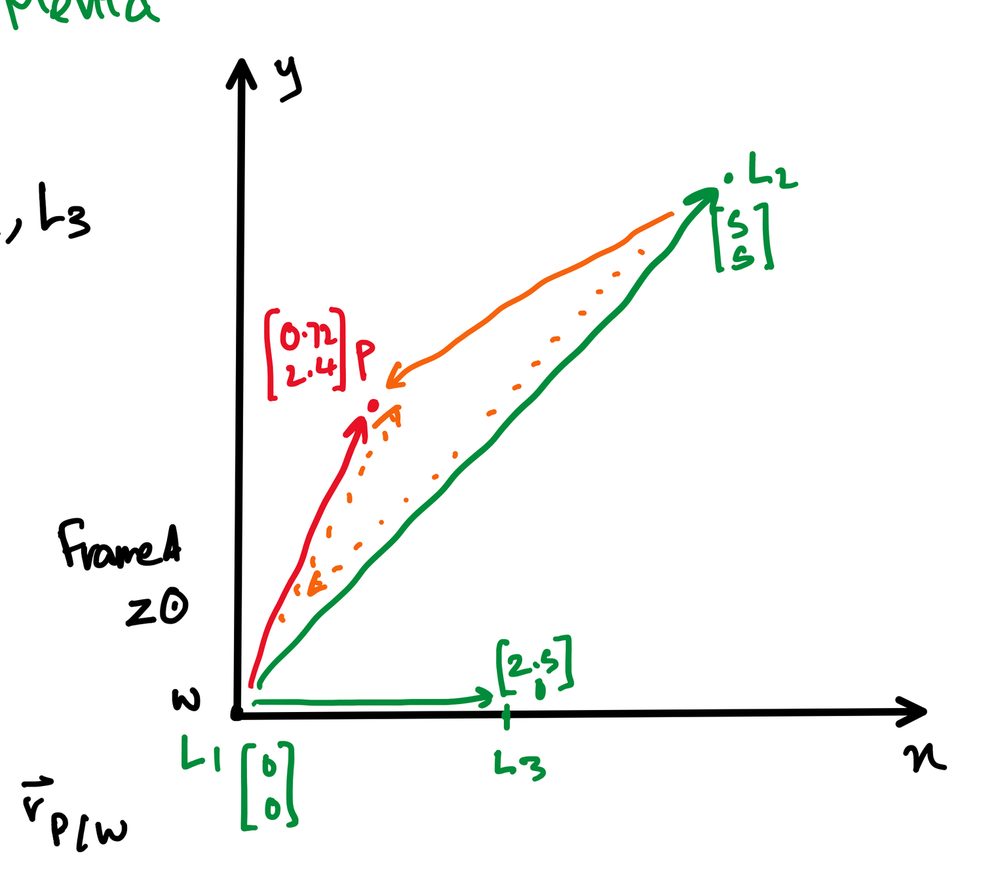
HW4 Akshat Dubey

Worked with Vishwa Mehta

Tuesday, November 12, 2024 5:40 PM



1) · We have a known point w and 3 Light houses L_1, L_2, L_3

- Vectors from w to Lighthouses

$$\vec{r}_{L_1/w} = \begin{bmatrix}0\\0\\0\end{bmatrix}, \quad \vec{r}_{L_2/w} = \begin{bmatrix}5\\5\\0\end{bmatrix}, \quad \vec{r}_{L_3/w} = \begin{bmatrix}2.5\\0\\0\end{bmatrix}$$

- Vector from w to point P : $\vec{r}_{P/w} = \begin{bmatrix}0.7212\\2.4080\\0\end{bmatrix}$

- Vectors from Lighthouses to point P: $\vec{r}_{P/L_i} = -\vec{r}_{L_i/w} + \vec{r}_{P/w}$
- Distance of vectors $r_i = \|\vec{r}_{P/L_i}\|$: $r_1 = 2.5$, $r_2 = 5$, $r_3 = 3$
- Our state X is the position of point P, which stays constant

$$X_{k+1} = X_k$$

- The filter estimates this position as $\hat{X}_k : [\hat{x}_k, \hat{y}_k]^T$, which is estimated $\hat{P}$
- We measure the distances $\hat{r}_i$ wrt $\hat{X}_k = \|\vec{r}_{\hat{P}/L_i}\|$ every second, which are corrupted by noise.

$$Y_k = \begin{bmatrix}\hat{r}_1\\\hat{r}_2\\\hat{r}_3\end{bmatrix} + D w_k, \quad D = \begin{bmatrix}0.1 & 0 & 0\\0 & 0.1 & 0\\0 & 0 & 0.1\end{bmatrix}, \quad w_k \sim \mathcal{N}(0, I_3)$$

$$\Rightarrow Y_k = \begin{bmatrix}\|\vec{r}_{\hat{P}/L_1}\|\\\|\vec{r}_{\hat{P}/L_2}\|\\\|\vec{r}_{\hat{P}/L_3}\|\end{bmatrix} + D w_k$$

$$\Rightarrow Y_k = \begin{bmatrix}\sqrt{(\hat{x}_k - x_1)^2 + (\hat{y}_k - y_1)^2}\\\sqrt{(\hat{x}_k - x_2)^2 + (\hat{y}_k - y_2)^2}\\\sqrt{(\hat{x}_k - x_3)^2 + (\hat{y}_k - y_3)^2}\end{bmatrix} + D w_k$$

→ This is non linear, need to linearize wrt x

x_i, y_i are x,y locations of Lighthouse L_i

1.a) Kalman Filter Prediction equations

- $\hat{X}_{k+1|k} = \hat{X}_k$
- $P_{k+1|k} = P_{k|k}$ (A = identity, Q(model noise) = 0)

Kalman filter Update equations

- $Y_{k+1} = g(X_{k+1}) + D w_k$ where $g(X_k) = \begin{bmatrix}\sqrt{(x_k - x_1)^2 + (y_k - y_1)^2}\\\sqrt{(x_k - x_2)^2 + (y_k - y_2)^2}\\\sqrt{(x_k - x_3)^2 + (y_k - y_3)^2}\end{bmatrix}$

