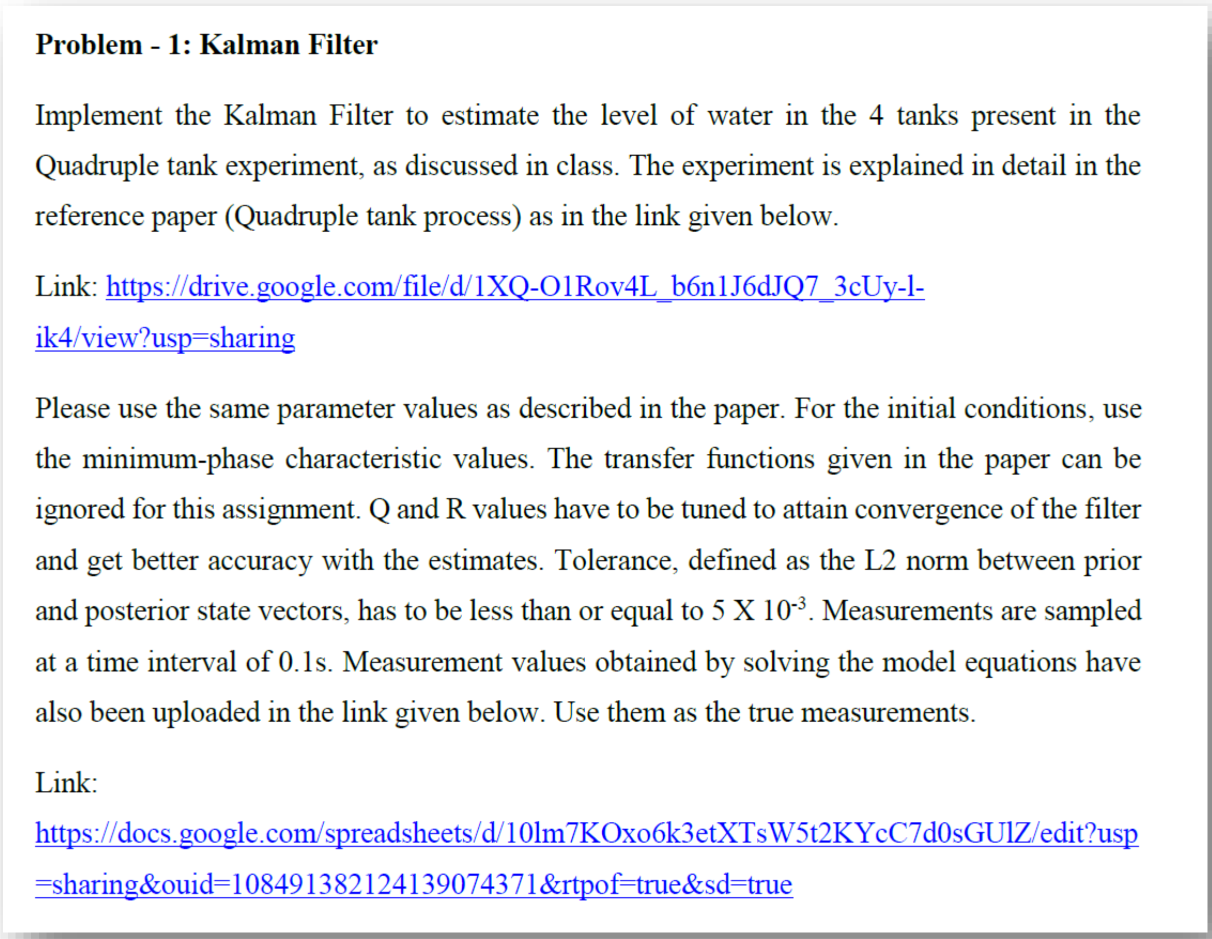
Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503



Problem - 1: Kalman Filter

Implement the Kalman Filter to estimate the level of water in the 4 tanks present in the Quadruple tank experiment, as discussed in class. The experiment is explained in detail in the reference paper (Quadruple tank process) as in the link given below.

Link: https://drive.google.com/file/d/1XQ-O1Rov4L_b6n1J6dJQ7_3cUy-l-ik4/view?usp=sharing

Please use the same parameter values as described in the paper. For the initial conditions, use the minimum-phase characteristic values. The transfer functions given in the paper can be ignored for this assignment. Q and R values have to be tuned to attain convergence of the filter and get better accuracy with the estimates. Tolerance, defined as the L2 norm between prior and posterior state vectors, has to be less than or equal to 5 X 10^{-3}. Measurements are sampled at a time interval of 0.1s. Measurement values obtained by solving the model equations have also been uploaded in the link given below. Use them as the true measurements.

Link:
https://docs.google.com/spreadsheets/d/10lm7KOxo6k3etXTsW5t2KYcC7d0sGUlZ/edit?usp=sharing&ouid=108491382124139074371&rtpof=true&sd=true

Introduction:

The Kalman Filter is a widely used algorithm in state estimation and data fusion applications. Its ability to combine noisy measurements with dynamic system models makes it a powerful tool in fields such as robotics, navigation, signal processing, and control systems. With the increasing availability of high-level programming languages like Python and the rich ecosystem of scientific libraries it offers, implementing the Kalman Filter has become more accessible and efficient.

This report aims to explore the implementation of the Kalman Filter using Python based on research paper titled **Kalman Filter Based on The Quadruple-Tank Process: A Multivariable Laboratory Process with an Adjustable Zero by**

Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503

Karl Henrik Johansson and investigate its performance in various scenarios. The primary objective is to develop a robust and accurate implementation that leverages the computational capabilities and ease of use of Python programming.

The need for accurate state estimation in dynamic systems is crucial for numerous real-world applications. The Kalman Filter offers a solution to this challenge by providing an optimal recursive estimation of the system's state based on noisy measurements and prior knowledge of system dynamics. By incorporating the principles of Bayesian estimation, the Kalman Filter minimizes the impact of measurement noise and system uncertainties, resulting in improved state estimation.

Python, with its simplicity, readability, and extensive libraries such as NumPy and Matplotlib, provides an ideal environment for implementing the Kalman Filter. This report harnesses the power of Python to develop a flexible and efficient implementation of the Kalman Filter, enabling easy adaptation to various domains and applications.

The report will cover the implementation details of the Kalman Filter, including state initialization, prediction, measurement update, and estimation. Various aspects, such as handling non-linear systems, incorporating process noise covariance, and addressing data association challenges, will be explored. Furthermore, the report aims to evaluate the performance of the implemented Kalman Filter by comparing it with existing approaches and assessing its accuracy, convergence, and computational efficiency.

The findings of this analysis can contribute to the advancement of state estimation techniques and provide valuable. The practical implementation using Python can serve as a resource for those seeking to implement the Kalman Filter in their own applications, facilitating faster and more reliable state estimation.

Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503

Reference Literature Overview:

Karl Henrik Johansson's work on the Kalman Filter implementation for the Quadruple-Tank Process provides valuable insights into the application of the Kalman Filter in a multivariable laboratory process with an adjustable zero. This work, published in http://www.diva-portal.org/smash/get/diva2:495784/FULLTEXT01.pdf
, has been influential in the field of control systems and has contributed to the understanding of the Kalman Filter's effectiveness in state estimation and control.

The Quadruple-Tank Process is a well-known benchmark system that exhibits complex dynamics, including interdependencies between tanks and the presence of an adjustable zero. His study focuses on the application of the Kalman Filter to estimate the tank levels accurately in the presence of measurement noise and system uncertainties.

His work emphasizes the importance of considering the unique characteristics of the Quadruple-Tank Process when designing the Kalman Filter. By analysing the process dynamics and noise sources, Johansson derives a suitable state-space model for the system. The derived model incorporates the interdependencies between the tanks and the adjustable zero, enabling accurate state estimation.

One significant contribution of his work is the incorporation of process noise covariance and measurement noise covariance matrices into the Kalman Filter formulation. By properly characterizing these covariance matrices, Johansson demonstrates improved convergence and estimation accuracy for the Quadruple-Tank Process.

Moreover, Johansson investigates the impact of different noise sources and their effects on the estimation process. He highlights the significance of properly modelling and accounting for measurement noise, process noise, and system uncertainties. Through experimental validation, Johansson demonstrates the effectiveness of the Kalman Filter in handling these noise sources and accurately estimating the tank levels.

His work also discusses the implications of system identification and model uncertainty on the Kalman Filter performance. He emphasizes the need for adaptive and robust filtering approaches to handle uncertainties in the system model. By incorporating adaptive techniques, Johansson showcases the ability to adapt the Kalman Filter to variations in the system parameters and achieve accurate state estimation.

Methodology:

This method begins by analysing the dynamics of the Quadruple-Tank Process and identifying the interdependencies between the tanks. Then formulates a suitable state-space model that captures the relationships between the tank levels, input flows, and the adjustable zero. The state-space model serves as the foundation for the Kalman Filter implementation.

The Kalman Filter is a recursive estimation algorithm that combines measurements with predictions based on a mathematical model of the system. In this study, formulation of the Kalman Filter is done using the state-space model of the Quadruple-Tank Process. The filter consists of two main steps:

- Prediction and
- Update

In the prediction step, the state and covariance estimates from the previous time step is used to predict the current state and its uncertainty. This prediction incorporates the system dynamics and the input flow measurements. The predicted state estimate is obtained by propagating the previous state estimate through the process model, accounting for the input flows and adjustable zero. The predicted covariance estimate represents the uncertainty associated with the predicted state.

The update step involves incorporating the measurements obtained from the tanks into the prediction to obtain an improved estimate of the tank levels. It employs a measurement model that relates the tank measurements to the state variables. He formulates the measurement model based on the sensor characteristics and the interdependencies between the tanks. The update step adjusts the predicted state estimate based on the difference between the predicted measurements and the actual measurements obtained from the tanks.

To account for measurement noise and process noise, defines measurement noise covariance and process noise covariance matrices. The measurement noise covariance matrix represents the uncertainties associated with the measurements, such as sensor noise. The process noise covariance matrix represents the uncertainties in the system dynamics and input flows. Emphasizing the importance of accurately characterizing these noise sources to achieve optimal estimation performance.

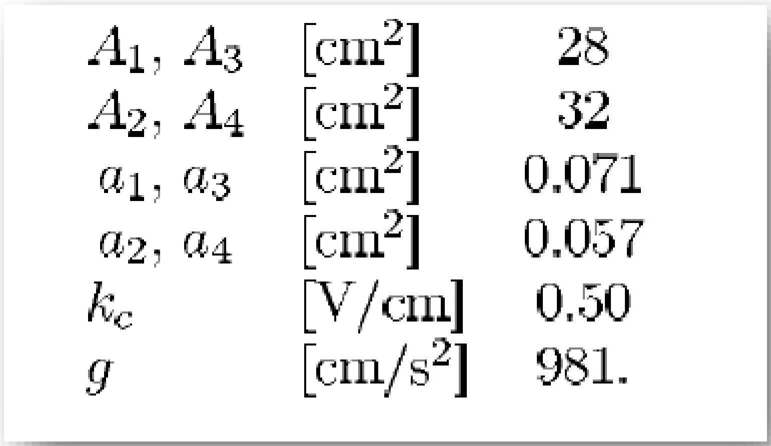
Addressing the issue of model uncertainty by incorporating adaptive techniques in the Kalman Filter implementation. Also exploring the use of an adaptive Kalman Filter that adjusts the filter gain based on the estimation error and the model uncertainties. This adaptive approach allows the Kalman Filter to adapt to variations in the system parameters and improve estimation accuracy.

Formulation:

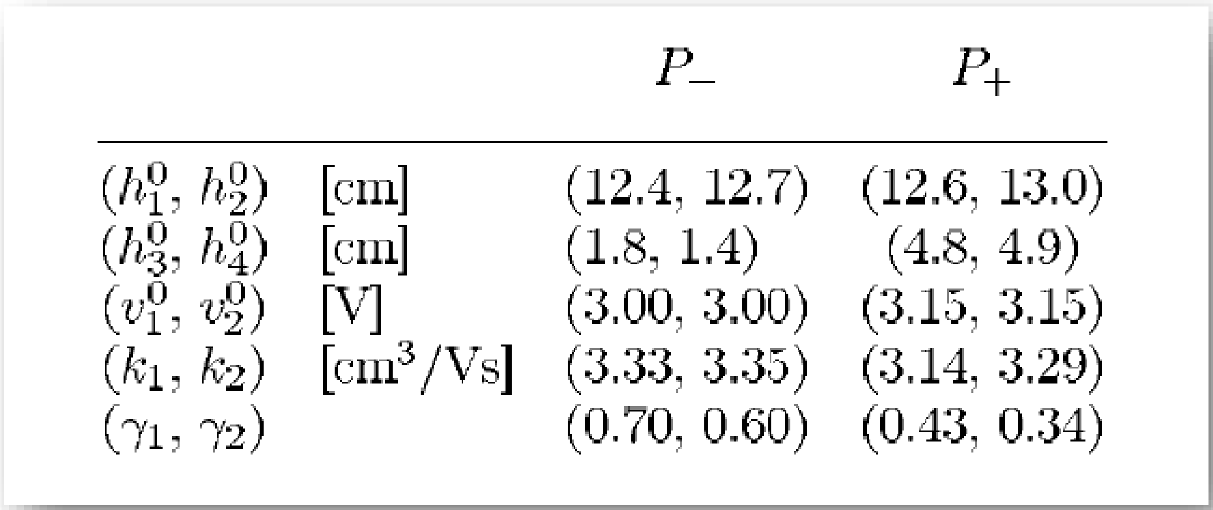
$$\frac{dh_1}{dt} = -\frac{a_1}{A_1}\sqrt{2gh_1} + \frac{a_3}{A_1}\sqrt{2gh_3} + \frac{\gamma_1 k_1}{A_1}v_1$$
$$\frac{dh_2}{dt} = -\frac{a_2}{A_2}\sqrt{2gh_2} + \frac{a_4}{A_2}\sqrt{2gh_4} + \frac{\gamma_2 k_2}{A_2}v_2$$
$$\frac{dh_3}{dt} = -\frac{a_3}{A_3}\sqrt{2gh_3} + \frac{(1-\gamma_2)k_2}{A_3}v_2$$
$$\frac{dh_4}{dt} = -\frac{a_4}{A_4}\sqrt{2gh_4} + \frac{(1-\gamma_1)k_1}{A_4}v_1$$

Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503

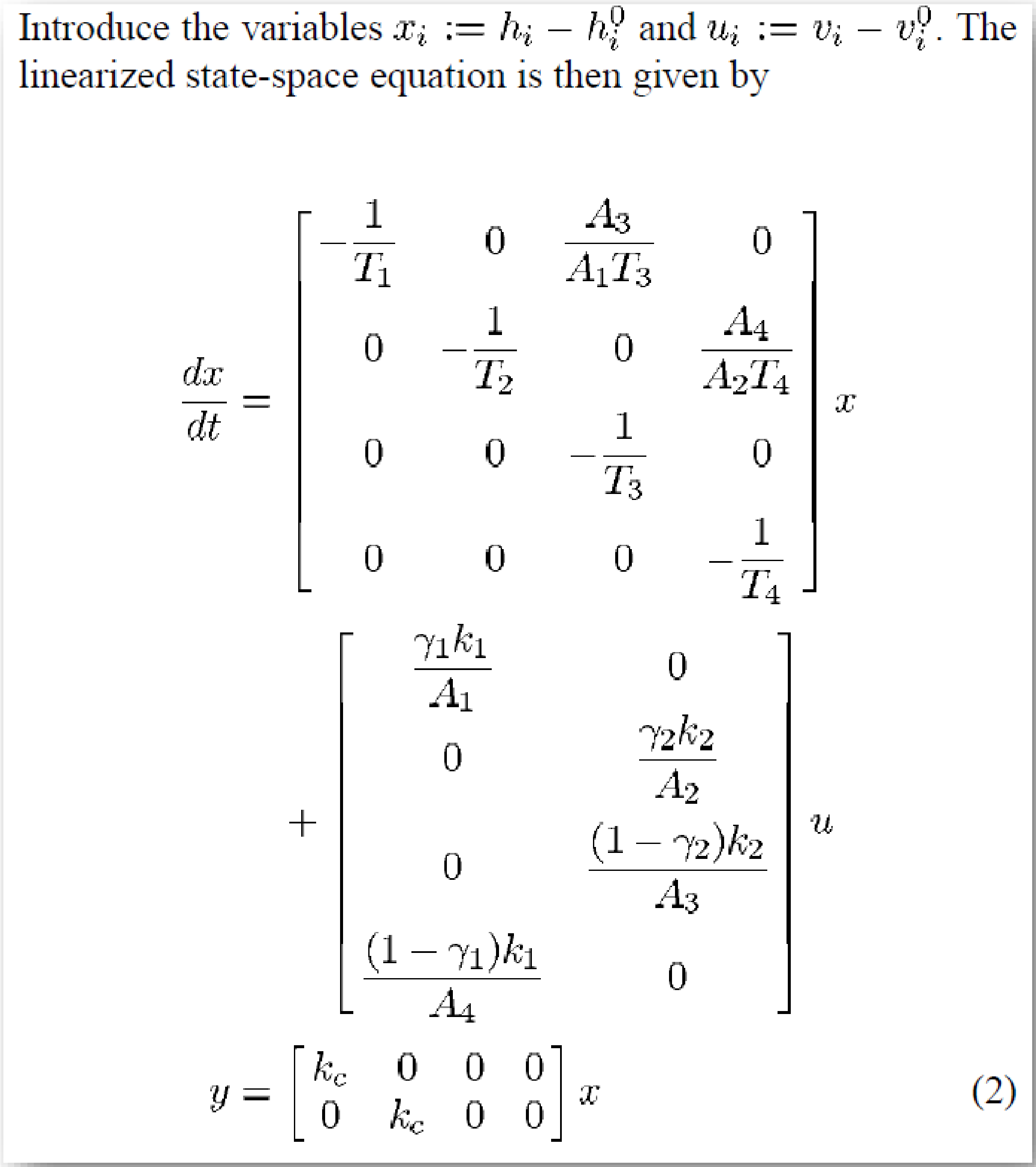


A_1, A_3	[cm^2]	28
A_2, A_4	[cm^2]	32
a_1, a_3	[cm^2]	0.071
a_2, a_4	[cm^2]	0.057
k_c	[V/cm]	0.50
g	[cm/s^2]	981.



