Could not load data from {file_path}")
12 |
13 |     return raw_data
14 |
15 | def parse(raw_data: np.ndarray) -> Measurement:
16 |     data = Measurement(raw_data[:, 0], raw_data[:, 1], raw_data[:, 2], raw_data[:, 3])
17 |
18 |     return data
19 |
20 | def load(file_path: str) -> Measurement:
21 |     raw_data = raw_load(file_path)
22 |     data = parse(raw_data)
23 |
24 |     return data
```