Define $C_{k+1} = \left.\frac{\partial g}{\partial X}\right|_{\hat{X}_{k+1|k}}$

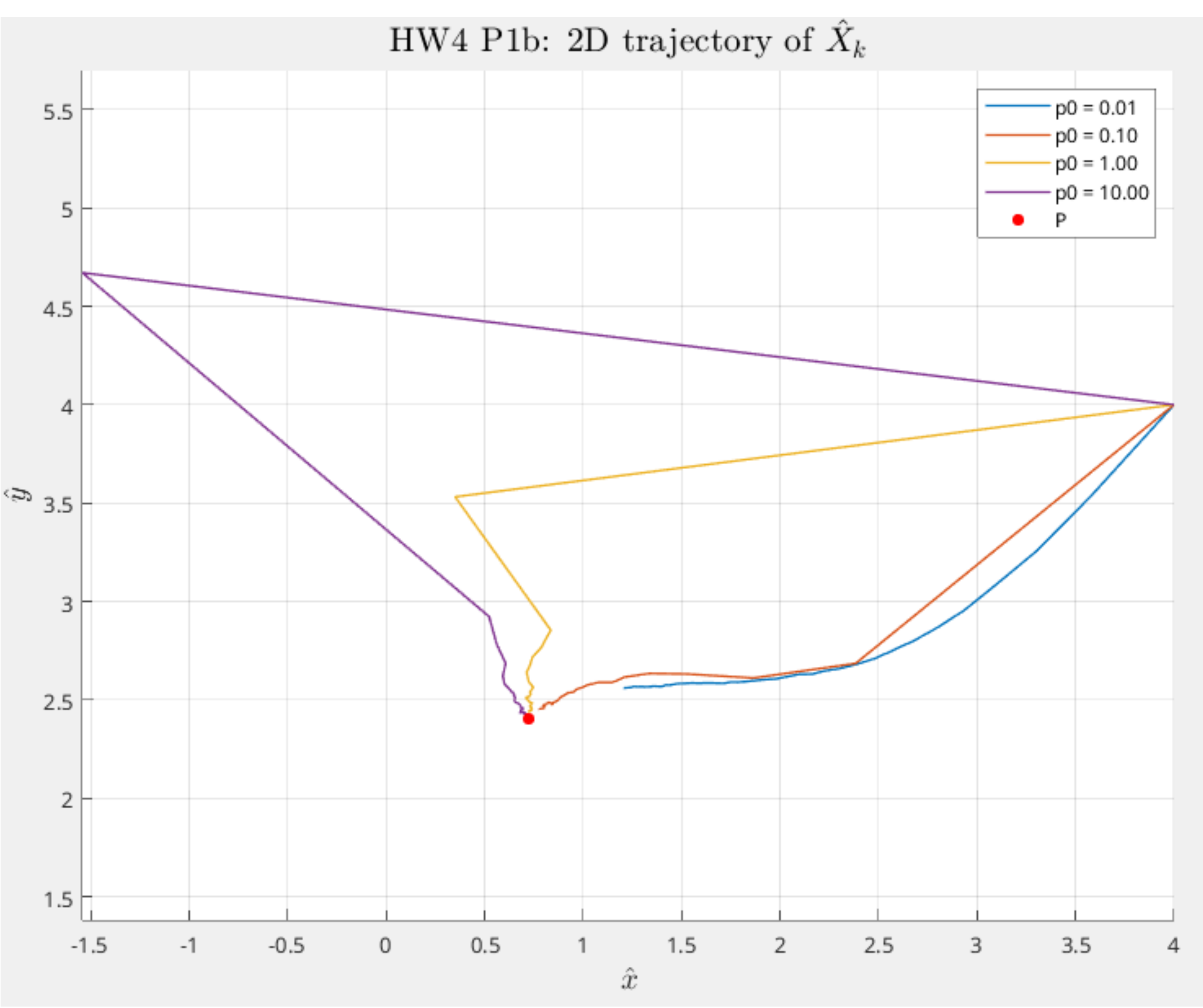
- $K_k = P_{k+1|k} C_{k+1}^T (C_{k+1} P_{k+1|k} C_{k+1}^T + R)^{-1}$ where $R = \begin{bmatrix}0.1 & 0 & 0\\0 & 0.1 & 0\\0 & 0 & 0.1\end{bmatrix}$
- $\hat{X}_{k+1|k+1} = \hat{X}_{k+1|k} + K_k (Y_{k+1} - g(\hat{X}_{k+1|k}))$
- $P_{k+1|k+1} = P_{k+1|k} - K_k C_{k+1} P_{k+1|k}$

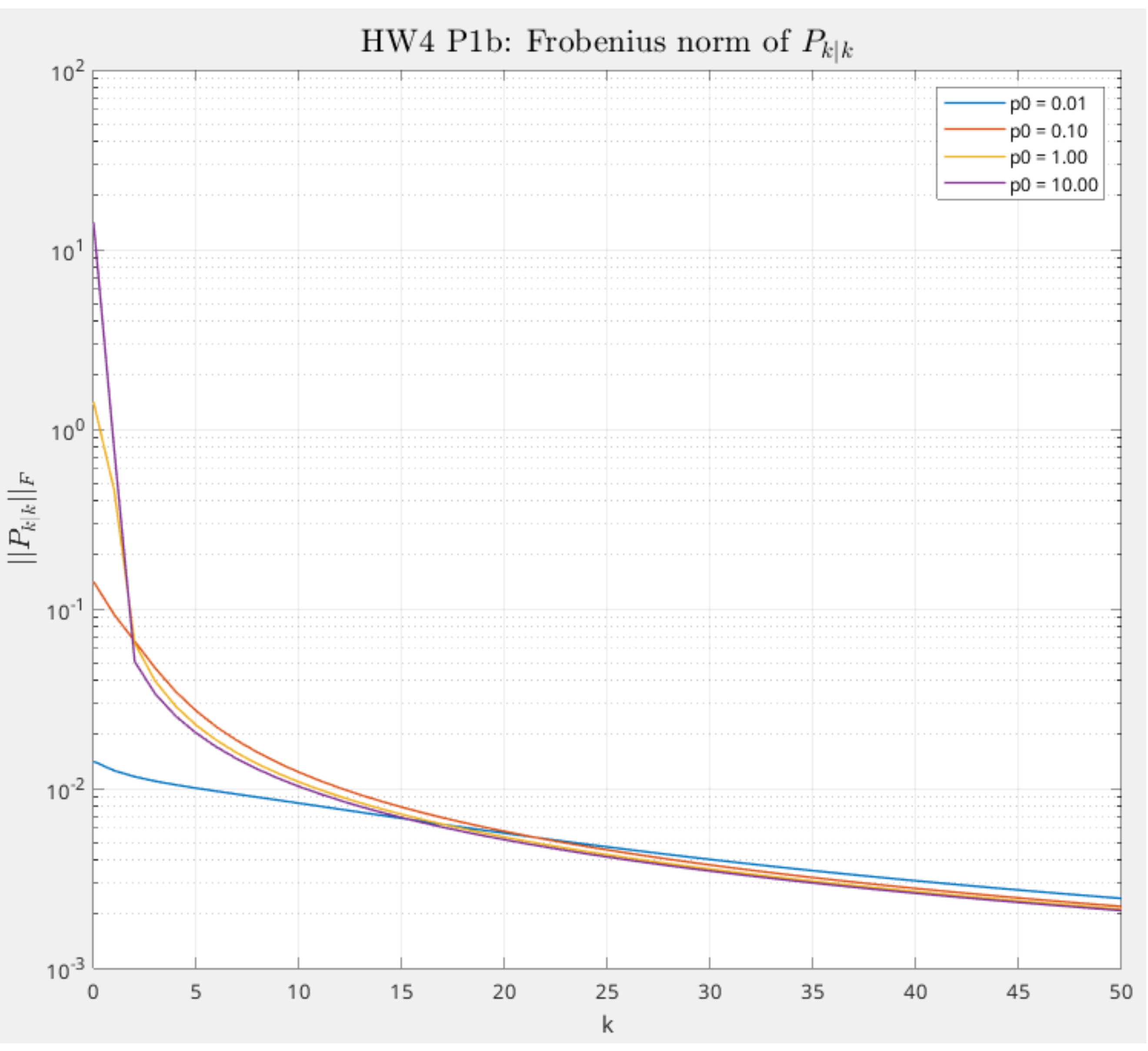
1.b). Model uncertainty : Q = 0

- Measurement Noise : $R = \begin{bmatrix}0.1 & 0 & 0\\0 & 0.1 & 0\\0 & 0 & 0.1\end{bmatrix}$

- Initial estimate $\hat{X}_0 = \begin{bmatrix}4\\4\end{bmatrix}$

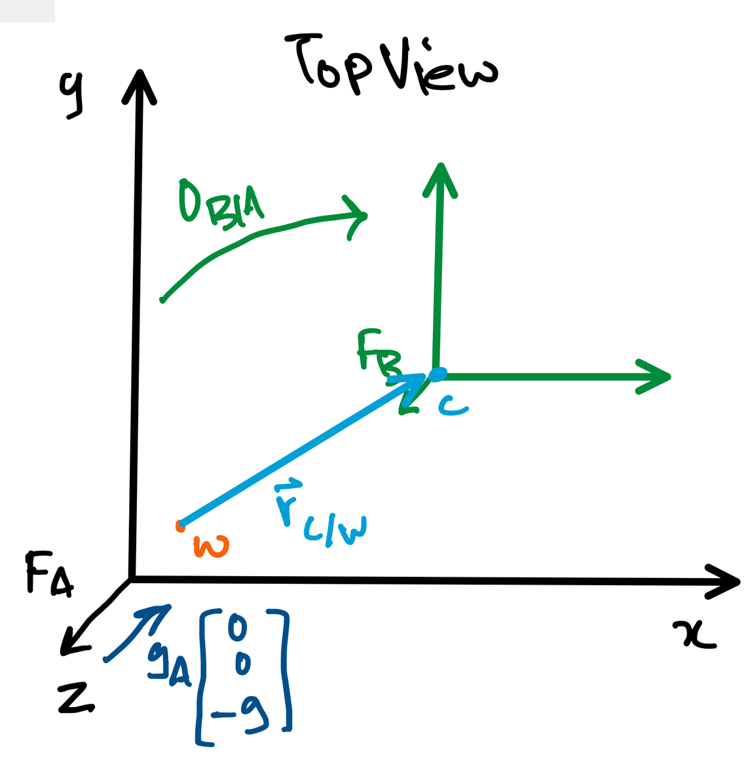
- $P_{0|0} = \begin{bmatrix}p_0 & 0\\0 & p_0\end{bmatrix}$ where $p_0 \in \{0.01, 0.1, 1, 10\}$