		P_-	P_+
(h_1^0, h_2^0)	[cm]	(12.4, 12.7)	(12.6, 13.0)
(h_3^0, h_4^0)	[cm]	(1.8, 1.4)	(4.8, 4.9)
(v_1^0, v_2^0)	[V]	(3.00, 3.00)	(3.15, 3.15)
(k_1, k_2)	[cm^3/Vs]	(3.33, 3.35)	(3.14, 3.29)
(γ_1, γ_2)		(0.70, 0.60)	(0.43, 0.34)

Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503



Introduce the variables $x_i := h_i - h_i^0$ and $u_i := v_i - v_i^0$. The linearized state-space equation is then given by

$$\frac{dx}{dt} = \begin{bmatrix} -\frac{1}{T_1} & 0 & \frac{A_3}{A_1 T_3} & 0 \\ 0 & -\frac{1}{T_2} & 0 & \frac{A_4}{A_2 T_4} \\ 0 & 0 & -\frac{1}{T_3} & 0 \\ 0 & 0 & 0 & -\frac{1}{T_4} \end{bmatrix} x + \begin{bmatrix} \frac{\gamma_1 k_1}{A_1} & 0 \\ 0 & \frac{\gamma_2 k_2}{A_2} \\ 0 & \frac{(1-\gamma_2)k_2}{A_3} \\ \frac{(1-\gamma_1)k_1}{A_4} & 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} k_c & 0 & 0 & 0 \\ 0 & k_c & 0 & 0 \end{bmatrix} x \tag{2}$$

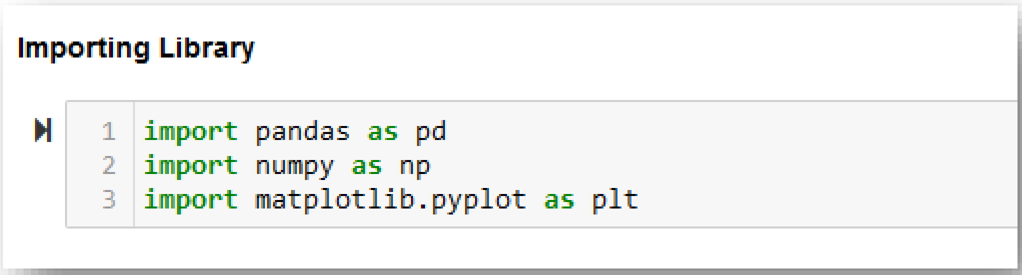
Mini Project – Kalman : Report

Author: Aloy Banerjee

Roll No.: CH22M503

Coding Explanation:

Step1: Importing all the necessary



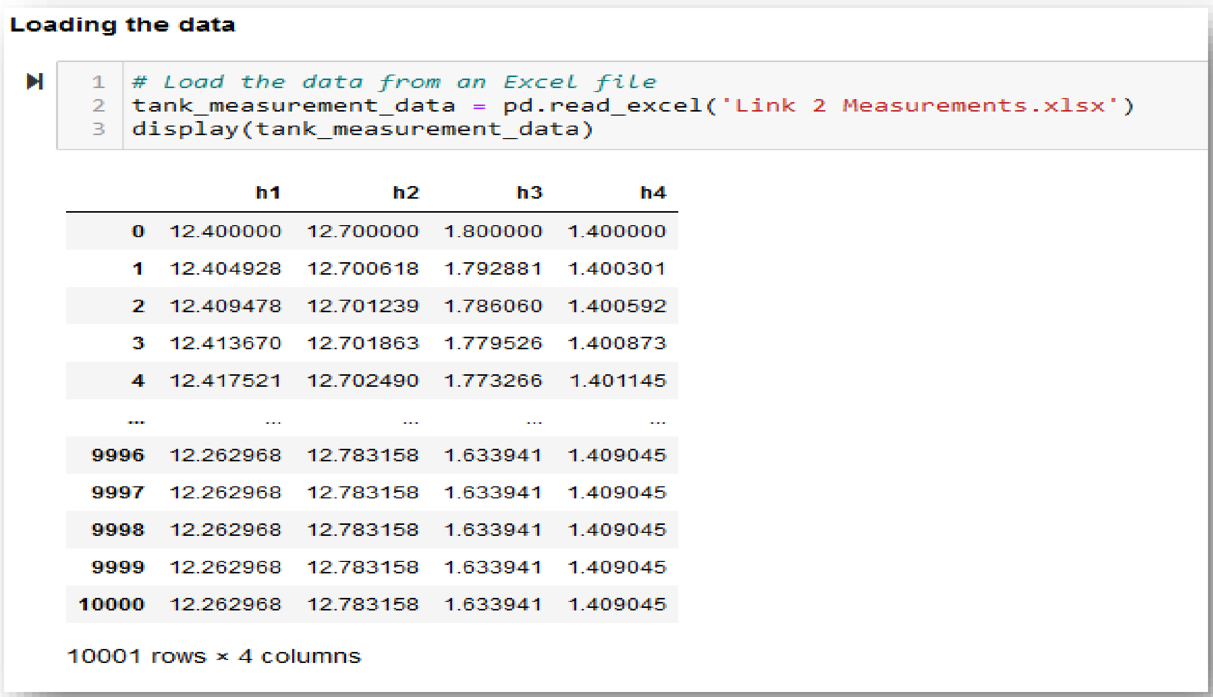
Importing Library

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Step2 - Data Loading: Load the measurement data of height given as a simulation data and analyse the statistical measure for the same.

Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503

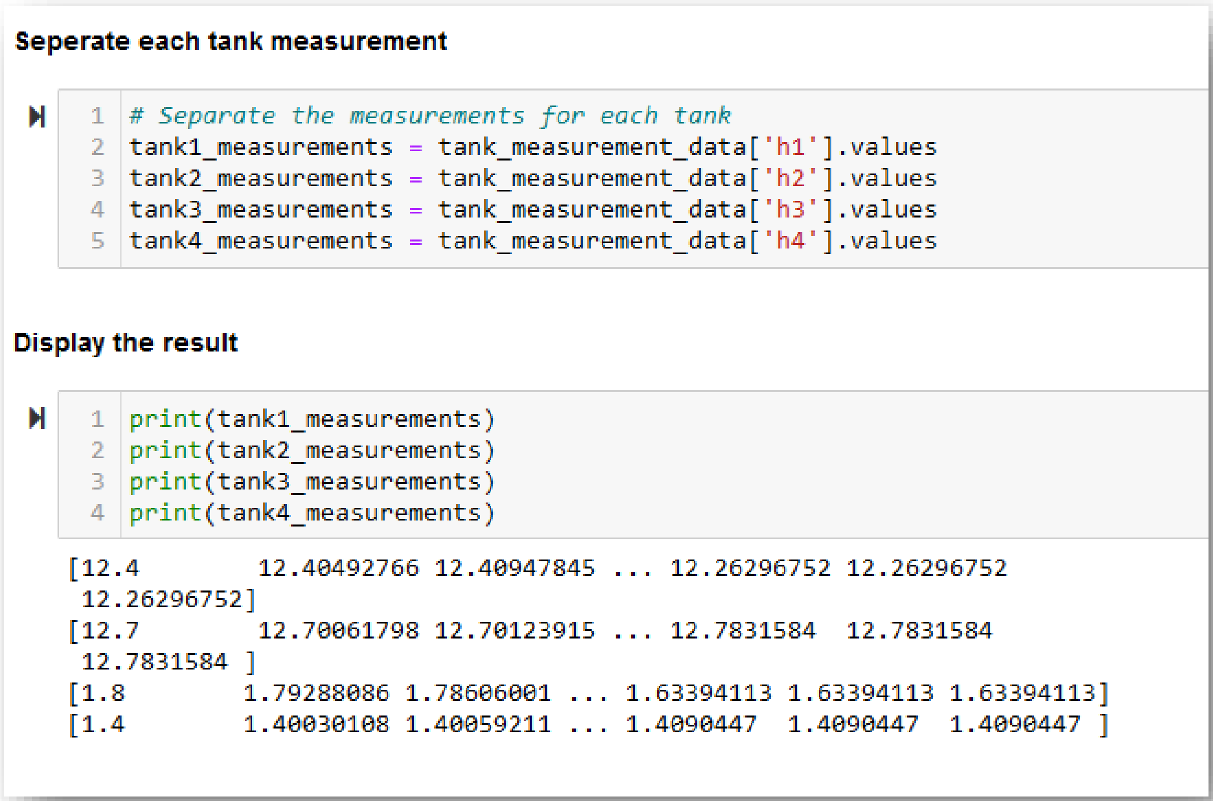


Loading the data

```python
# Load the data from an Excel file
tank_measurement_data = pd.read_excel('Link 2 Measurements.xlsx')
display(tank_measurement_data)
```

	h1	h2	h3	h4
0	12.400000	12.700000	1.800000	1.400000
1	12.404928	12.700618	1.792881	1.400301
2	12.409478	12.701239	1.786060	1.400592
3	12.413670	12.701863	1.779526	1.400873
4	12.417521	12.702490	1.773266	1.401145
...	...	...	...	...
9996	12.262968	12.783158	1.633941	1.409045
9997	12.262968	12.783158	1.633941	1.409045
9998	12.262968	12.783158	1.633941	1.409045
9999	12.262968	12.783158	1.633941	1.409045
10000	12.262968	12.783158	1.633941	1.409045

10001 rows × 4 columns



Seperate each tank measurement

```python
# Separate the measurements for each tank
tank1_measurements = tank_measurement_data['h1'].values
tank2_measurements = tank_measurement_data['h2'].values
tank3_measurements = tank_measurement_data['h3'].values
tank4_measurements = tank_measurement_data['h4'].values
```

Display the result

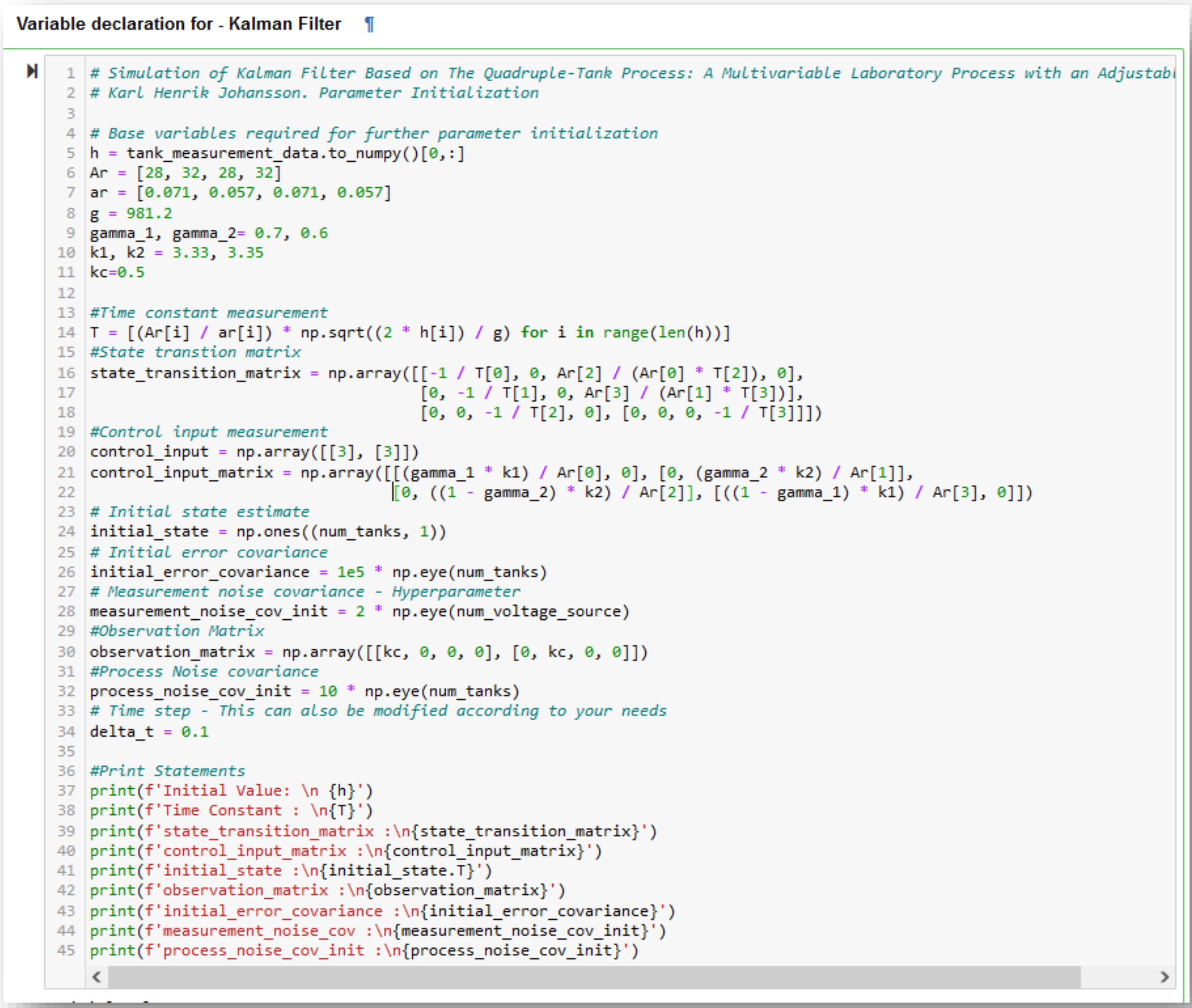
```python
print(tank1_measurements)
print(tank2_measurements)
print(tank3_measurements)
print(tank4_measurements)
```

```
[12.4        12.40492766 12.40947845 ... 12.26296752 12.26296752
 12.26296752]
[12.7        12.70061798 12.70123915 ... 12.7831584  12.7831584
 12.7831584 ]
[1.8        1.79288086 1.78606001 ... 1.63394113 1.63394113 1.63394113]
[1.4        1.40030108 1.40059211 ... 1.4090447  1.4090447  1.4090447 ]
```

Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503

<u>Step3 – Parameter Declaration for Kalman Filter Implementation based on paper:</u> All the required parameter like control input, process noise covariance, measurement noise covariance, state transition matrix etc. are defined here in this declaration section,