2) $g = 9.80665$, $\phi = \pi/6$,

$$g_A = \begin{bmatrix}0\\0\\-g\end{bmatrix}$$



- For drone sensors in Frame B, sampling interval T = 0.01s
- ang vel (gyro): $\omega_k = \begin{bmatrix}0\\0\\1\end{bmatrix} + D_1 w_{1,k}$, $D_1 = \begin{bmatrix}0.1 & 0 & 0\\0 & 0.1 & 0\\0 & 0 & 0.1\end{bmatrix}$, $w_{1,k} \sim \mathcal{N}(0, I_3)$
- acceleration (accel) = $a_k = \begin{bmatrix}-1 - g\sin\phi\sin kT\\-g\sin\phi\cos kT\\-g\cos\phi\end{bmatrix} + D_2 w_{2,k}$, $D_2 = D_1$, $w_{2,k} = w_{1,k}$
- At t=0, $\vec{r}_{c/w|A}(0)$, $\overset{A}{\dot{\vec{r}}}_{c/w|A}(0)$, $O_{B|A}(0)$ are given

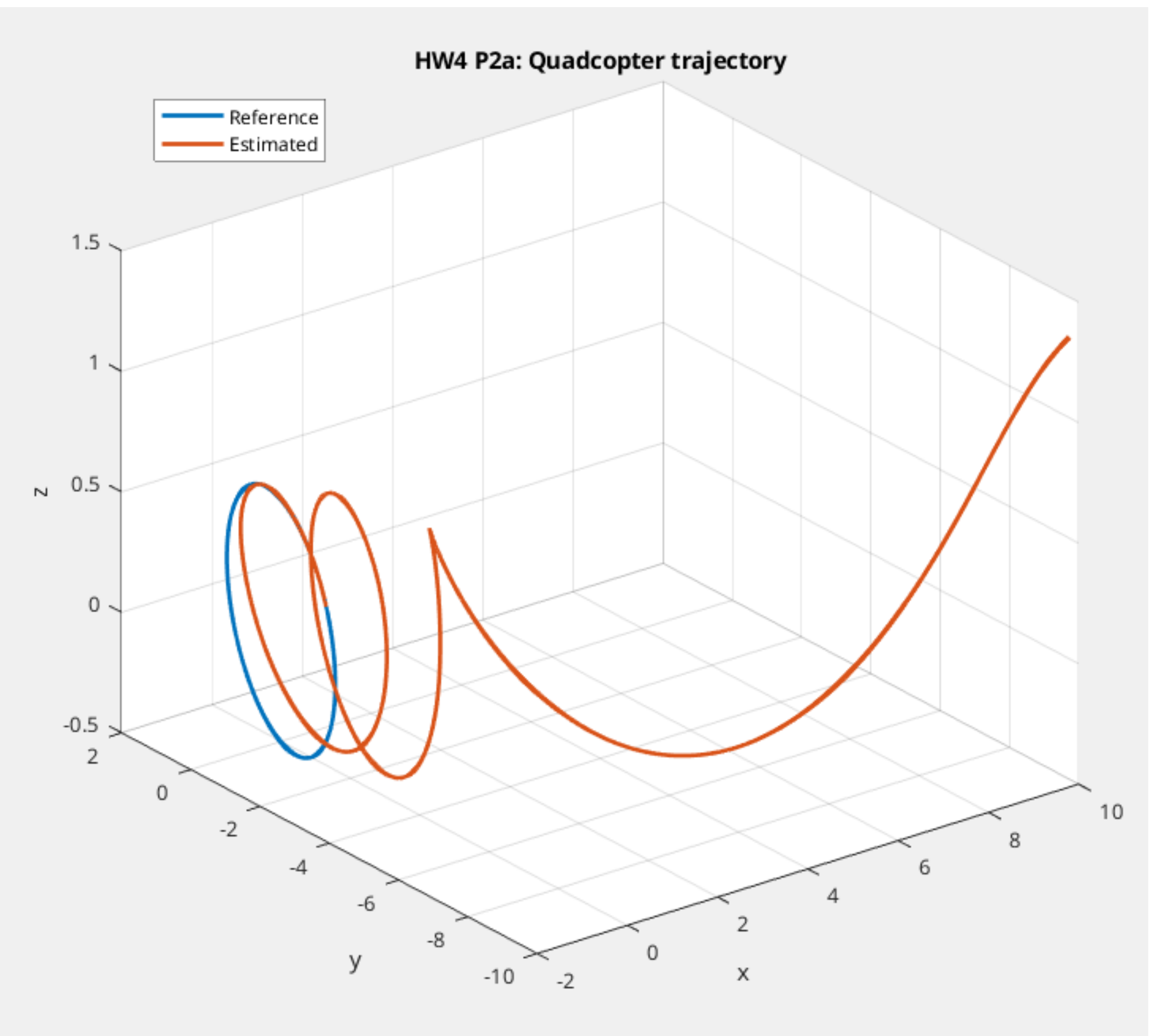
speed vel orient mat

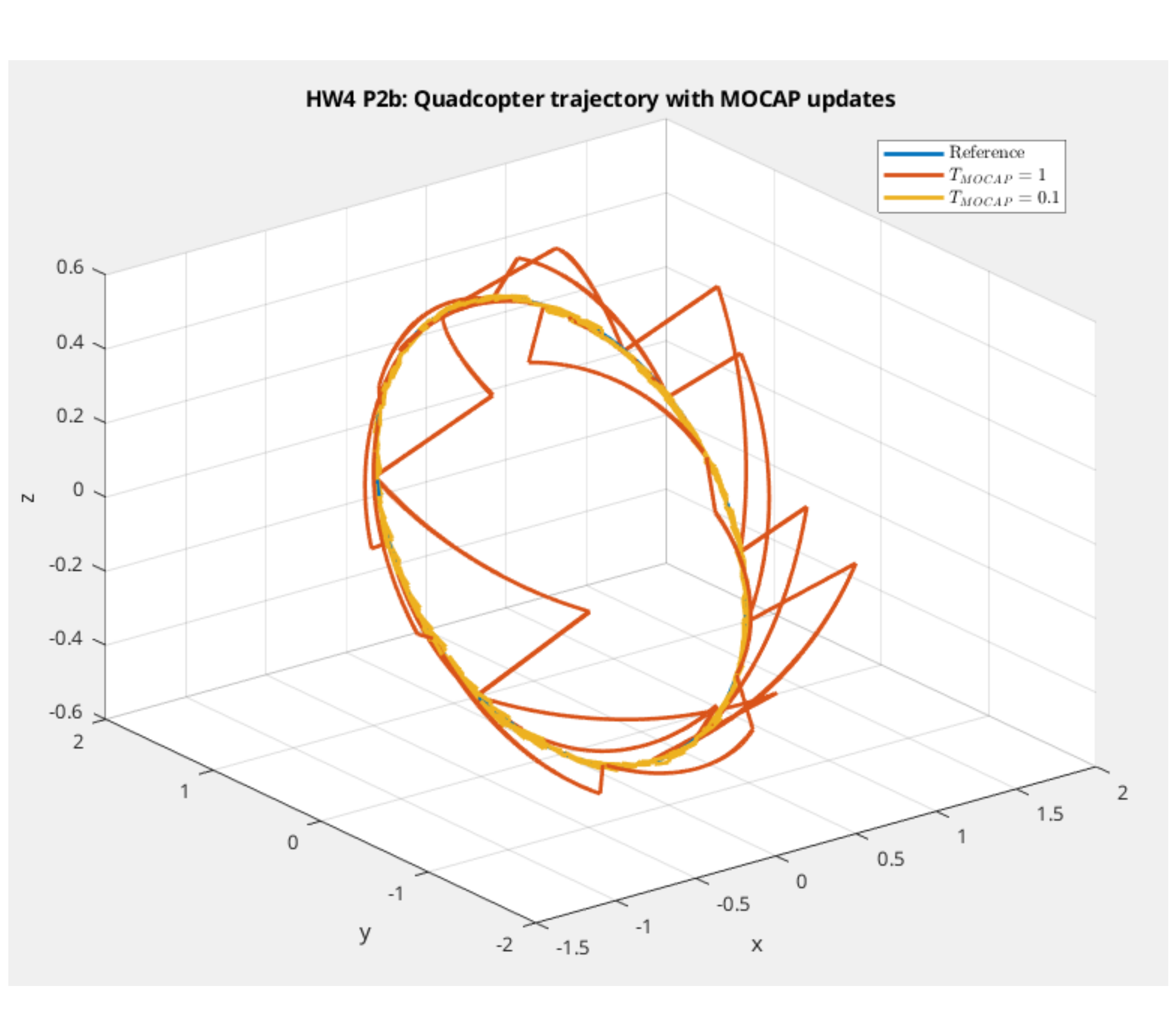
2.a) We can only implement the prediction step, so Kalman Filter eqs are Notes 10, slide 14 slides, 11

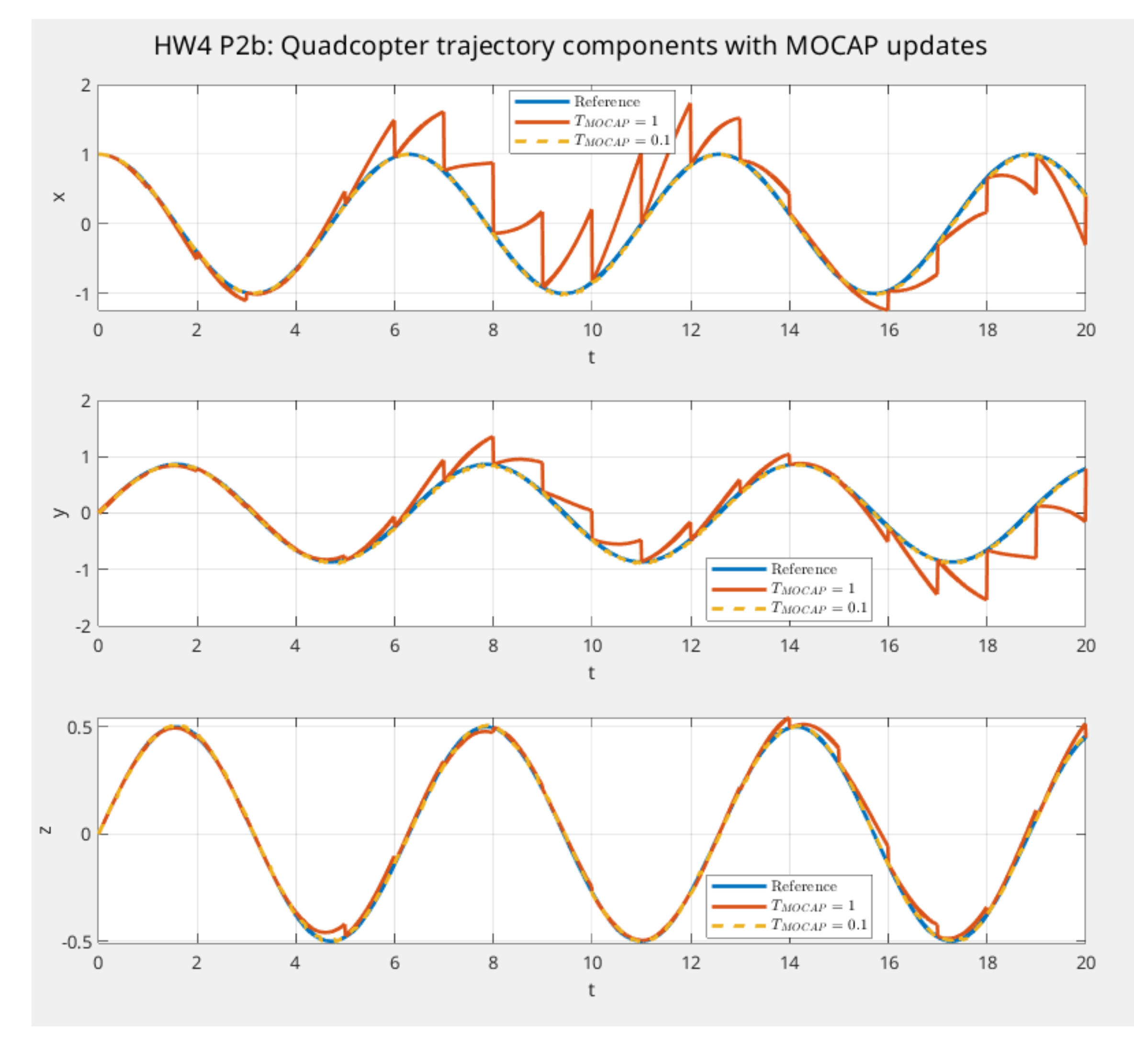
- Gyro: $\hat{x}_{k+1|k} = A_k \hat{x}_{k|k}$, $A_k = \Omega_{d,k}$ with noise in ω_k

$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k}$

- Accel : $\hat{x}_{k+1|k} =$







Problem 3

i Entry Guidance: A Unified Method, Ping Lu

The Main contrbutions of this paper are the development of a unified entry guidance algorithm that can be used for a wide range of entry vehicles. This algorithm is designed to be robust and adaptable, and minimizes the need to tune parameters for different vehicles reducing complexity for designing new missions based on existing vehicle designs as well.

The methodology used is a numerical predictor-corrector algorithm that adjusts bank angles dynamically by iteratively perdicting and correcting trajectory deviations. This algorithm is similar to a model predictive control scheme, but is more robust to changes in external conditions or model mismatch. In case of high Lift-to-Drag ratio vehicles, the algorithm compensates for attitude rate deviations between different phases of entry, ennabling efficient control over heating and structural load factors.

The key findings in this paper are that one, this algorith effectively demonstrates reliable performance across all tested vehicle types and mission scenarios. It works especially well in vehicles with a high lift-to-drag ratio vehicles, achieving a stable, controlled descent. Two, the algorithm is able to manage structural and thermal constraints effectively, ensuring the safety and performance of the vehicle in a high-stress atmospheric reentry scenario.

ii Mid-Lift-to-Drag Ratio Rigid Vehicle Control System Design and Simulation for Human Mars Entry, Breanna Johnson et al.

The main contribution is the The Mid-Lift-to-Drag (Mid-L/D) Rigid Vehicle, MRV's, control system, which combines aerodynamic surfaces and a reaction control system (RCS) to enable stable and precise control during Mars entry. This hybrid approach provides an effective balance between aerodynamic stability and maneuverability in Mars' thin atmosphere, addressing the complex requirements of human Mars landing missions. In addition, instead of traditional parachutes being used for descent, this paper adapts a Supersonic Retro-Propulsion (SRP) system to slow the vehicle down which is useful for landing heavier payloads on Mars given its thin atmosphere.

The methodology used is a 6DOF simulation with Monte Carlo analysis which helps achieve robust control design during the simulation phase under a variety of conditions. This simulation assesses MRV's control performance, with specific attention to variations in RCS jet thrust, vehicle aerodynamic stability, and atmospheric conditions. This allowed for the identification of optimal bank rate limits and effective jet placements to maximize torque while minimizing propellant usage under a variety of conditions.