Variable declaration for - Kalman Filter ¶

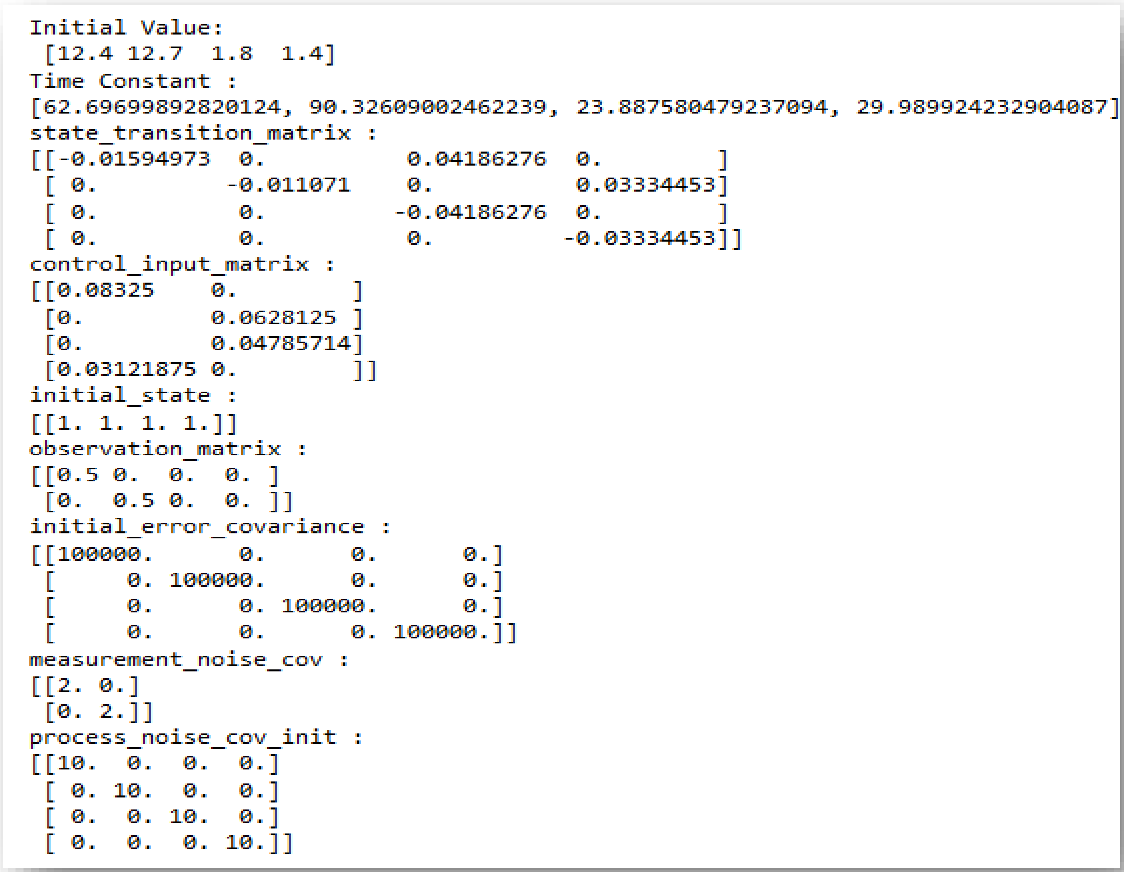
```python
# Simulation of Kalman Filter Based on The Quadruple-Tank Process: A Multivariable Laboratory Process with an Adjustabl
# Karl Henrik Johansson. Parameter Initialization

# Base variables required for further parameter initialization
h = tank_measurement_data.to_numpy()[0,:]
Ar = [28, 32, 28, 32]
ar = [0.071, 0.057, 0.071, 0.057]
g = 981.2
gamma_1, gamma_2= 0.7, 0.6
k1, k2 = 3.33, 3.35
kc=0.5

#Time constant measurement
T = [(Ar[i] / ar[i]) * np.sqrt((2 * h[i]) / g) for i in range(len(h))]
#State transtion matrix
state_transition_matrix = np.array([[-1 / T[0], 0, Ar[2] / (Ar[0] * T[2]), 0],
                                    [0, -1 / T[1], 0, Ar[3] / (Ar[1] * T[3])],
                                    [0, 0, -1 / T[2], 0], [0, 0, 0, -1 / T[3]]])
#Control input measurement
control_input = np.array([[3], [3]])
control_input_matrix = np.array([[(gamma_1 * k1) / Ar[0], 0], [0, (gamma_2 * k2) / Ar[1]],
                                 [0, ((1 - gamma_2) * k2) / Ar[2]], [((1 - gamma_1) * k1) / Ar[3], 0]])
# Initial state estimate
initial_state = np.ones((num_tanks, 1))
# Initial error covariance
initial_error_covariance = 1e5 * np.eye(num_tanks)
# Measurement noise covariance - Hyperparameter
measurement_noise_cov_init = 2 * np.eye(num_voltage_source)
#Observation Matrix
observation_matrix = np.array([[kc, 0, 0, 0], [0, kc, 0, 0]])
#Process Noise covariance
process_noise_cov_init = 10 * np.eye(num_tanks)
# Time step - This can also be modified according to your needs
delta_t = 0.1

#Print Statements
print(f'Initial Value: \n {h}')
print(f'Time Constant : \n{T}')
print(f'state_transition_matrix :\n{state_transition_matrix}')
print(f'control_input_matrix :\n{control_input_matrix}')
print(f'initial_state :\n{initial_state.T}')
print(f'observation_matrix :\n{observation_matrix}')
print(f'initial_error_covariance :\n{initial_error_covariance}')
print(f'measurement_noise_cov :\n{measurement_noise_cov_init}')
print(f'process_noise_cov_init :\n{process_noise_cov_init}')
```

Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503



```
Initial Value:
 [12.4 12.7  1.8  1.4]
Time Constant :
[62.69699892820124, 90.32609002462239, 23.887580479237094, 29.989924232904087]
state_transition_matrix :
[[-0.01594973  0.          0.04186276  0.        ]
 [ 0.         -0.011071    0.          0.03334453]
 [ 0.          0.         -0.04186276  0.        ]
 [ 0.          0.          0.         -0.03334453]]
control_input_matrix :
[[0.08325    0.        ]
 [0.         0.0628125 ]
 [0.         0.04785714]
 [0.03121875 0.        ]]
initial_state :
[[1. 1. 1. 1.]]
observation_matrix :
[[0.5 0.  0.  0. ]
 [0.  0.5 0.  0. ]]
initial_error_covariance :
[[100000.      0.      0.      0.]
 [     0. 100000.      0.      0.]
 [     0.      0. 100000.      0.]
 [     0.      0.      0. 100000.]]
measurement_noise_cov :
[[2. 0.]
 [0. 2.]]
process_noise_cov_init :
[[10.  0.  0.  0.]
 [ 0. 10.  0.  0.]
 [ 0.  0. 10.  0.]
 [ 0.  0.  0. 10.]]
```

Step4 – Common Function Declaration: As per our thought process we have considered two approaches for performing the Kalman Filter implementation,

1. **First Approach :** Given the observations and system characteristics, simulates a Kalman filter. Followed solely the explanations given in class, however encountered several problems using different measurement units.

Explanation of first approach:

1. **Initialize the necessary variables and lists for storing the simulation results.**
   a. **X_post_list, X_prior_list, P_post_list, P_pri_list, Kalman_gain_list, and measurement_error_list** are empty lists to store the estimated states, post-measurement state covariances,

prior-measurement state covariances, Kalman gains, and measurement errors, respectively.

2. **Iterate over each measurement in the z array.**
    a. Loop through the range of **num_measurements** (length of z array).

3. **Prediction Step:**
    a. Calculate the prior state estimate (**X_pri**) by multiplying the system dynamics matrix (**A**) with the current state vector (**x**) and adding the product of the control matrix (**B)** and the corresponding control input (**u**).
    b. Calculate the prior state covariance (**P_pri**) using the system dynamics matrix (**A**), initial state covariance **(P**), and process noise covariance matrix (**Q**).

4. **Update Step:**
    a. Calculate the Kalman gain (**K**) by multiplying the prior state covariance **(P_pri)** with the transpose of the observation measurement matrix (**H**), and then multiplying by the inverse of the product of the observation measurement matrix (**H**), prior state covariance (**P_pri**), and its transpose, along with the measurement noise covariance matrix (R).
    b. Estimate the measured value (**Z_est**) by multiplying the observation measurement matrix (H) with the prior state estimate (**X_pri**).
    c. Calculate the measurement error (**E**) by subtracting the estimated measurement (**Z_est**) from the actual measurement (**z**) for the current iteration.

5. **Update the state estimate and covariance:**
    a. Update the state estimate (**initial_state**) by adding the product of the Kalman gain (**K**) and the measurement error (**E**) to the prior state estimate (**X_pri**).
    b. Update the state covariance **(P**) by subtracting the product of the product of the Kalman gain (**K**), observation measurement matrix

(**H**), and the prior state covariance (**P_pri**) from the prior state covariance (**P_pri**).

6. <u>**Store the calculated values:**</u>
   a. Append the current values of the state estimate (**initial_state**), prior state estimate (**X_pri),** post-measurement state covariance **(P),** prior-measurement state covariance **(P_pri),** Kalman gain **(K),** and measurement error (E) to their respective lists **(X_post_list, X_prior_list, P_post_list, P_pri_list, Kalman_gain_list, measurement_error_list).**

7. <u>**Check for convergence:**</u>
   a. Calculate the L2 norm (l2_norm) between the prior state estimate (**X_pri**) and the updated state estimate (**initial_state**).
   b. If the L2 norm is within a predefined threshold value (**threshold**), set l2_norm_converged to True and print a message indicating the convergence at the current iteration.

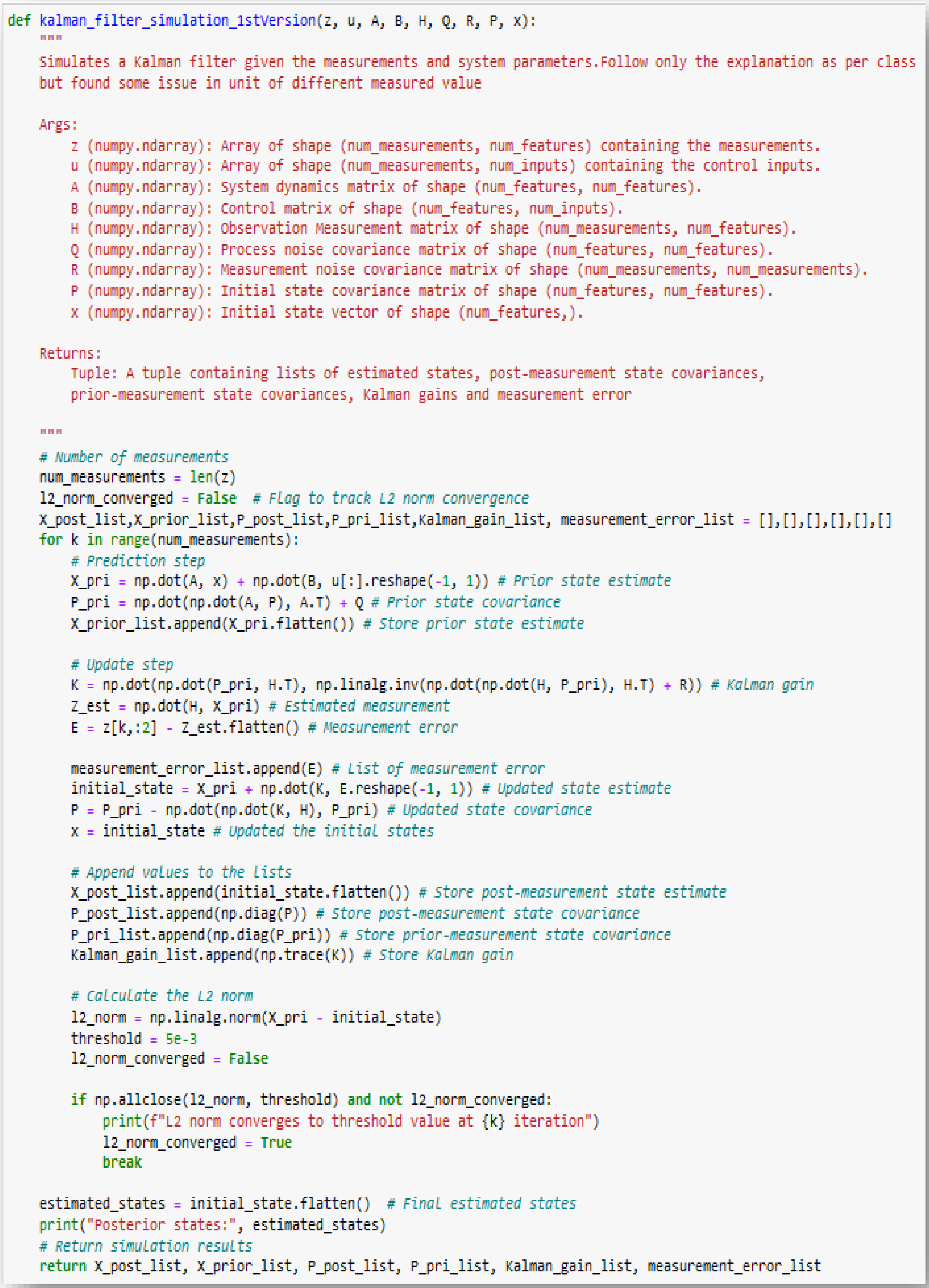
8. <u>**Finalize the estimation:**</u>
   a. Flatten the final estimated state (**initial_state)** into a 1D array and store it in the **estimated_states** variable.

9. <u>**Print the posterior states:**</u>
   a. Print the final estimated states (**estimated_states**) after the simulation.

10. <u>**Return the simulation results:**</u>
    a. Return the lists of estimated states (**X_post_list and X_prior_list**), post-measurement state covariances **(P_post_list),** prior-measurement state covariances (**P_pri_list**), Kalman gains (**Kalman_gain_list**), and measurement errors **(measurement_error_list)** as a tuple.

Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503



```python
def kalman_filter_simulation_1stVersion(z, u, A, B, H, Q, R, P, x):
    """
    Simulates a Kalman filter given the measurements and system parameters.Follow only the explanation as per class
    but found some issue in unit of different measured value

    Args:
        z (numpy.ndarray): Array of shape (num_measurements, num_features) containing the measurements.
        u (numpy.ndarray): Array of shape (num_measurements, num_inputs) containing the control inputs.
        A (numpy.ndarray): System dynamics matrix of shape (num_features, num_features).
        B (numpy.ndarray): Control matrix of shape (num_features, num_inputs).
        H (numpy.ndarray): Observation Measurement matrix of shape (num_measurements, num_features).
        Q (numpy.ndarray): Process noise covariance matrix of shape (num_features, num_features).
        R (numpy.ndarray): Measurement noise covariance matrix of shape (num_measurements, num_measurements).
        P (numpy.ndarray): Initial state covariance matrix of shape (num_features, num_features).
        x (numpy.ndarray): Initial state vector of shape (num_features,).

    Returns:
        Tuple: A tuple containing lists of estimated states, post-measurement state covariances,
        prior-measurement state covariances, Kalman gains and measurement error

    """
    # Number of measurements
    num_measurements = len(z)
    l2_norm_converged = False  # Flag to track L2 norm convergence
    X_post_list,X_prior_list,P_post_list,P_pri_list,Kalman_gain_list, measurement_error_list = [],[],[],[],[],[]
    for k in range(num_measurements):
        # Prediction step
        X_pri = np.dot(A, x) + np.dot(B, u[:].reshape(-1, 1)) # Prior state estimate
        P_pri = np.dot(np.dot(A, P), A.T) + Q # Prior state covariance
        X_prior_list.append(X_pri.flatten()) # Store prior state estimate

        # Update step
        K = np.dot(np.dot(P_pri, H.T), np.linalg.inv(np.dot(np.dot(H, P_pri), H.T) + R)) # Kalman gain
        Z_est = np.dot(H, X_pri) # Estimated measurement
        E = z[k,:2] - Z_est.flatten() # Measurement error

        measurement_error_list.append(E) # List of measurement error
        initial_state = X_pri + np.dot(K, E.reshape(-1, 1)) # Updated state estimate
        P = P_pri - np.dot(np.dot(K, H), P_pri) # Updated state covariance
        x = initial_state # Updated the initial states

        # Append values to the lists
        X_post_list.append(initial_state.flatten()) # Store post-measurement state estimate
        P_post_list.append(np.diag(P)) # Store post-measurement state covariance
        P_pri_list.append(np.diag(P_pri)) # Store prior-measurement state covariance
        Kalman_gain_list.append(np.trace(K)) # Store Kalman gain

        # Calculate the L2 norm
        l2_norm = np.linalg.norm(X_pri - initial_state)
        threshold = 5e-3
        l2_norm_converged = False

        if np.allclose(l2_norm, threshold) and not l2_norm_converged:
            print(f"L2 norm converges to threshold value at {k} iteration")
            l2_norm_converged = True
            break

    estimated_states = initial_state.flatten()  # Final estimated states
    print("Posterior states:", estimated_states)
    # Return simulation results
    return X_post_list, X_prior_list, P_post_list, P_pri_list, Kalman_gain_list, measurement_error_list
```

Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503

2. **Second Approach :** Given the observations and system characteristics, simulates a Kalman filter. This has been updated to align with the thesis statement of the research report and the concept covered in class, taking the unit measurement for each measured value into account.

<u>Explanation of second approach</u>:

1. **<u>Initialize the necessary variables and lists for storing the simulation results:</u>**
   a. **X_post_init** is set as the initial state estimate, extracted from the Z array, reshaped as a column vector.
   b. **P_posterior_0** is initialized as a diagonal matrix with large values to represent high uncertainty.
   c. **X_posterior_store, X_prior_store, P_posterior_store, P_prior_store, Kalman_Gain_store, Innovation_store, and Residue_store** are empty lists to store the estimated states, post-measurement state covariances, prior-measurement state covariances, Kalman gains, measurement errors, innovation, and residue, respectively.

2. **<u>Iterate over each measurement in the Z array:</u>**
   a. Loop through the range of **Z.shape[0] - 1 (number of measurements minus 1**).

3. **<u>Prediction Step:</u>**
   a. Calculate the measurement prediction **(V_i)** by multiplying the observation measurement matrix (**H**) with the current measurement from the Z array, reshaped as a column vector.
   b. Calculate the prior state estimate **(X_pri)** by updating the initial state estimate **(X_post_init)** using the system dynamics matrix (**A**), control matrix (B), control inputs (**U**), and previous measurement prediction (**V_i**). The result is scaled by the time step (**delta_t**) and then added to the initial state (**h0**).

c. Update the initial state (**h0**) for the next iteration with the current measurement from **the Z array**, reshaped as a column vector.
d. Calculate the prior state covariance (**P_pri**) by updating the initial state covariance (**P_posterior_0**) using the system dynamics matrix (**A**), prior state covariance (**P_posterior_0**), process noise covariance matrix (**Q**), and the square of the time step (**delta_t**).