The key findings are that the MRV's control system demonstrated sufficient stability and accuracy for a precise Mars landing, even under various atmospheric dispersions. The combined aerosurface and RCS control structure proved effective for managing vehicle orientation with minimal fuel consumption while carrying a heavy load. This paper also found that Supersonic Retro-Propulsion (SRP)s are feasible and effective for controlled descent in Mars's low-density atmosphere, supporting the MRV's ability to deliver large payloads while maintaining landing precision.

iii Pterodactyl: Development and Performance of Guidance Algorithms for a Mechanically Deployed Entry Vehicle, Breanna Johnson et al

The main contributions here are specific for the Pterodactyl vehicle, which is a mechanically deployed entry vehicle (DEV) designed for Mars missions. These DEVs lack the rigid structure of traditional capsules, and offer more flexibility and potential efficiency in packing and deploying payloads for Mars or lunar missions. This paper also delves into the integration of a fully numerical predictor-corrector guidance algorithm for the Pterodactyl vehicle, which provides precise control over the vehicle's entry trajectory, including bank angle adjustments for configurations requiring significant lateral maneuvering.

For other configurations, an uncoupled range control (URC) was added to provide guidance through adjustments in angle of attack and sideslip angle. This paper also has extensive 6-DOF simulations to validate the guidance algorithms.

The methodology used in this paper is a combination of numerical predictor-corrector guidance algorithms and 6-DOF simulations with the use of Monte carlo analysis. This provides robust optimization of the controller which is essential for the Pterodactyl vehicle given its unique structure and mission profile. The guidance algorithms also takes into account the structural, aerodynamic, and thermal performance of the vehicle to ensure in-flight stablity. They use a custom FNPEG for bank angle control, and developed URC for vehicles requiring angle of attack and sideslip guidance.

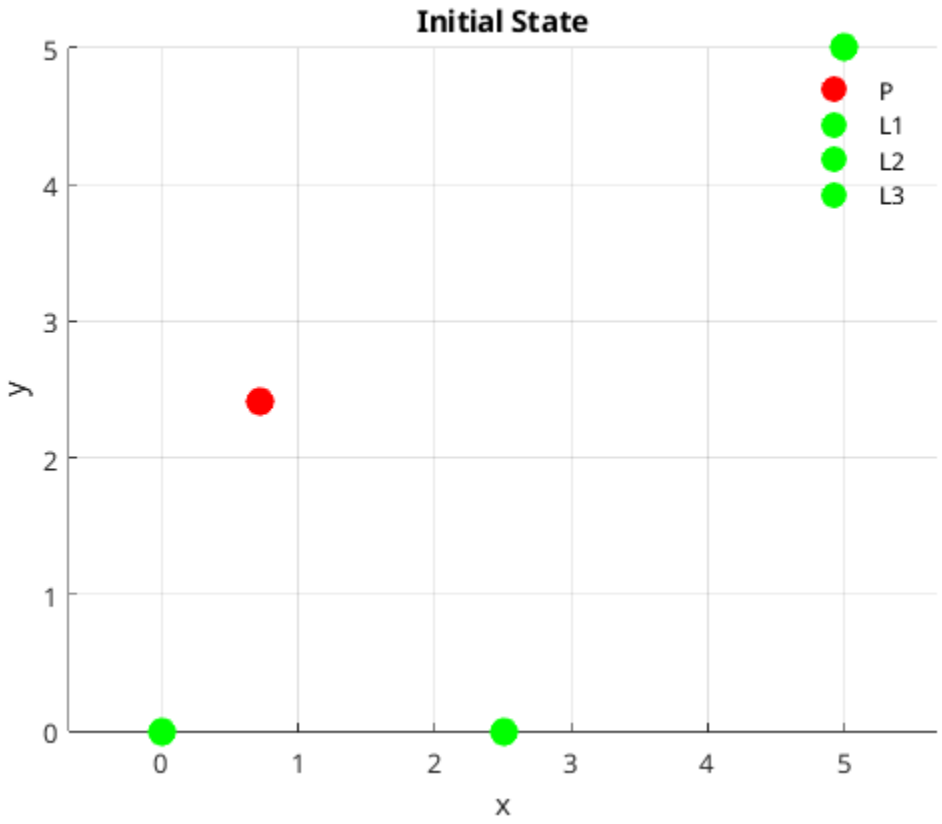
They key findings are that the GNC algorithm was able to achieve precise control over the Pterodactyl vehicle's landings under the contraints of miss distance, heat rate, and g-loads. It was also able to minimize fuel use by mainly using aerodynamic adjustments instead of RCS thrusters, which is especially valuable for future interplanetary missions with limited fuel on-board. Finally, this paper demonstrated the scalability of the GNC algorithm for future missions, showing that it can suppert a wide range of planetary missions with different vehicle configurations and mission profiles.

iv Question for Breanna Johnson

How modular are the models used for atmosphere/gravity for different missions? Do you develop a new model for each mission based on the vehicle and mission profile or reuse parts of existing models used in previous missions? What I mean is, when doing simulations for a new mission with a new vehicle, how much of the vehicle's dynamics are made from scrach and how much is reused from previous missions?

```
clc; clear; close all;

% setup given values without z
r_Pw = [0.7212; 2.4080];
r_L1w = [0; 0];
r_L2w = [5; 5];
r_L3w = [2.5; 0];

plot_init_state(r_Pw, r_L1w, r_L2w, r_L3w);
r1 = norm(r_L1w - r_Pw);
r2 = norm(r_L2w - r_Pw);
r3 = norm(r_L3w - r_Pw);

% setup jacobian for C
syms x y;
X = [x; y];
f = Gx(X, r_L1w, r_L2w, r_L3w);
C = jacobian(f, X);

% setup simulation
ks = 50;
p0s = [0.01, 0.1, 1, 10];

x0 = r_Pw;
xhat0 = [4;4];

data.xhat = zeros(2, ks+1, length(p0s));
data.Pnorm = zeros(ks+1, length(p0s));

for i = 1:length(p0s)
    P0 = p0s(i) * eye(2);
    x_k = x0;
    xhat_k = xhat0;
    P_k = P0;
    data.xhat(:, 1, i) = xhat_k;
    data.Pnorm(1, i) = norm(P_k, "fro");
    for k = 1:ks
        % predict
        xhat_kp1 = ekf_predict(xhat_k, P_k);

        % propagate real system
        x_kp1 = x_k;
        % measure wtr true pos
        y_kp1 = measure_noisy(x_kp1, r_L1w, r_L2w, r_L3w);

        % update
        C_kp1 = vpa(subs(C, X, xhat_kp1)); % get C wrt xhat
        [xhat_kp1, P_k] = ekf_update(xhat_k, P_k, y_kp1, C_kp1, r_L1w, r_L2w,
r_L3w);

        % store data
        data.xhat(:, k+1, i) = xhat_kp1;
```

```
        data.Pnorm(k+1, i) = norm(P_k, "fro");

        % update vars for next iteration
        x_k = x_kp1;
        xhat_k = xhat_kp1;
    end
end
```