4. **Update Step:**
    a. Calculate the Kalman gain (**K**) by multiplying the prior state covariance (**P_pri**) with the transpose of the observation measurement matrix (**H**), and then multiplying by the inverse of the sum of the product of the observation measurement matrix (H), prior state covariance (**P_pri**), and its transpose, along with the measurement noise covariance matrix (R).
    b. Calculate the true measurement (**V_true**) by multiplying the observation measurement matrix (**H**) with the previous measurement from the Z array, reshaped as a column vector.
    c. Calculate the innovation or measurement residual (**E**) by subtracting the observation measurement of the prior state estimate (**H.dot(X_pri))** from the true measurement (**V_true**).

5. **Update the state estimate and covariance:**
    a. Update the state estimate (**X_posterior**) by adding the product of the Kalman gain (**K**) and the measurement residual (E) to the prior state estimate (**X_pri**).
    b. Update the state covariance (**P**) by subtracting the product of the product of the Kalman gain (**K**), observation measurement matrix (**H**), and the prior state covariance (**P_pri**) from the prior state covariance (**P_pri**).

6. **Store the calculated values:**
    a. Append the current values of the state estimate (**X_posterior**), prior state estimate (**X_pri),** post-measurement state covariance (P), prior-measurement state covariance (**P_pri),** Kalman gain (K),

measurement error (E), and residue (Z[iteration, :2].reshape(-1, 1) - H.dot(**X_posterior**)) to their respective lists (**X_posterior_store, X_prior_store, P_posterior_store, P_prior_store, Kalman_Gain_store, Innovation_store, Residue_store**).

7. **<u>Check for convergence:</u>**
   a. Calculate the L2 norm (l2_norm) between the prior state estimate (**X_pri**) and the updated state estimate (**X_posterior**).
   b. If the L2 norm is within a predefined threshold value (threshold), set l2_norm_converged to True and print a message indicating the convergence at the current iteration.

8. **<u>Calculate the mean estimation for each tank:</u>**
   a. Compute the mean estimation for each tank by taking the mean value of the corresponding elements in the **X_posterior** array.

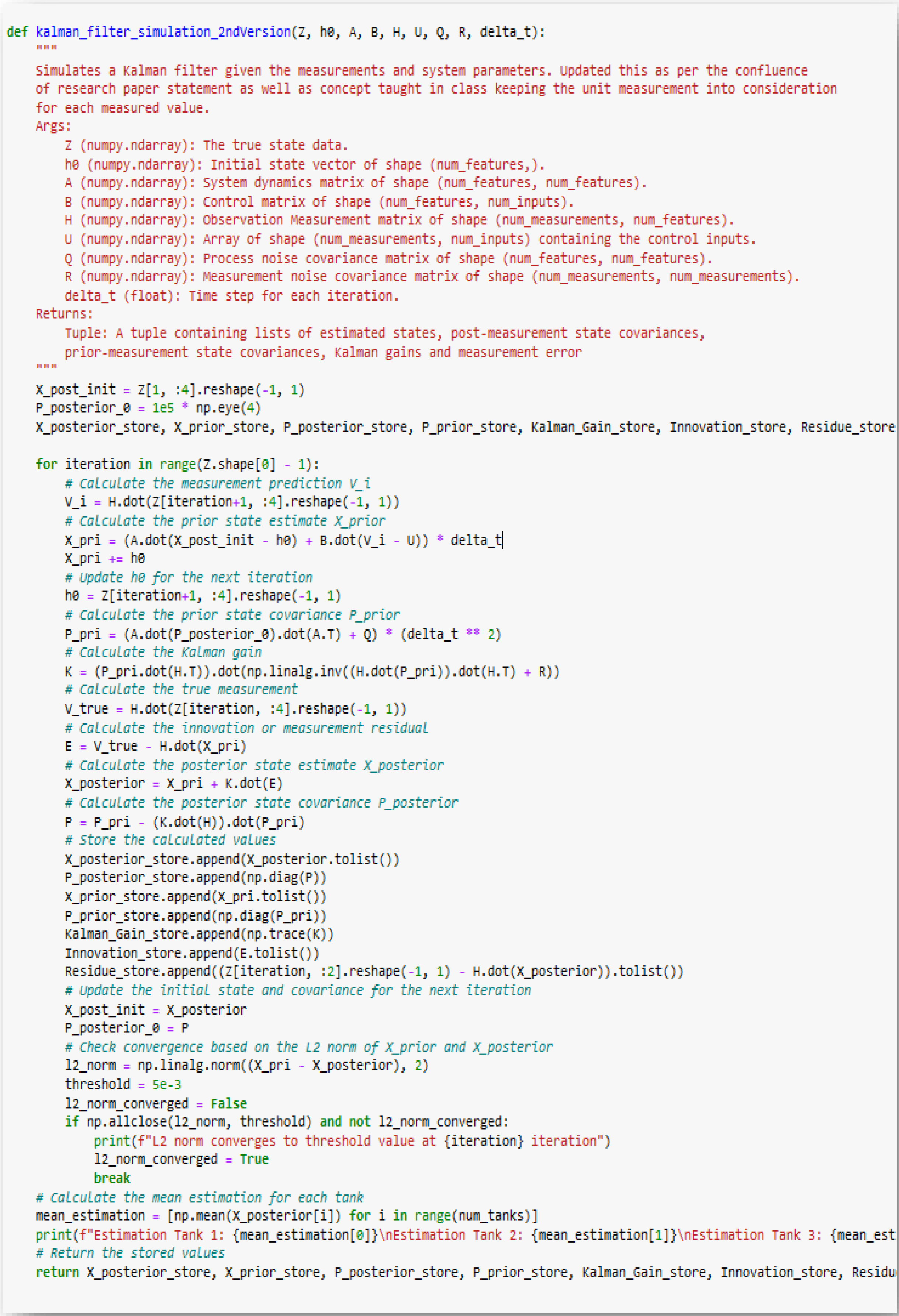
9. **<u>Print the estimation for each tank:</u>**
   a. Print the mean estimation values for Tank 1, Tank 2, Tank 3, and Tank 4.

10. **<u>Return the stored values:</u>**
   a. Return the lists of estimated states (**X_posterior_store**), prior state estimates (**X_prior_store**), post-measurement state covariances (**P_posterior_store**), prior-measurement state covariances (**P_prior_store**), Kalman gains (**Kalman_Gain_store**), innovation (**Innovation_store**), and residue (**Residue_store**) as a tuple.

Mini Project – Kalman : Report

Author: Aloy Banerjee

Roll No.: CH22M503

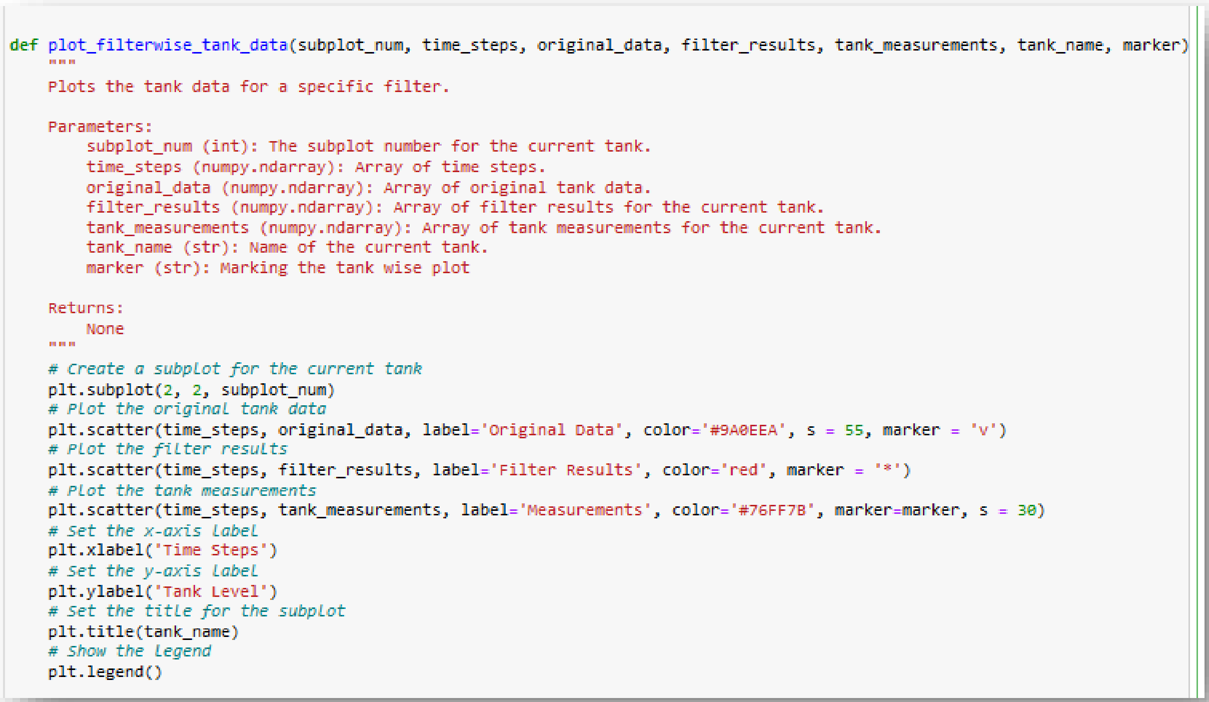


```
def kalman_filter_simulation_2ndVersion(Z, h0, A, B, H, U, Q, R, delta_t):
    """
    Simulates a Kalman filter given the measurements and system parameters. Updated this as per the confluence
    of research paper statement as well as concept taught in class keeping the unit measurement into consideration
    for each measured value.
    Args:
        Z (numpy.ndarray): The true state data.
        h0 (numpy.ndarray): Initial state vector of shape (num_features,).
        A (numpy.ndarray): System dynamics matrix of shape (num_features, num_features).
        B (numpy.ndarray): Control matrix of shape (num_features, num_inputs).
        H (numpy.ndarray): Observation Measurement matrix of shape (num_measurements, num_features).
        U (numpy.ndarray): Array of shape (num_measurements, num_inputs) containing the control inputs.
        Q (numpy.ndarray): Process noise covariance matrix of shape (num_features, num_features).
        R (numpy.ndarray): Measurement noise covariance matrix of shape (num_measurements, num_measurements).
        delta_t (float): Time step for each iteration.
    Returns:
        Tuple: A tuple containing lists of estimated states, post-measurement state covariances,
        prior-measurement state covariances, Kalman gains and measurement error
    """
    X_post_init = Z[1, :4].reshape(-1, 1)
    P_posterior_0 = 1e5 * np.eye(4)
    X_posterior_store, X_prior_store, P_posterior_store, P_prior_store, Kalman_Gain_store, Innovation_store, Residue_store

    for iteration in range(Z.shape[0] - 1):
        # Calculate the measurement prediction V_i
        V_i = H.dot(Z[iteration+1, :4].reshape(-1, 1))
        # Calculate the prior state estimate X_prior
        X_pri = (A.dot(X_post_init - h0) + B.dot(V_i - U)) * delta_t
        X_pri += h0
        # Update h0 for the next iteration
        h0 = Z[iteration+1, :4].reshape(-1, 1)
        # Calculate the prior state covariance P_prior
        P_pri = (A.dot(P_posterior_0).dot(A.T) + Q) * (delta_t ** 2)
        # Calculate the Kalman gain
        K = (P_pri.dot(H.T)).dot(np.linalg.inv((H.dot(P_pri)).dot(H.T) + R))
        # Calculate the true measurement
        V_true = H.dot(Z[iteration, :4].reshape(-1, 1))
        # Calculate the innovation or measurement residual
        E = V_true - H.dot(X_pri)
        # Calculate the posterior state estimate X_posterior
        X_posterior = X_pri + K.dot(E)
        # Calculate the posterior state covariance P_posterior
        P = P_pri - (K.dot(H)).dot(P_pri)
        # Store the calculated values
        X_posterior_store.append(X_posterior.tolist())
        P_posterior_store.append(np.diag(P))
        X_prior_store.append(X_pri.tolist())
        P_prior_store.append(np.diag(P_pri))
        Kalman_Gain_store.append(np.trace(K))
        Innovation_store.append(E.tolist())
        Residue_store.append((Z[iteration, :2].reshape(-1, 1) - H.dot(X_posterior)).tolist())
        # Update the initial state and covariance for the next iteration
        X_post_init = X_posterior
        P_posterior_0 = P
        # Check convergence based on the L2 norm of X_prior and X_posterior
        l2_norm = np.linalg.norm((X_pri - X_posterior), 2)
        threshold = 5e-3
        l2_norm_converged = False
        if np.allclose(l2_norm, threshold) and not l2_norm_converged:
            print(f"L2 norm converges to threshold value at {iteration} iteration")
            l2_norm_converged = True
            break
    # Calculate the mean estimation for each tank
    mean_estimation = [np.mean(X_posterior[i]) for i in range(num_tanks)]
    print(f"Estimation Tank 1: {mean_estimation[0]}\nEstimation Tank 2: {mean_estimation[1]}\nEstimation Tank 3: {mean_est
    # Return the stored values
    return X_posterior_store, X_prior_store, P_posterior_store, P_prior_store, Kalman_Gain_store, Innovation_store, Residu
```

Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503

Apart from the above two method for Kalman Filter implementation we have couple of common functionalities for plotting the different requested graphs,



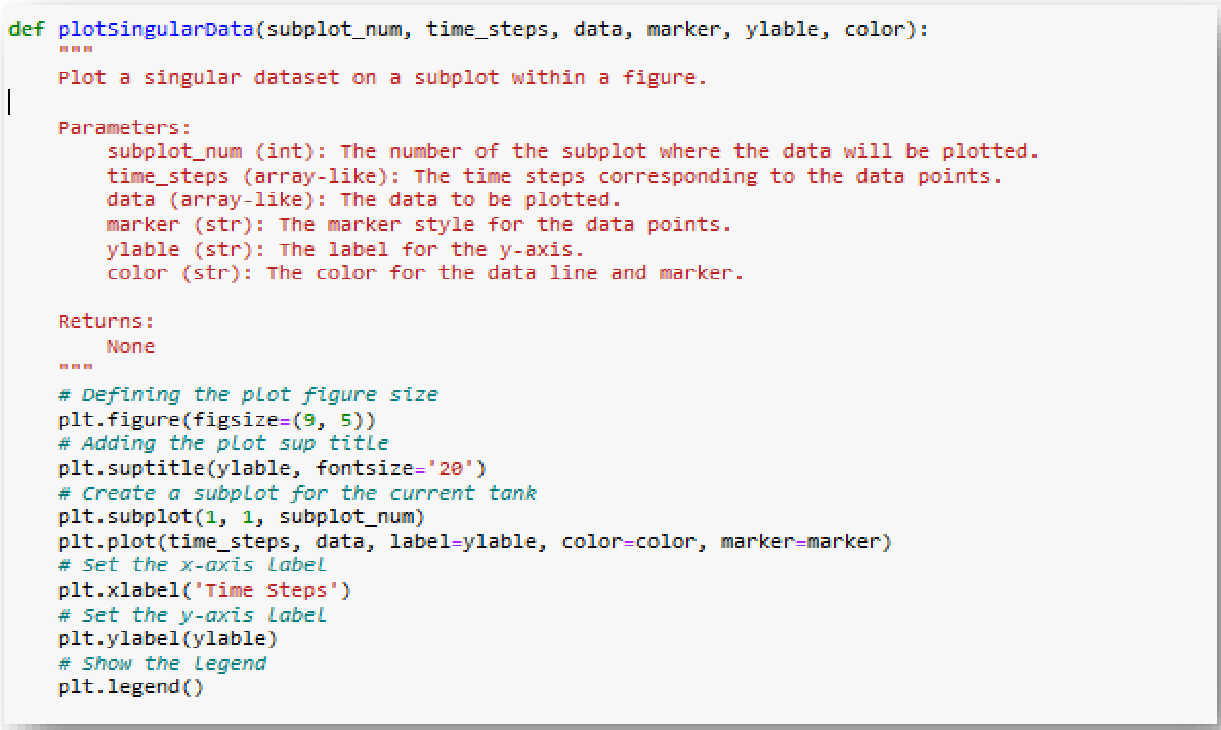
```
def plot_filterwise_tank_data(subplot_num, time_steps, original_data, filter_results, tank_measurements, tank_name, marker)
    """
    Plots the tank data for a specific filter.

    Parameters:
        subplot_num (int): The subplot number for the current tank.
        time_steps (numpy.ndarray): Array of time steps.
        original_data (numpy.ndarray): Array of original tank data.
        filter_results (numpy.ndarray): Array of filter results for the current tank.
        tank_measurements (numpy.ndarray): Array of tank measurements for the current tank.
        tank_name (str): Name of the current tank.
        marker (str): Marking the tank wise plot

    Returns:
        None
    """
    # Create a subplot for the current tank
    plt.subplot(2, 2, subplot_num)
    # Plot the original tank data
    plt.scatter(time_steps, original_data, label='Original Data', color='#9A0EEA', s = 55, marker = 'v')
    # Plot the filter results
    plt.scatter(time_steps, filter_results, label='Filter Results', color='red', marker = '*')
    # Plot the tank measurements
    plt.scatter(time_steps, tank_measurements, label='Measurements', color='#76FF7B', marker=marker, s = 30)
    # Set the x-axis label
    plt.xlabel('Time Steps')
    # Set the y-axis label
    plt.ylabel('Tank Level')
    # Set the title for the subplot
    plt.title(tank_name)
    # Show the legend
    plt.legend()
```