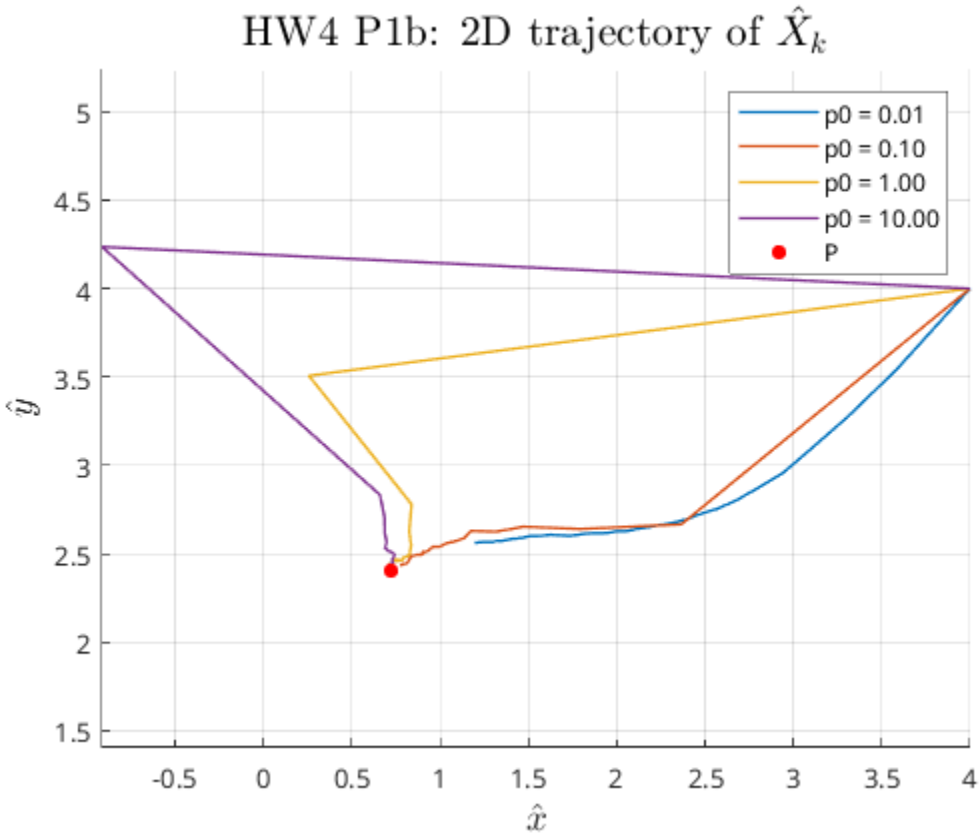


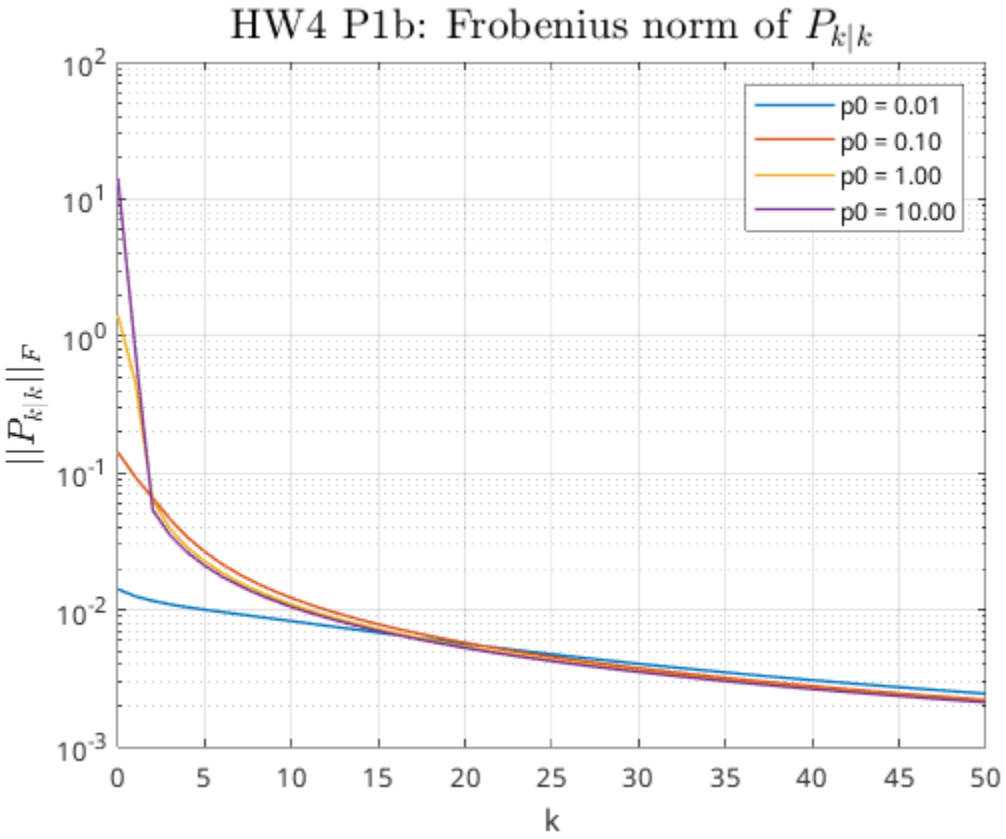
plot

```
plot_xhat(data, p0s, r_Pw);
plot_Pnorm(data, p0s);

function y = Gx(x, L1, L2, L3)
y = [
    norm(x - L1);
    norm(x - L2);
    norm(x - L3);
    ];
end

function Yk = measure_noisy(x, L1, L2, L3)
D = diag([0.1, 0.1, 0.1]);
wk = mvnrnd([0; 0; 0], eye(3))';
Yk = Gx(x, L1, L2, L3) + D*wk;
end
```

```
function [xhat_p, P_p] = ekf_predict(xhat, P)
xhat_p = xhat;
P_p = P;
end

function [xhat_u, P_u] = ekf_update(xhat_p, P_p, Yk, Ck, L1, L2, L3)
R = 0.1*eye(3);
K_k = P_p * Ck' * inv(Ck * P_p * Ck' + R);
xhat_u = xhat_p + K_k * (Yk - Gx(xhat_p, L1, L2, L3));
P_u = P_p - K_k * Ck * P_p;
end

function plot_init_state(r_Pw, r_L1w, r_L2w, r_L3w)
figure;
hold on;
plot(r_Pw(1), r_Pw(2), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r',
'DisplayName', 'P');
plot(r_L1w(1), r_L1w(2), 'go', 'MarkerSize', 10, 'MarkerFaceColor', 'g',
'DisplayName', 'L1');
plot(r_L2w(1), r_L2w(2), 'go', 'MarkerSize', 10, 'MarkerFaceColor', 'g',
'DisplayName', 'L2');
plot(r_L3w(1), r_L3w(2), 'go', 'MarkerSize', 10, 'MarkerFaceColor', 'g',
'DisplayName', 'L3');
xlabel('x');
ylabel('y');
title('Initial State');
legend boxoff; grid on;
axis equal;
end

function plot_xhat(data, p0s, r_Pw)
figure;
hold on;
for i = 1:length(p0s)
    plot(data.xhat(1, :, i), data.xhat(2, :, i), 'DisplayName', sprintf('p0 =
%.2f', p0s(i)), 'LineWidth', 1);
end
plot(r_Pw(1), r_Pw(2), 'ro', 'MarkerSize', 5, 'MarkerFaceColor', 'r',
'DisplayName', 'P');

title('HW4 P1b: 2D trajectory of $\hat{X}_k$', 'Interpreter', 'latex',
'FontSize', 16);
xlabel("$\hat{x}$", 'Interpreter', 'latex', 'FontSize', 14);
ylabel("$\hat{y}$", 'Interpreter', 'latex', 'FontSize', 14);
legend; grid on;
axis equal;
end

function plot_Pnorm(data, p0s)
figure;
for i = 1:size(data.Pnorm, 2)
    semilogy(0:size(data.Pnorm, 1)-1, data.Pnorm(:, i), 'DisplayName',
sprintf('p0 = %.2f', p0s(i)), 'LineWidth', 1);
```

```
    hold on;
end
hold off;

title('HW4 P1b: Frobenius norm of $P_{k|k}$', 'Interpreter', 'latex',
'FontSize', 16);
xlabel('k');
ylabel('$||P_{k|k}||_F$', 'Interpreter', 'latex', 'FontSize', 14);
legend; grid on;
end
```





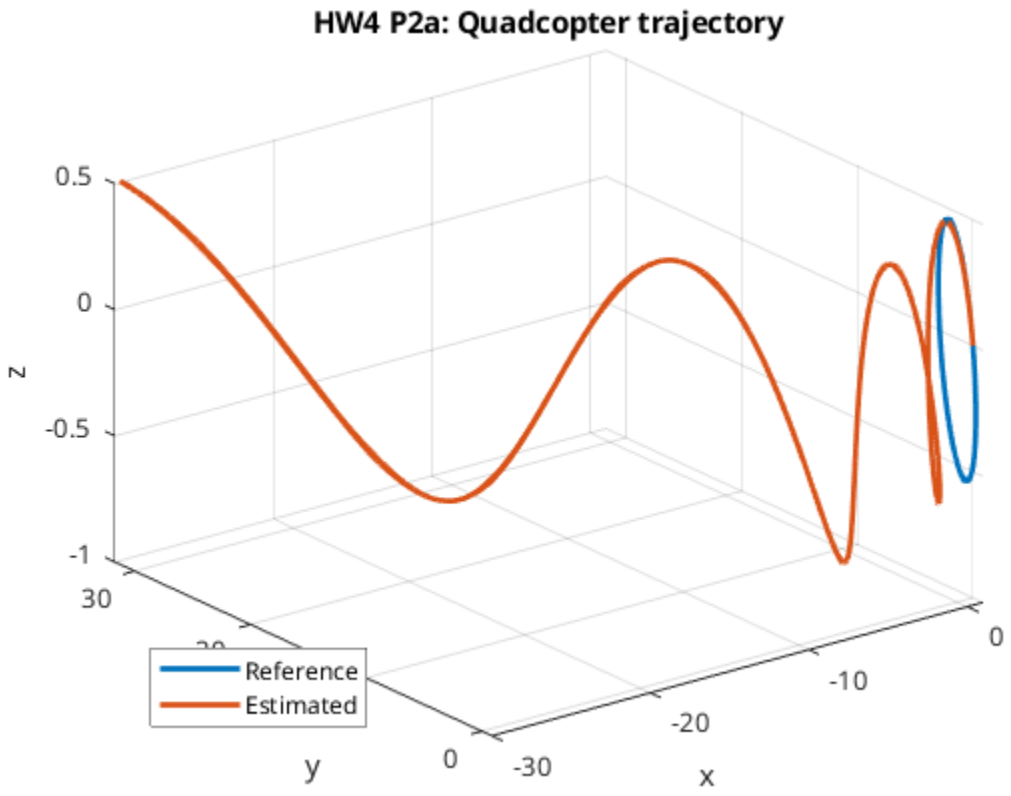
Published with MATLAB® R2023b

```
clc; clear; close all;

% consts
g = 9.80665; % m/s^2
phi = pi/6; % rad
g_A = [0;0;-g]; % m/s^2
T = 0.01; % s, sampling interval
D = diag([0.1,0.1,0.1]); % rad^2/s^2, noise covariance matrix
% D = diag([0,0,0]); % rad^2/s^2, noise covariance matrix
noise = @() D*mvnrnd(zeros(3,1), eye(3))'; % noise function

% sensors
omega_k_noisy = @() [0;0;1] + noise(); % rad/s
a_k_noisy = @(k) [
    -1-g*sin(phi)*sin(k*T);
    -g*sin(phi)*cos(k*T);
    -g*cos(phi);
    ] + noise();

% inital conditions
r_0 = [1;0;0]; % initial position
rdot_0 = [0;cos(phi);sin(phi)]; % initial angular velocity
omat_BA_0 = [
    1, 0,           0;
    0, cos(phi),  sin(phi);
    0, -sin(phi), cos(phi);
    ];

ks = 2000;
x_0 = [r_0; rdot_0]; % pos + accel state
y_0 = omat_BA_0; % as per lecture 16 slide 5, gyro state
```

part a, just predict

```
data.x = zeros(6,ks+1);
data.x(:,1) = x_0;
x_k = x_0;
y_k = y_0;

for k=1:ks
    % get sensor data
    omega_k = omega_k_noisy();
    a_k = a_k_noisy(k);

    % get dynamics
    [Ad, Bd, OMEGAd] = get_discrete_sys(T, omega_k);

    % propagate gyro
    y_kp1 = OMEGAd*y_k;
    % propagate accelerometer
    x_kp1 = Ad*x_k + Bd*(y_k'*a_k - g_A);
```

```
    % store data
    data.x(:,k+1) = x_kp1;

    % prep for next iteration
    x_k = x_kp1;
    y_k = y_kp1;
end

load('rcwA.mat');
plot_trajectory2(rcwA_Ts_0_01, data.x, 'Estimated', 'HW4 P2a: Quadcopter
trajectory');
```



part b, predict and update

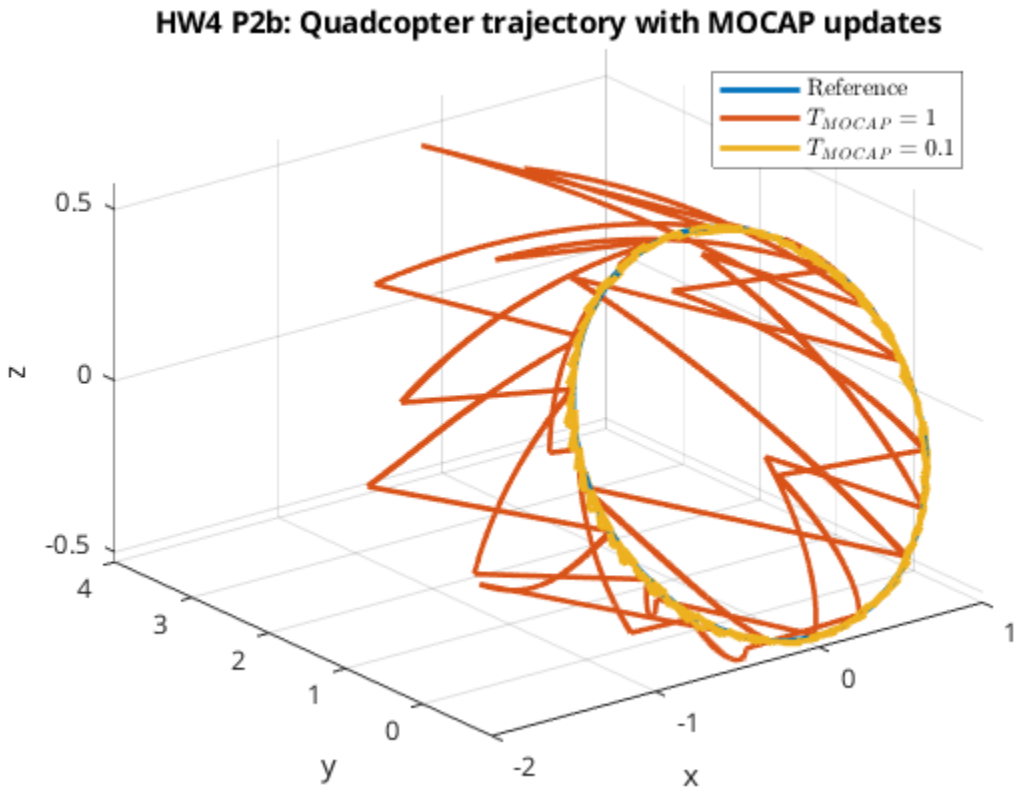
```
D3 = diag([0.005, 0.005, 0.005]);
r_noisy = zeros(3,ks+1);
for i=1:ks+1
    r_noisy(:,i) = rcwA_Ts_0_01(:, i) + D3*mvnrnd(zeros(3,1), eye(3))';
end
R = 0.001*eye(3); Q = 10*eye(6);
P_0 = 10*eye(6);
C = [eye(3), zeros(3)];

data.x_01 = zeros(6,ks+1);
data.x_01(:,1) = x_0;
data.x_1 = zeros(6,ks+1);
```