Mini Project – Kalman : Report

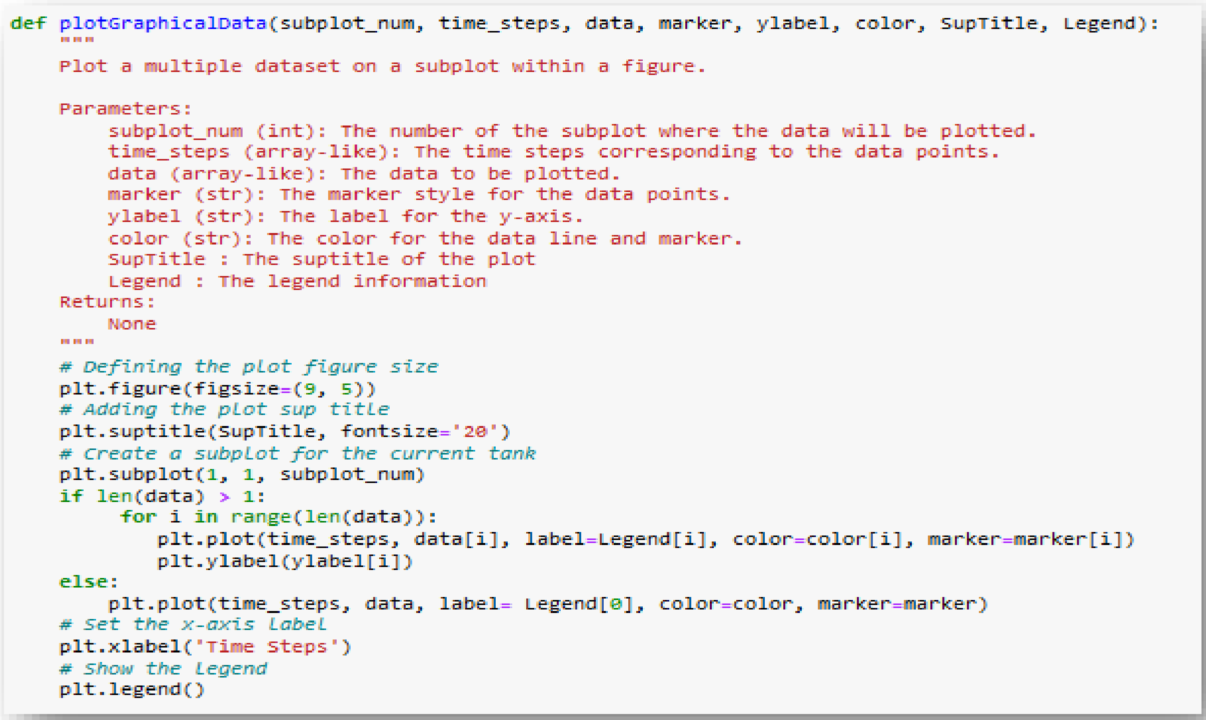
Author: Aloy Banerjee
Roll No.: CH22M503



```python
def plotSingularData(subplot_num, time_steps, data, marker, ylable, color):
    """
    Plot a singular dataset on a subplot within a figure.

    Parameters:
        subplot_num (int): The number of the subplot where the data will be plotted.
        time_steps (array-like): The time steps corresponding to the data points.
        data (array-like): The data to be plotted.
        marker (str): The marker style for the data points.
        ylable (str): The label for the y-axis.
        color (str): The color for the data line and marker.

    Returns:
        None
    """
    # Defining the plot figure size
    plt.figure(figsize=(9, 5))
    # Adding the plot sup title
    plt.suptitle(ylable, fontsize='20')
    # Create a subplot for the current tank
    plt.subplot(1, 1, subplot_num)
    plt.plot(time_steps, data, label=ylable, color=color, marker=marker)
    # Set the x-axis label
    plt.xlabel('Time Steps')
    # Set the y-axis label
    plt.ylabel(ylable)
    # Show the legend
    plt.legend()
```



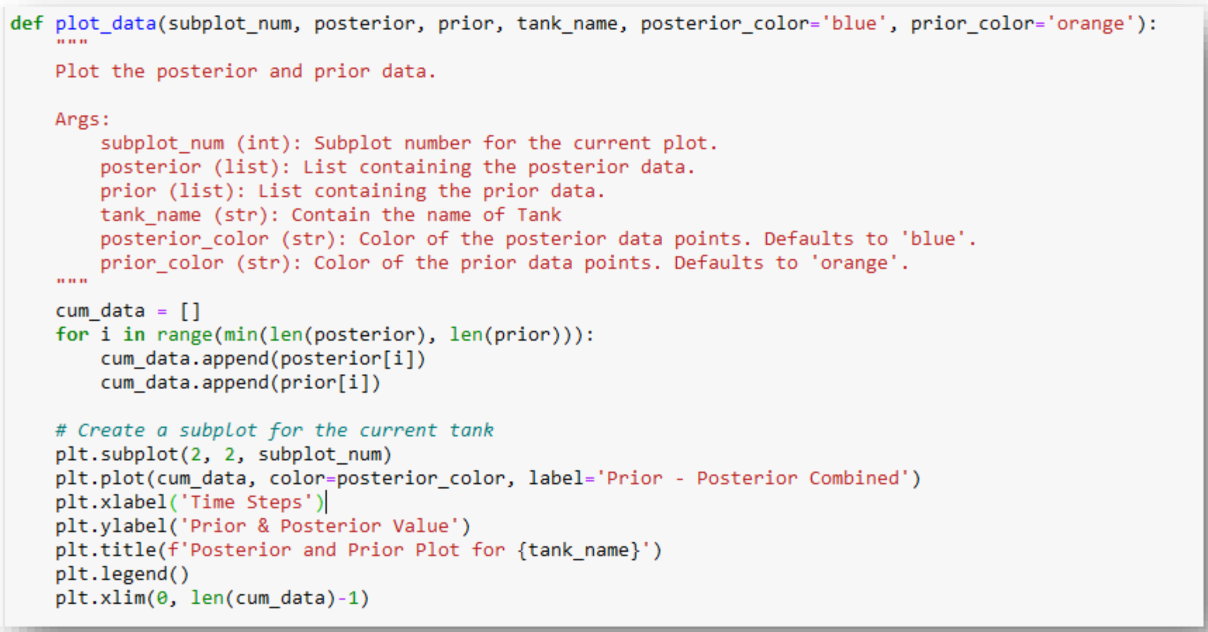
```python
def plotGraphicalData(subplot_num, time_steps, data, marker, ylabel, color, SupTitle, Legend):
    """
    Plot a multiple dataset on a subplot within a figure.

    Parameters:
        subplot_num (int): The number of the subplot where the data will be plotted.
        time_steps (array-like): The time steps corresponding to the data points.
        data (array-like): The data to be plotted.
        marker (str): The marker style for the data points.
        ylabel (str): The label for the y-axis.
        color (str): The color for the data line and marker.
        SupTitle : The suptitle of the plot
        Legend : The legend information
    Returns:
        None
    """
    # Defining the plot figure size
    plt.figure(figsize=(9, 5))
    # Adding the plot sup title
    plt.suptitle(SupTitle, fontsize='20')
    # Create a subplot for the current tank
    plt.subplot(1, 1, subplot_num)
    if len(data) > 1:
         for i in range(len(data)):
            plt.plot(time_steps, data[i], label=Legend[i], color=color[i], marker=marker[i])
            plt.ylabel(ylabel[i])
    else:
        plt.plot(time_steps, data, label= Legend[0], color=color, marker=marker)
    # Set the x-axis label
    plt.xlabel('Time Steps')
    # Show the legend
    plt.legend()
```

Mini Project – Kalman : Report

Author: Aloy Banerjee

Roll No.: CH22M503



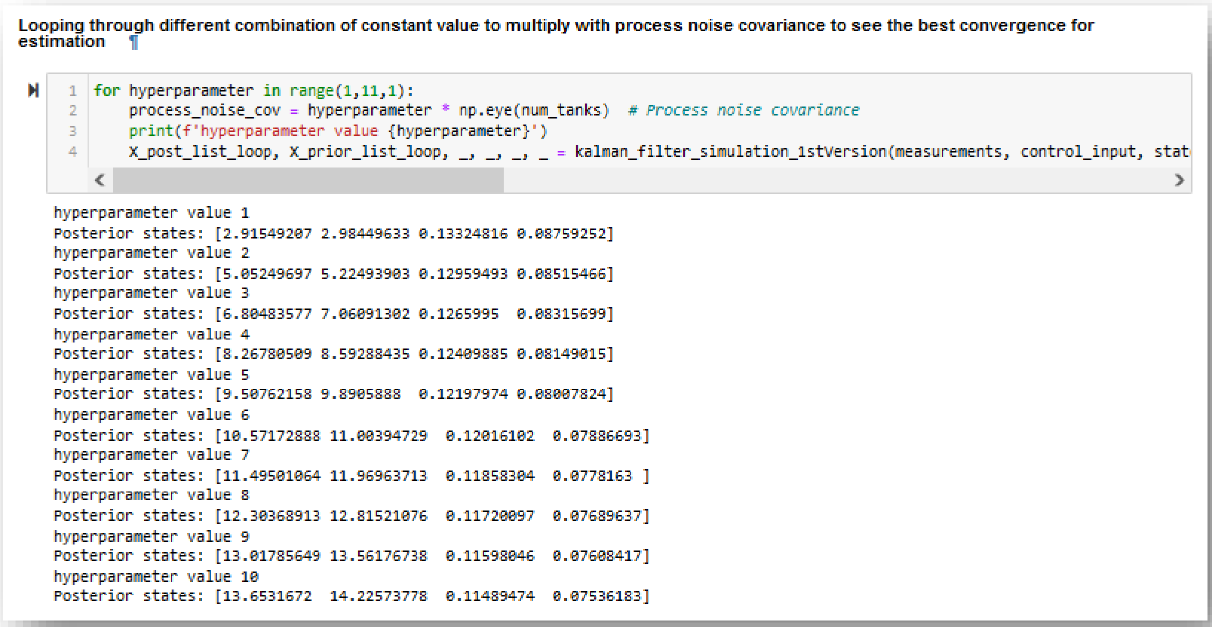
```python
def plot_data(subplot_num, posterior, prior, tank_name, posterior_color='blue', prior_color='orange'):
    """
    Plot the posterior and prior data.

    Args:
        subplot_num (int): Subplot number for the current plot.
        posterior (list): List containing the posterior data.
        prior (list): List containing the prior data.
        tank_name (str): Contain the name of Tank
        posterior_color (str): Color of the posterior data points. Defaults to 'blue'.
        prior_color (str): Color of the prior data points. Defaults to 'orange'.
    """
    cum_data = []
    for i in range(min(len(posterior), len(prior))):
        cum_data.append(posterior[i])
        cum_data.append(prior[i])

    # Create a subplot for the current tank
    plt.subplot(2, 2, subplot_num)
    plt.plot(cum_data, color=posterior_color, label='Prior - Posterior Combined')
    plt.xlabel('Time Steps')
    plt.ylabel('Prior & Posterior Value')
    plt.title(f'Posterior and Prior Plot for {tank_name}')
    plt.legend()
    plt.xlim(0, len(cum_data)-1)
```

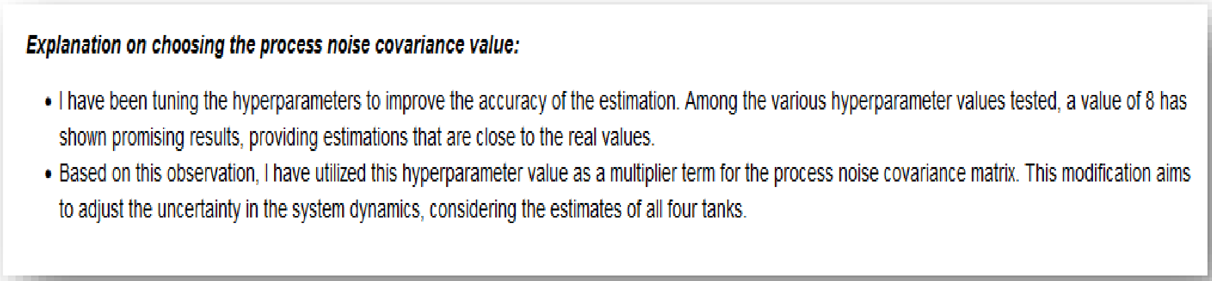
Implementation of Kalman Filter using 1st approach

Step1 – Simulate the Kalman Filter implementation : The process noise covariance matrix and the measurement noise covariance matrix are hyperparameters that can be adjusted to obtain the convergence of the Tank height measurement, as was described in the methodology description. I am changing those parameters as a first step to see the convergence of the tank height measurement.


Looping through different combination of constant value to multiply with process noise covariance to see the best convergence for estimation

```
for hyperparameter in range(1,11,1):
    process_noise_cov = hyperparameter * np.eye(num_tanks)  # Process noise covariance
    print(f'hyperparameter value {hyperparameter}')
    X_post_list_loop, X_prior_list_loop, _, _, _, _ = kalman_filter_simulation_1stVersion(measurements, control_input, stat
```

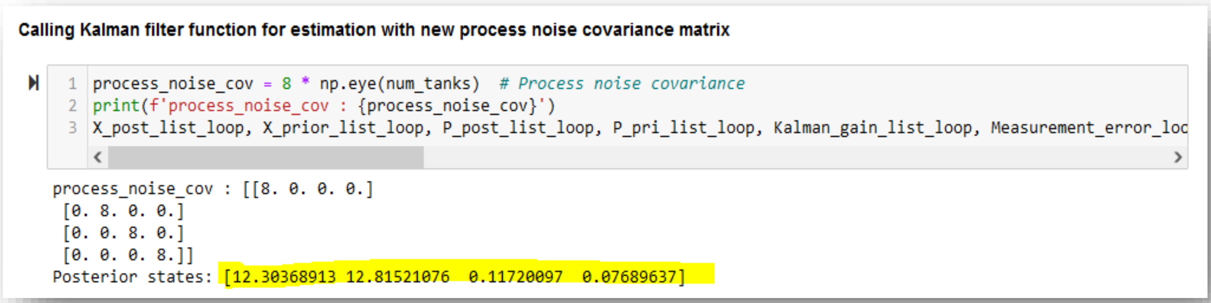
```
hyperparameter value 1
Posterior states: [2.91549207 2.98449633 0.13324816 0.08759252]
hyperparameter value 2
Posterior states: [5.05249697 5.22493903 0.12959493 0.08515466]
hyperparameter value 3
Posterior states: [6.80483577 7.06091302 0.1265995  0.08315699]
hyperparameter value 4
Posterior states: [8.26780509 8.59288435 0.12409885 0.08149015]
hyperparameter value 5
Posterior states: [9.50762158 9.8905888  0.12197974 0.08007824]
hyperparameter value 6
Posterior states: [10.57172888 11.00394729  0.12016102  0.07886693]
hyperparameter value 7
Posterior states: [11.49501064 11.96963713  0.11858304  0.0778163 ]
hyperparameter value 8
Posterior states: [12.30368913 12.81521076  0.11720097  0.07689637]
hyperparameter value 9
Posterior states: [13.01785649 13.56176738  0.11598046  0.07608417]
hyperparameter value 10
Posterior states: [13.6531672  14.22573778  0.11489474  0.07536183]
```


Explanation on choosing the process noise covariance value:

- I have been tuning the hyperparameters to improve the accuracy of the estimation. Among the various hyperparameter values tested, a value of 8 has shown promising results, providing estimations that are close to the real values.
- Based on this observation, I have utilized this hyperparameter value as a multiplier term for the process noise covariance matrix. This modification aims to adjust the uncertainty in the system dynamics, considering the estimates of all four tanks.

So, after the selection of the process noise covariance we are running the Kalman filter implementation again to display the posterior estimation of the tank's height.

Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503


Calling Kalman filter function for estimation with new process noise covariance matrix

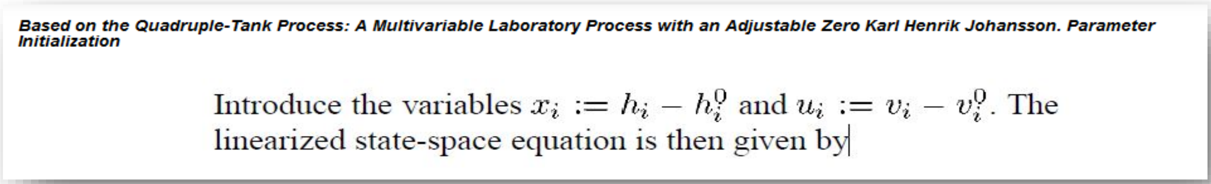
```
process_noise_cov = 8 * np.eye(num_tanks)  # Process noise covariance
print(f'process_noise_cov : {process_noise_cov}')
X_post_list_loop, X_prior_list_loop, P_post_list_loop, P_pri_list_loop, Kalman_gain_list_loop, Measurement_error_loc
```

```
process_noise_cov : [[8. 0. 0. 0.]
 [0. 8. 0. 0.]
 [0. 0. 8. 0.]
 [0. 0. 0. 8.]]
Posterior states: [12.30368913 12.81521076  0.11720097  0.07689637]
```

Looking at the measurement I can see the convergence happen nicely for Tank 1 and Tank 2, but Tank 3 and Tank 4 is bit far.

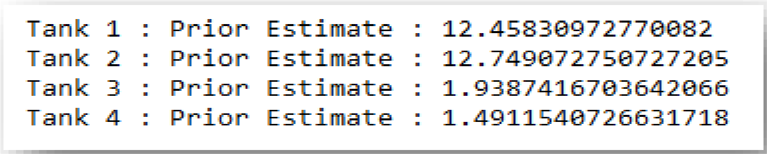
Step2 - Now let’s try to observe the plots:

Prior Plots:

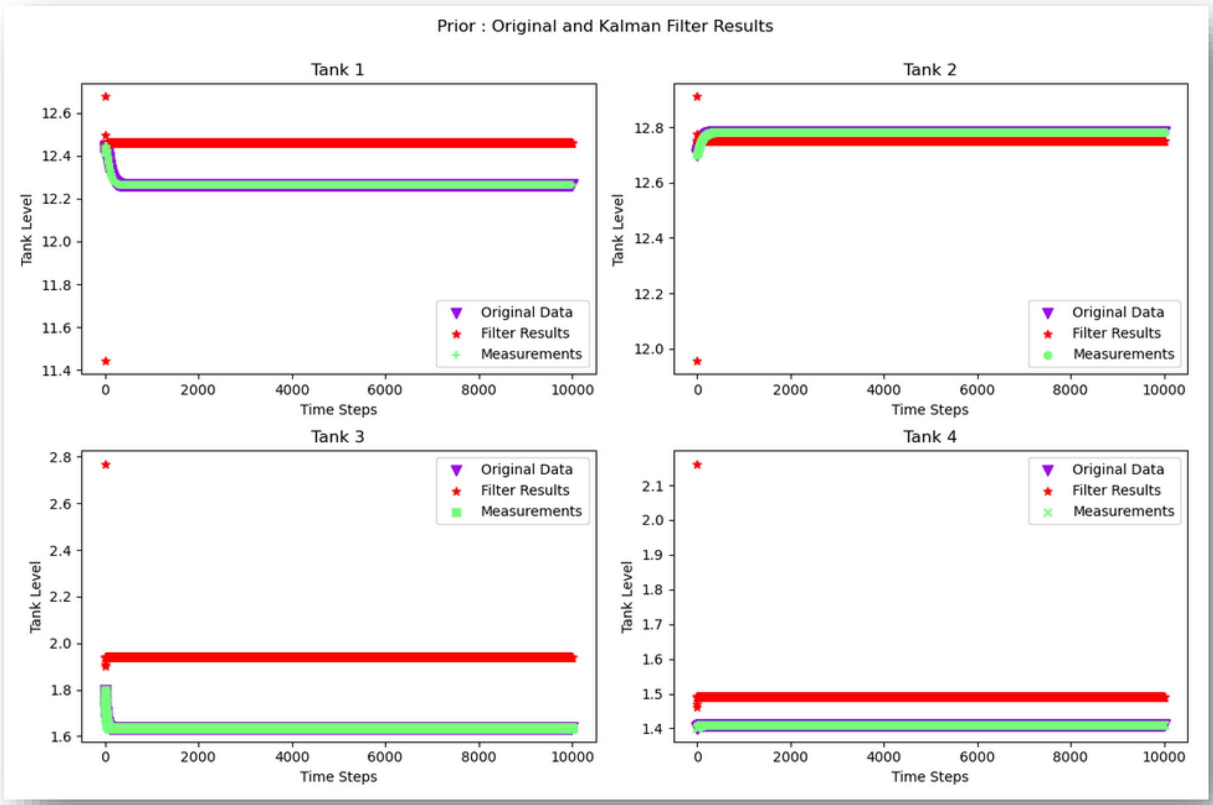

Based on the Quadruple-Tank Process: A Multivariable Laboratory Process with an Adjustable Zero Karl Henrik Johansson. Parameter Initialization

Introduce the variables $x_i := h_i - h_i^0$ and $u_i := v_i - v_i^0$. The linearized state-space equation is then given by

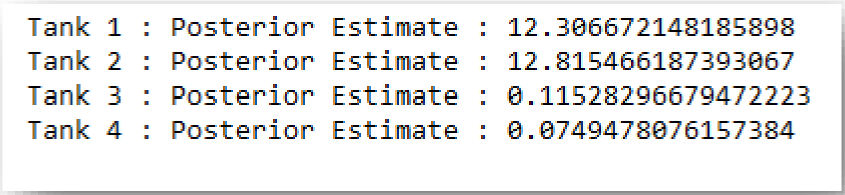
Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503



```
Tank 1 : Prior Estimate : 12.45830972770082
Tank 2 : Prior Estimate : 12.749072750727205
Tank 3 : Prior Estimate : 1.9387416703642066
Tank 4 : Prior Estimate : 1.4911540726631718
```

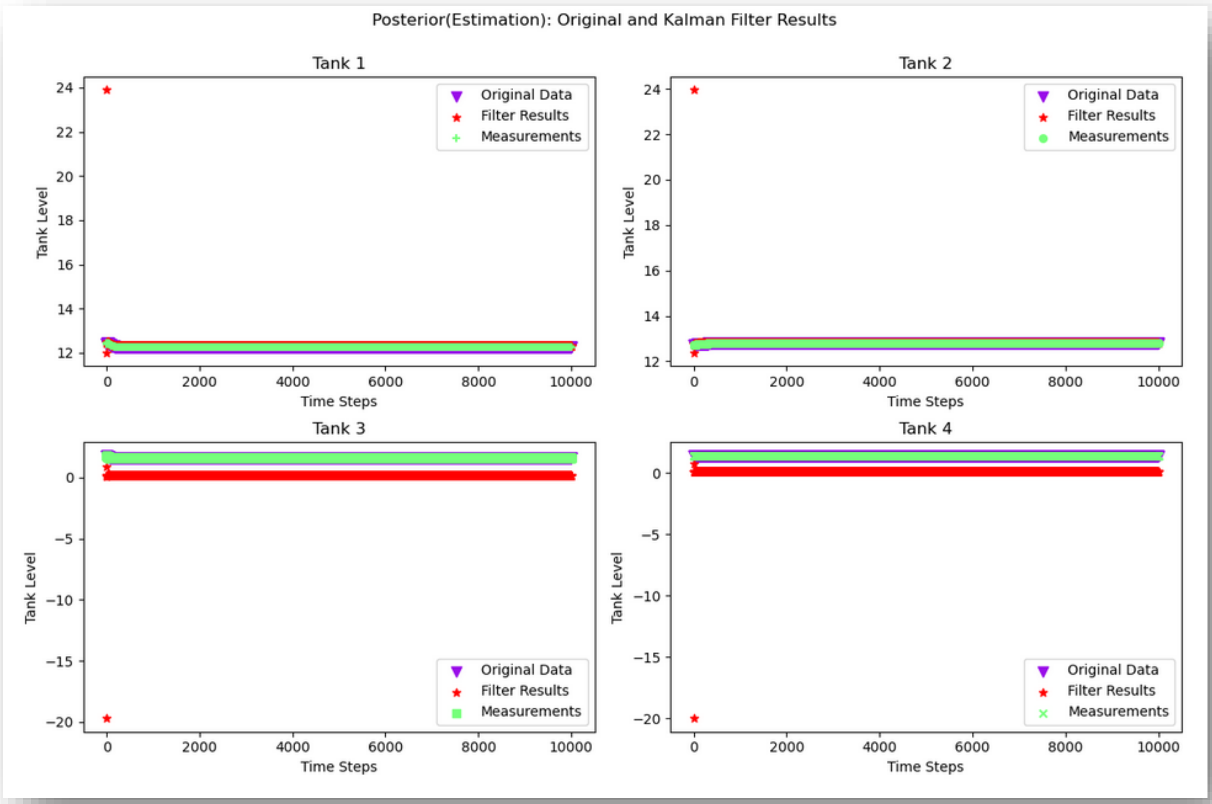


Posterior Plots:



```
Tank 1 : Posterior Estimate : 12.306672148185898
Tank 2 : Posterior Estimate : 12.815466187393067
Tank 3 : Posterior Estimate : 0.11528296679472223
Tank 4 : Posterior Estimate : 0.0749478076157384
```

Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503

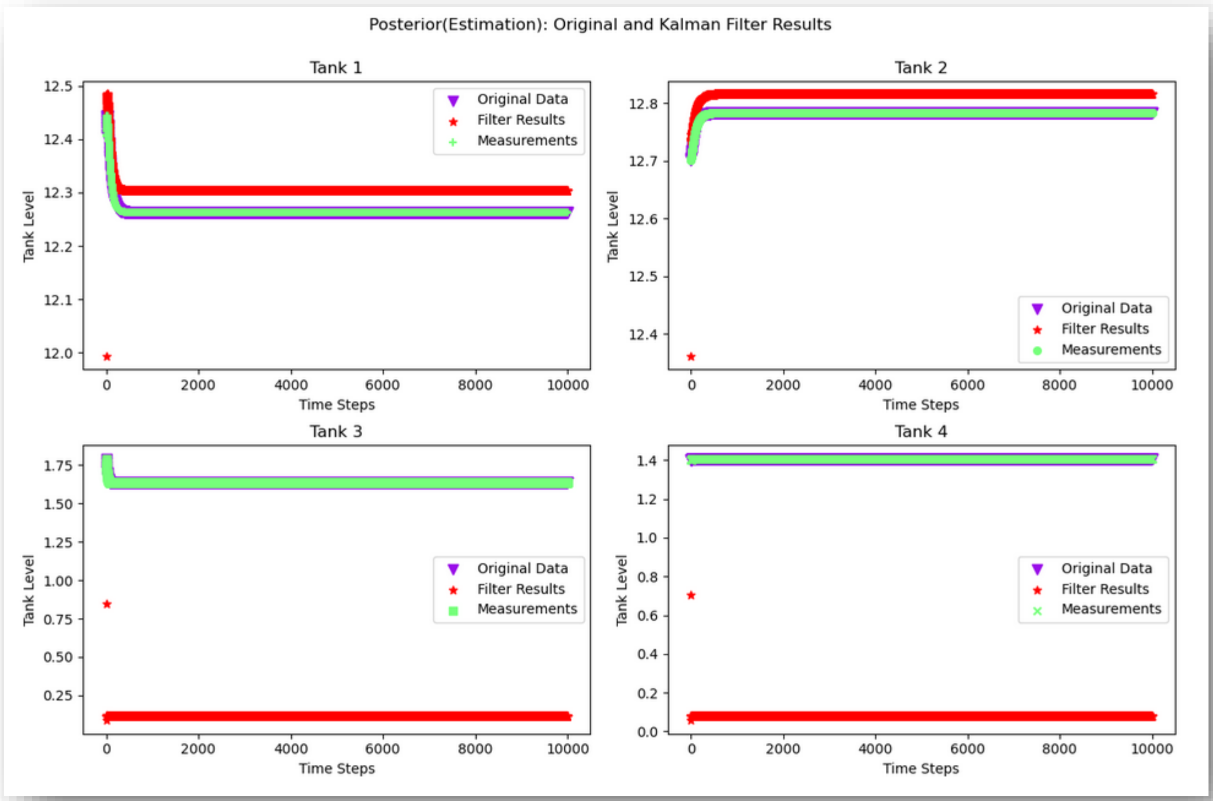


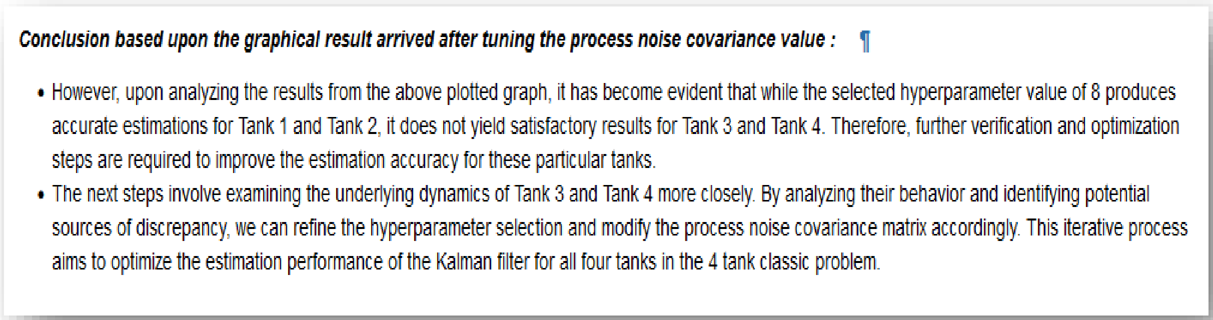
Posterior plot after removing the first measurement to visualize the variation much closely,

Mini Project – Kalman : Report

Author: Aloy Banerjee

Roll No.: CH22M503

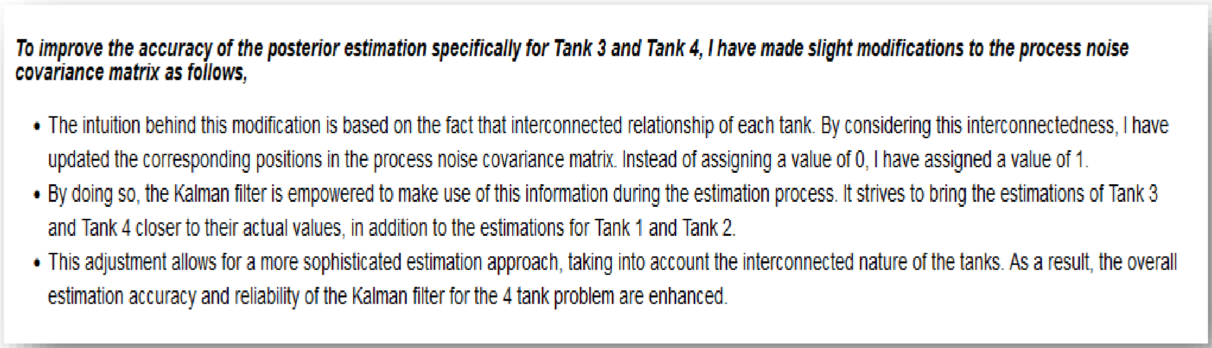




Conclusion based upon the graphical result arrived after tuning the process noise covariance value : ¶

- However, upon analyzing the results from the above plotted graph, it has become evident that while the selected hyperparameter value of 8 produces accurate estimations for Tank 1 and Tank 2, it does not yield satisfactory results for Tank 3 and Tank 4. Therefore, further verification and optimization steps are required to improve the estimation accuracy for these particular tanks.
- The next steps involve examining the underlying dynamics of Tank 3 and Tank 4 more closely. By analyzing their behavior and identifying potential sources of discrepancy, we can refine the hyperparameter selection and modify the process noise covariance matrix accordingly. This iterative process aims to optimize the estimation performance of the Kalman filter for all four tanks in the 4 tank classic problem.

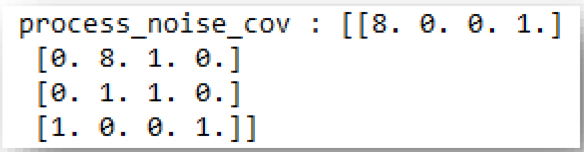
Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503



To improve the accuracy of the posterior estimation specifically for Tank 3 and Tank 4, I have made slight modifications to the process noise covariance matrix as follows,

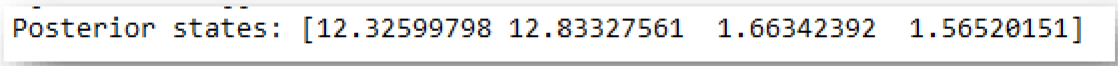
- The intuition behind this modification is based on the fact that interconnected relationship of each tank. By considering this interconnectedness, I have updated the corresponding positions in the process noise covariance matrix. Instead of assigning a value of 0, I have assigned a value of 1.
- By doing so, the Kalman filter is empowered to make use of this information during the estimation process. It strives to bring the estimations of Tank 3 and Tank 4 closer to their actual values, in addition to the estimations for Tank 1 and Tank 2.
- This adjustment allows for a more sophisticated estimation approach, taking into account the interconnected nature of the tanks. As a result, the overall estimation accuracy and reliability of the Kalman filter for the 4 tank problem are enhanced.