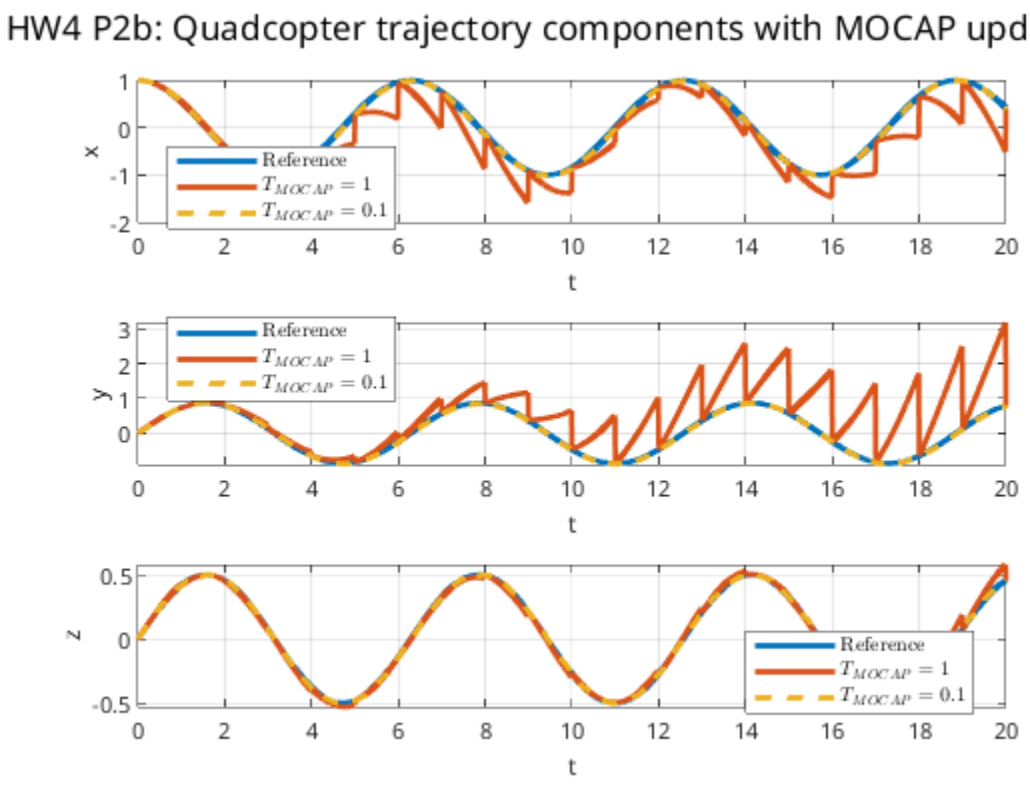
```
data.x_1(:,1) = x_0;

% for Tmocap = 1
x_k = x_0;
y_k = y_0;
P_k = P_0;
modT = 1/T;
for k=1:ks
    % get sensor data
    omega_k = omega_k_noisy();
    a_k = a_k_noisy(k);

    % get dynamics
    [Ad, Bd, OMEGAd] = get_discrete_sys(T, omega_k);

    % predict
    y_kp1 = OMEGAd*y_k;
    x_kp1_p = Ad*x_k + Bd*(y_k'*a_k - g_A);

    if mod(k,modT) == 0
        % Get mocap data
        r_k = r_noisy(:,k);
        % covariance stuff
        P_kp1_p = Ad*P_k*Ad' + Q;
        K_k = P_kp1_p*C'/(C*P_kp1_p*C' + R);
        P_k = P_kp1_p - K_k*C*P_kp1_p;

        % update
        x_kp1 = x_kp1_p + K_k*(r_k - C*x_kp1_p);
    else
        x_kp1 = x_kp1_p;
    end

    % store data
    data.x_1(:,k+1) = x_kp1;

    % prep for next iteration
    x_k = x_kp1;
    y_k = y_kp1;
end

% for Tmocap = 0.1
x_k = x_0;
y_k = y_0;
P_k = P_0;
modT = 0.1/T;
for k=1:ks
    % get sensor data
    omega_k = omega_k_noisy();
    a_k = a_k_noisy(k);

    % get dynamics
    [Ad, Bd, OMEGAd] = get_discrete_sys(T, omega_k);
```

```
    % predict
    y_kp1 = OMEGAd*y_k;
    x_kp1_p = Ad*x_k + Bd*(y_k'*a_k - g_A);

    if mod(k,modT) == 0
        % Get mocap data
        r_k = r_noisy(:,k);
        % covariance stuff
        P_kp1_p = Ad*P_k*Ad' + Q;
        K_k = P_kp1_p*C'/(C*P_kp1_p*C' + R);
        P_k = P_kp1_p - K_k*C*P_kp1_p;

        % update
        x_kp1 = x_kp1_p + K_k*(r_k - C*x_kp1_p);
    else
        x_kp1 = x_kp1_p;
    end

    % store data
    data.x_01(:,k+1) = x_kp1;

    % prep for next iteration
    x_k = x_kp1;
    y_k = y_kp1;
end

plot_trajectory3(rcwA_Ts_0_01, data.x_1, data.x_01,
'$T_{MOCAP}=1$','$T_{MOCAP}=0.1$', 'HW4 P2b: Quadcopter trajectory with MOCAP
updates');
plot_components(rcwA_Ts_0_01, data.x_1, data.x_01,
'$T_{MOCAP}=1$','$T_{MOCAP}=0.1$', T, 'HW4 P2b: Quadcopter trajectory
components with MOCAP updates');

function crossMat = crMat(X)
crossMat = [
    0      -X(3)  X(2);
    X(3)    0    -X(1);
    -X(2)  X(1)   0;
    ];
end

function [Ad, Bd, OMEGAd] = get_discrete_sys(T, omega_k)
Ad = [
    eye(3), T*eye(3);
    zeros(3), eye(3)
    ];
Bd = [
    0.5*T^2*eye(3);
    T*eye(3)
    ];
nhat_k = omega_k / norm(omega_k);
OMEGAd = expm(-norm(omega_k)*T*crMat(nhat_k));
end
```

```
function plot_trajectory2(ref, data, label, title_str)
figure;
plot3(ref(1,:), ref(2,:), ref(3,:), 'LineWidth', 2, 'DisplayName',
'Reference');
hold on;
plot3(data(1,:), data(2,:), data(3,:), 'LineWidth', 2, 'DisplayName', label);
xlabel('x');
ylabel('y');
zlabel('z');
title(title_str);
grid on;
legend("Location", "best");
end

function plot_trajectory3(ref, data1, data2, label1, label2, title_str)
figure;
plot3(ref(1,:), ref(2,:), ref(3,:), 'LineWidth', 2, 'DisplayName',
'Reference');
hold on;
plot3(data1(1,:), data1(2,:), data1(3,:), 'LineWidth', 2, 'DisplayName',
label1);
plot3(data2(1,:), data2(2,:), data2(3,:), 'LineWidth', 2, 'DisplayName',
label2);
xlabel('x');
ylabel('y');
zlabel('z');
title(title_str);
grid on;
legend("Location", "best", "Interpreter", "latex");
end

function plot_components(ref, data1, data2, label1, label2, T, title_str)
figure;
sgtitle(title_str);

subplot(3,1,1);
plot((0:length(ref)-1) * T, ref(1,:) , 'LineWidth', 2, 'DisplayName',
'Reference');
hold on;
plot((0:length(ref)-1) * T, data1(1,:) , 'LineWidth', 2, 'DisplayName',
label1);
plot((0:length(ref)-1) * T, data2(1,:) , '--', 'LineWidth', 2, 'DisplayName',
label2);
xlabel('t');
ylabel('x');
grid on;
legend("Location", "best", "Interpreter", "latex");

subplot(3,1,2);
plot((0:length(ref)-1) * T, ref(2,:) , 'LineWidth', 2, 'DisplayName',
'Reference');
hold on;
plot((0:length(ref)-1) * T, data1(2,:) , 'LineWidth', 2, 'DisplayName',
label1);
```

```
plot((0:length(ref)-1) * T, data2(2,:) , '--', 'LineWidth', 2, 'DisplayName',
label2);
xlabel('t');
ylabel('y');
grid on;
legend("Location", "best", "Interpreter", "latex");

subplot(3,1,3);
plot((0:length(ref)-1) * T, ref(3,:) , 'LineWidth', 2, 'DisplayName',
'Reference');
hold on;
plot((0:length(ref)-1) * T, data1(3,:) , 'LineWidth', 2, 'DisplayName',
label1);
plot((0:length(ref)-1) * T, data2(3,:) , '--', 'LineWidth', 2, 'DisplayName',
label2);
xlabel('t');
ylabel('z');
grid on;
legend("Location", "best", "Interpreter", "latex");

end
```





Published with MATLAB® R2023b