Modified process covariance matrix is chosen as below,



```
process_noise_cov : [[8. 0. 0. 1.]
 [0. 8. 1. 0.]
 [0. 1. 1. 0.]
 [1. 0. 0. 1.]]
```

Now again running the Kalman filter and observe the estimation,



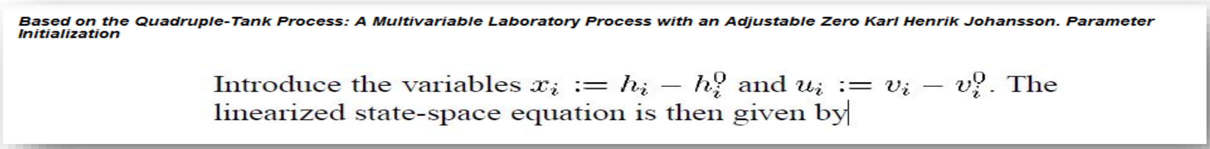
```
Posterior states: [12.32599798 12.83327561  1.66342392  1.56520151]
```

We can see the estimation for all the Tank's height is converging. And same indication has been given in the plots too.

Mini Project – Kalman : Report

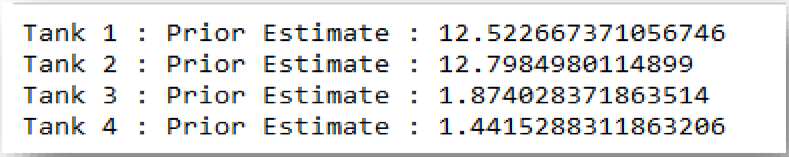
Author: Aloy Banerjee
Roll No.: CH22M503

Prior Plots:

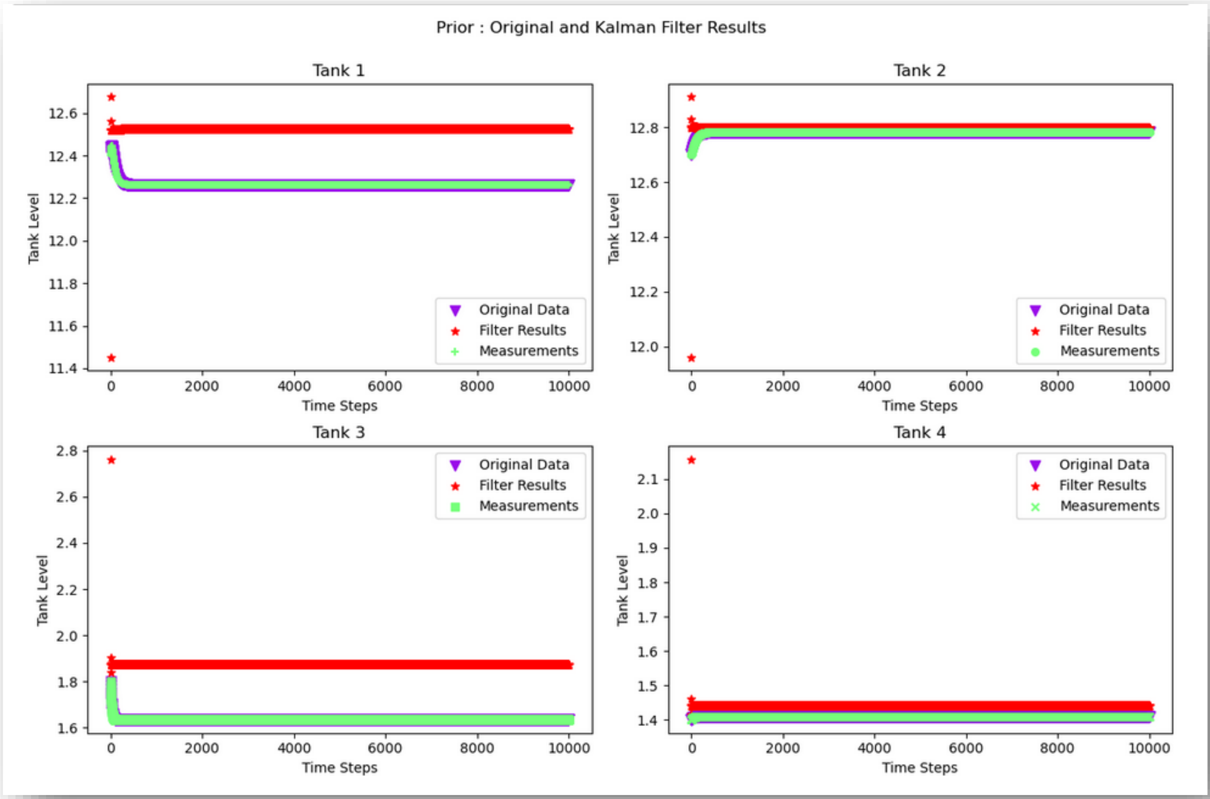


Based on the Quadruple-Tank Process: A Multivariable Laboratory Process with an Adjustable Zero Karl Henrik Johansson. Parameter Initialization

Introduce the variables $x_i := h_i - h_i^0$ and $u_i := v_i - v_i^0$. The linearized state-space equation is then given by



```
Tank 1 : Prior Estimate : 12.522667371056746
Tank 2 : Prior Estimate : 12.7984980114899
Tank 3 : Prior Estimate : 1.874028371863514
Tank 4 : Prior Estimate : 1.4415288311863206
```

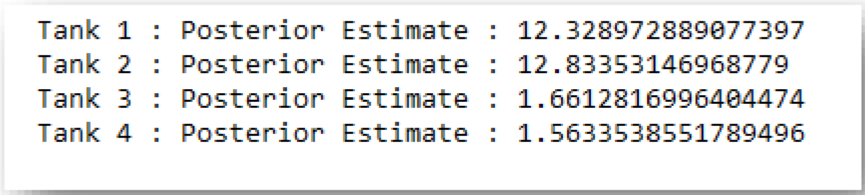


Posterior Plots:

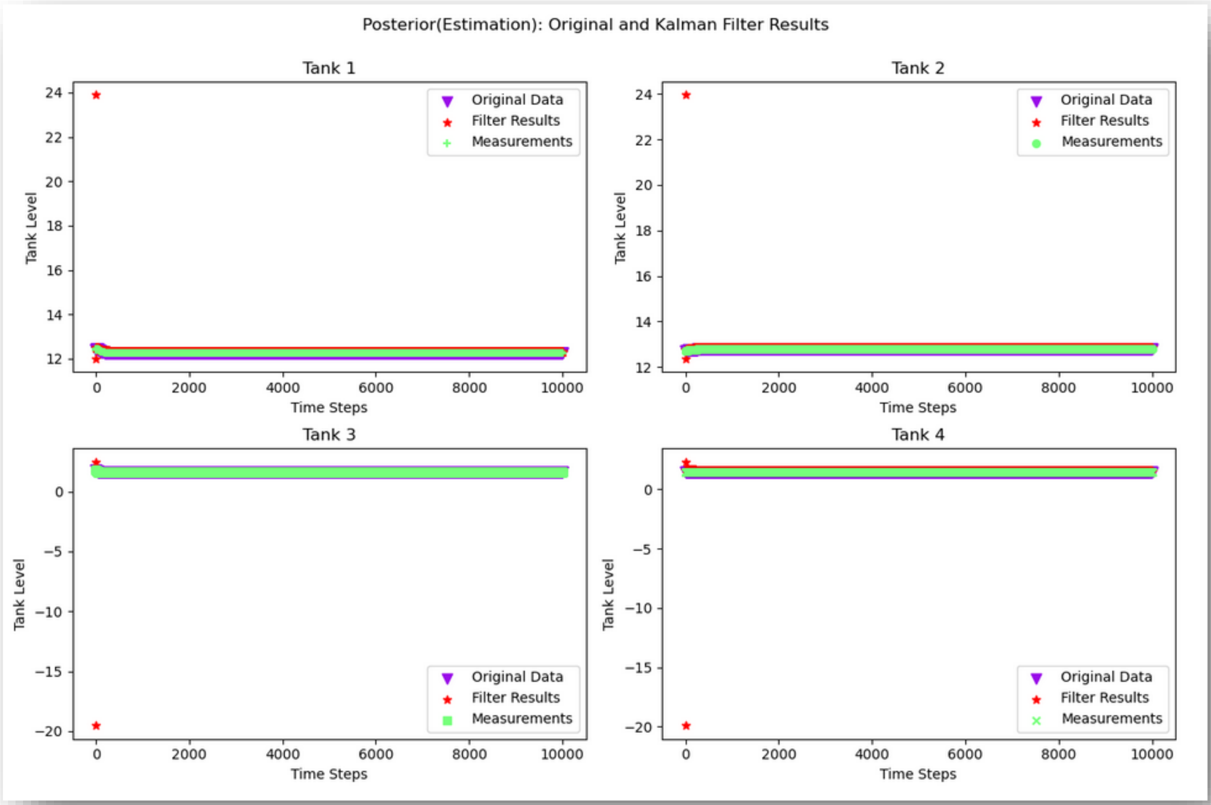
Mini Project – Kalman : Report

Author: Aloy Banerjee

Roll No.: CH22M503

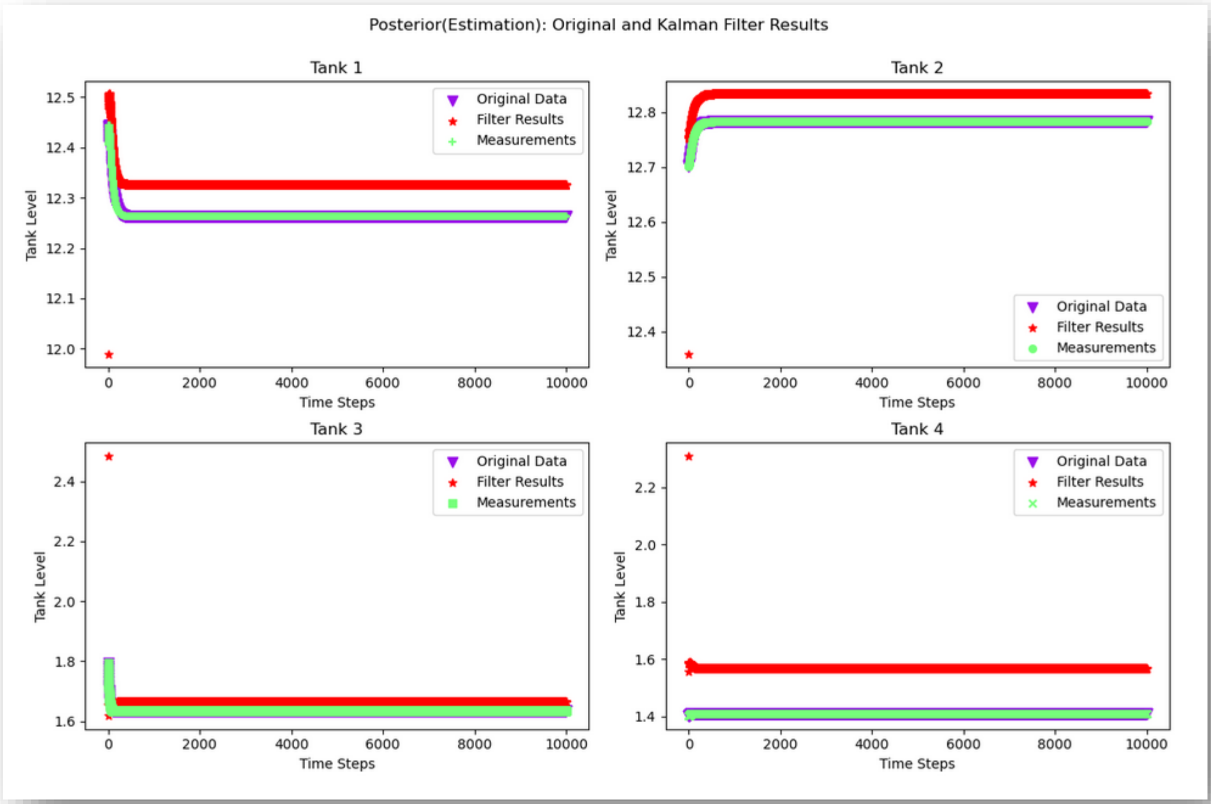


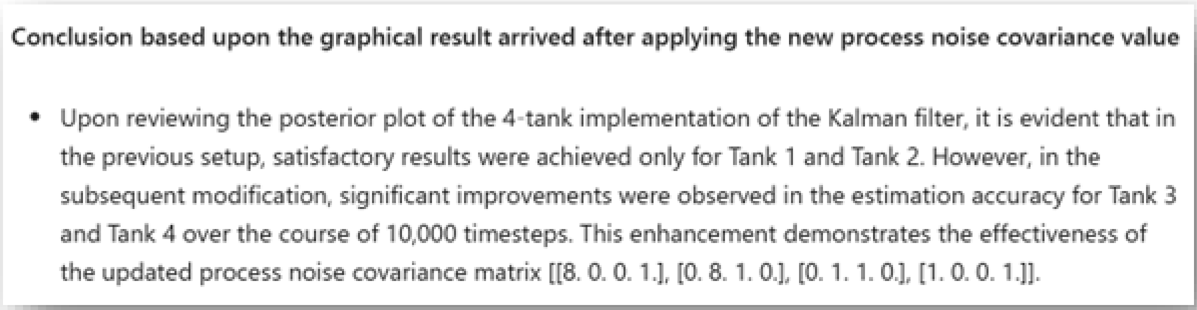
```
Tank 1 : Posterior Estimate : 12.328972889077397
Tank 2 : Posterior Estimate : 12.83353146968779
Tank 3 : Posterior Estimate : 1.6612816996404474
Tank 4 : Posterior Estimate : 1.5633538551789496
```



Posterior plot after removing the first measurement to visualize the variation much closely,

Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503

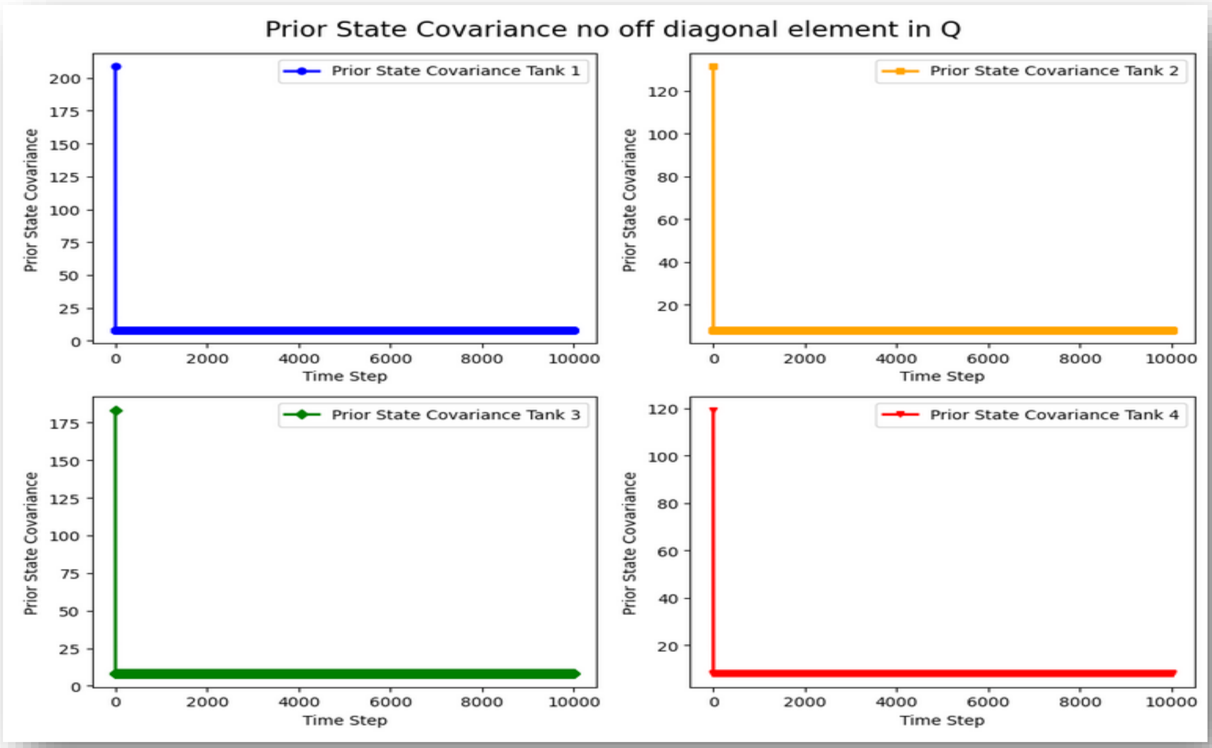


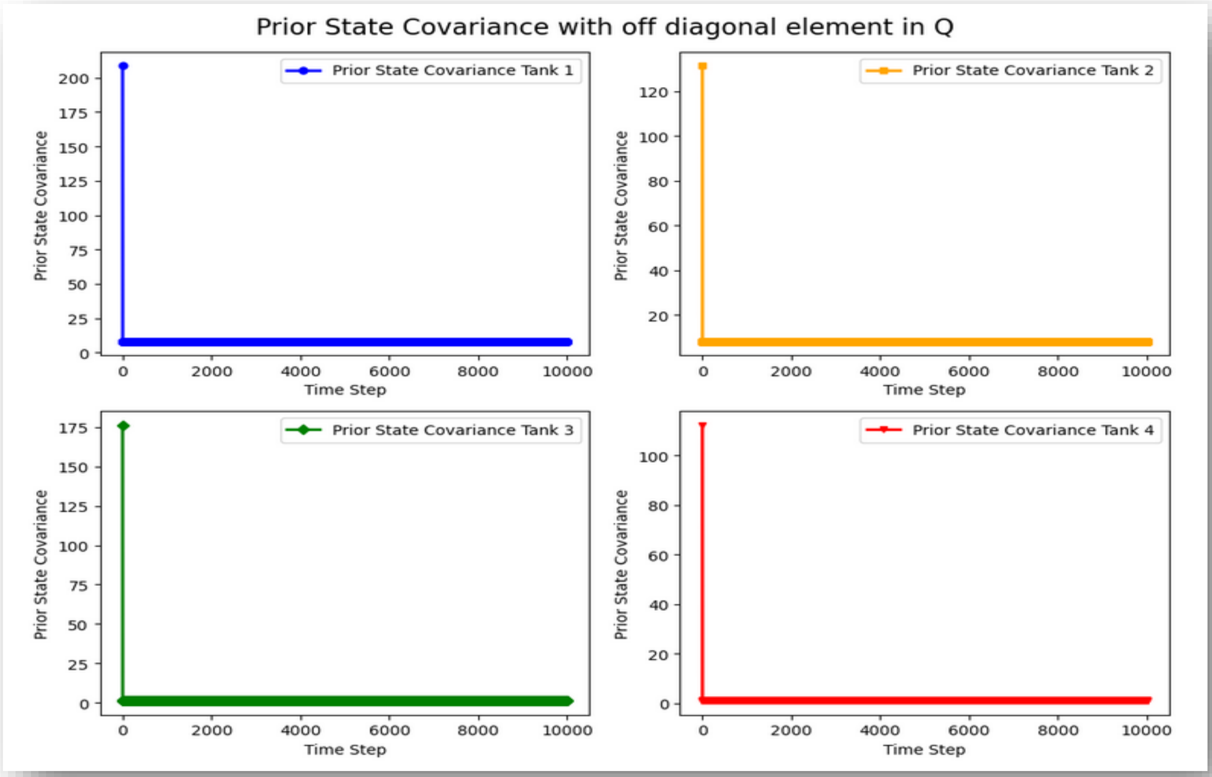

Conclusion based upon the graphical result arrived after applying the new process noise covariance value

- Upon reviewing the posterior plot of the 4-tank implementation of the Kalman filter, it is evident that in the previous setup, satisfactory results were achieved only for Tank 1 and Tank 2. However, in the subsequent modification, significant improvements were observed in the estimation accuracy for Tank 3 and Tank 4 over the course of 10,000 timesteps. This enhancement demonstrates the effectiveness of the updated process noise covariance matrix [[8. 0. 0. 1.], [0. 8. 1. 0.], [0. 1. 1. 0.], [1. 0. 0. 1.]].

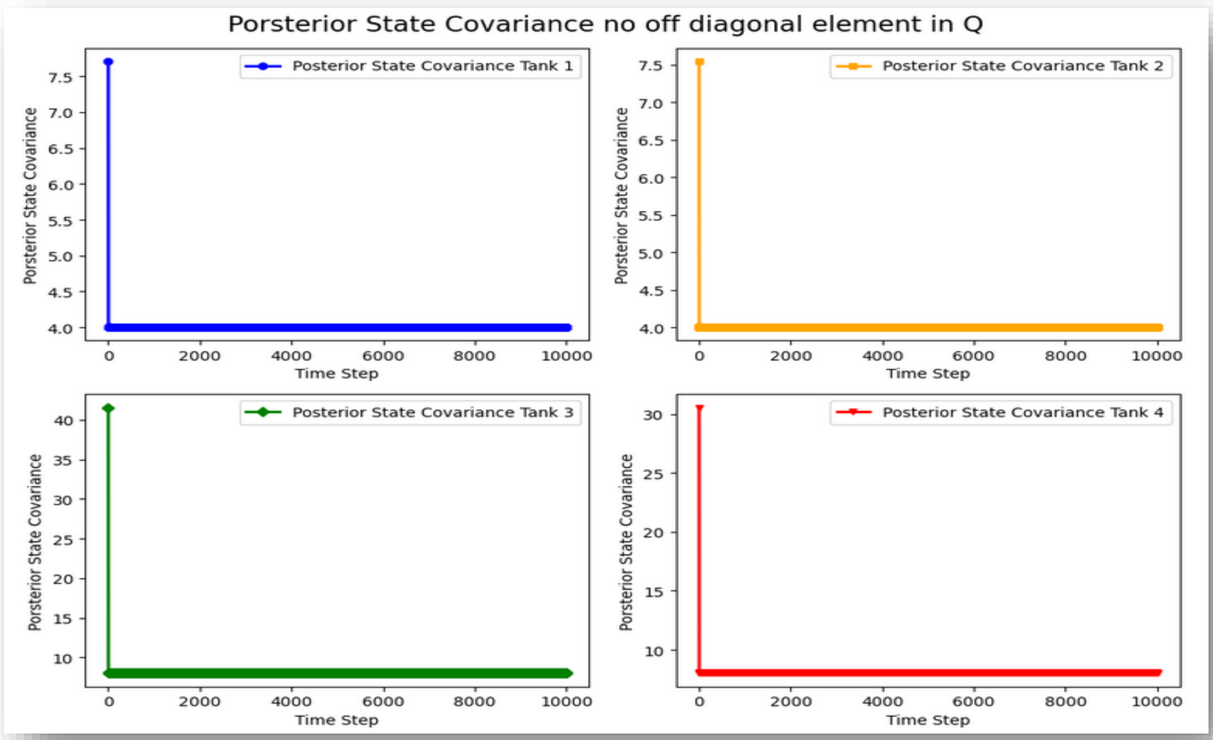
Different graphical plots for Kalman Filter using 1st Approach

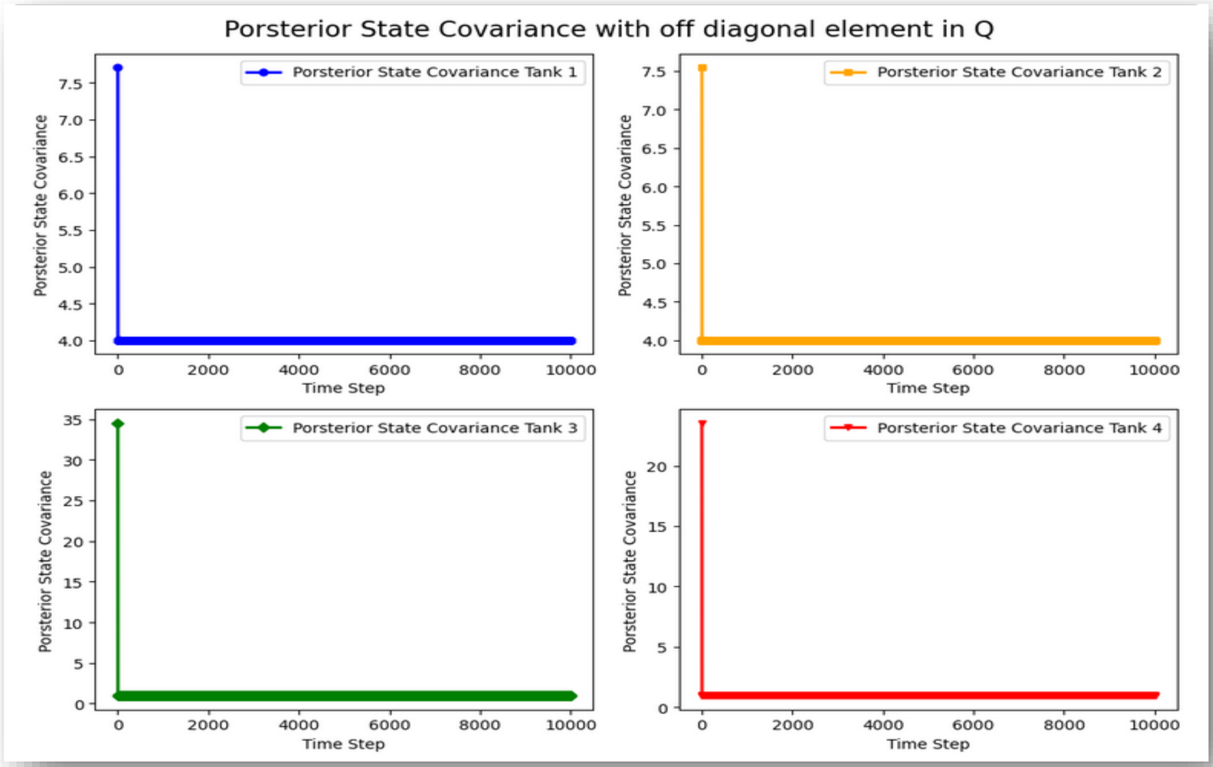
Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503



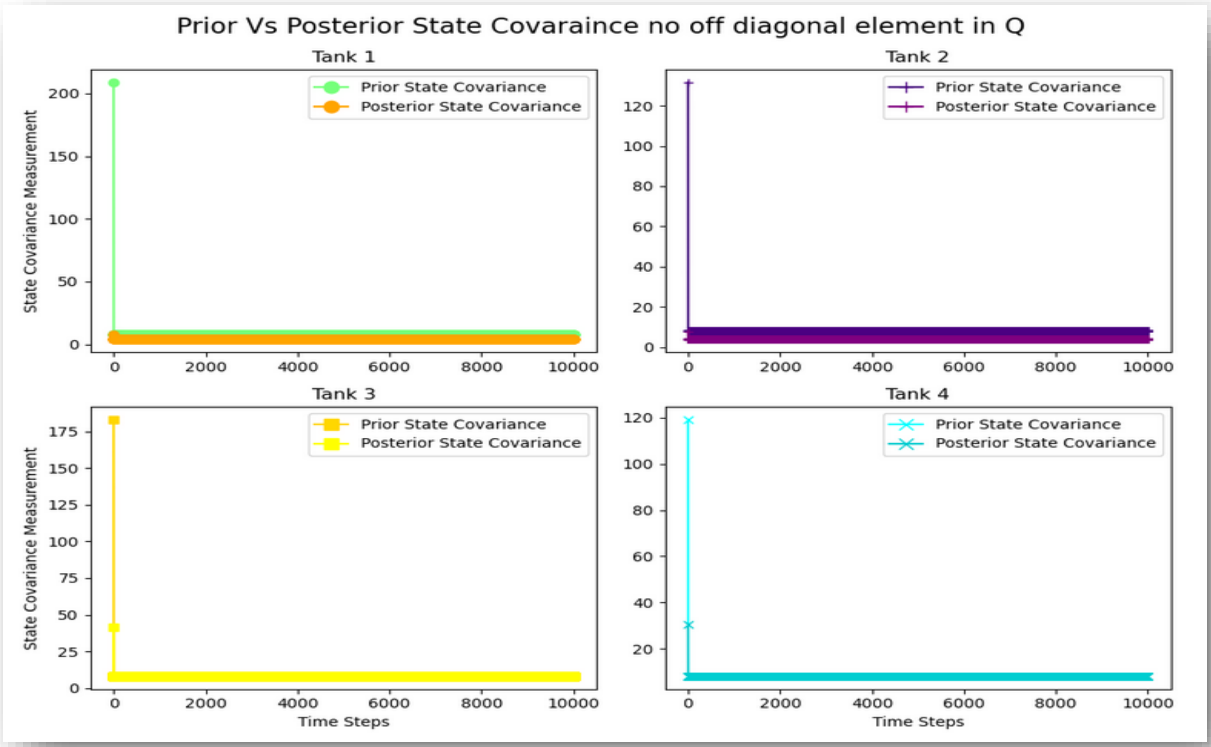


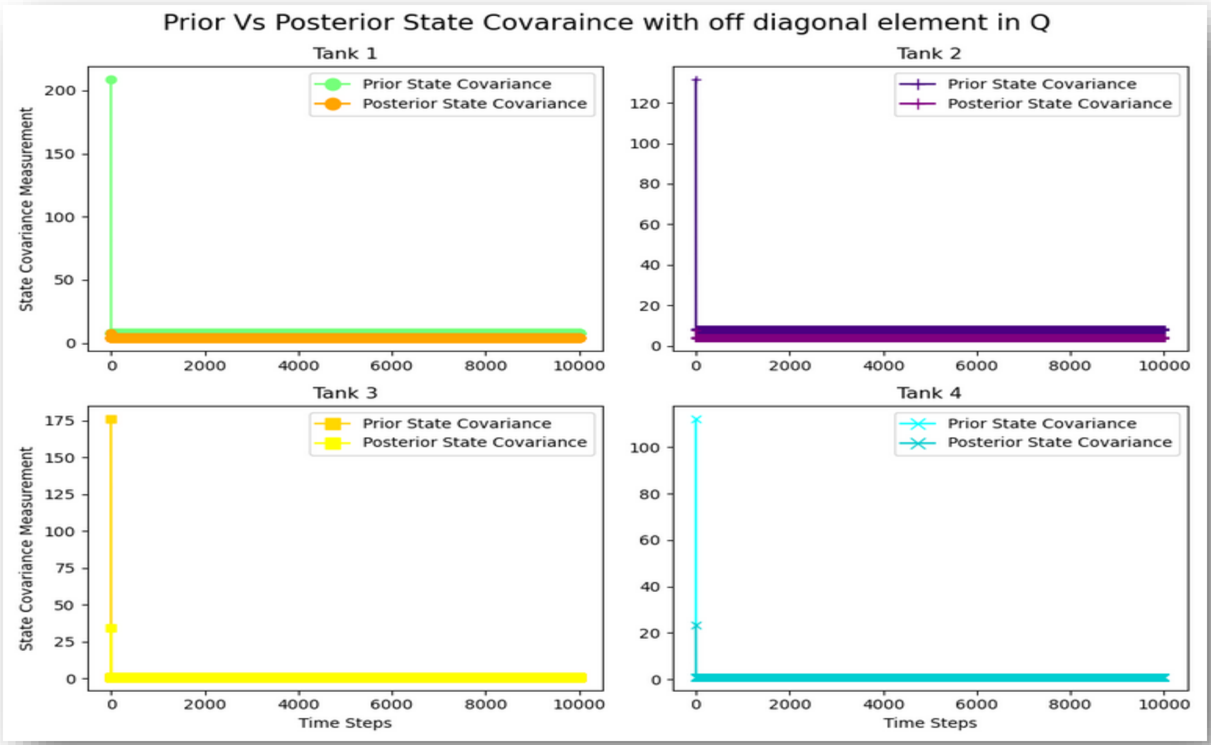
Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503





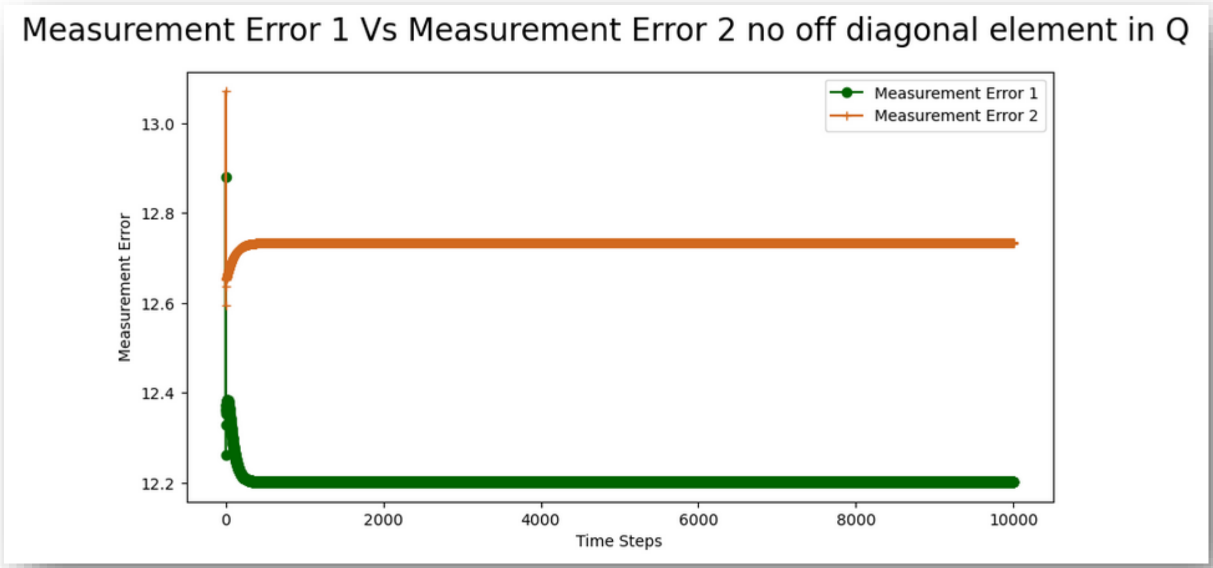
Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503

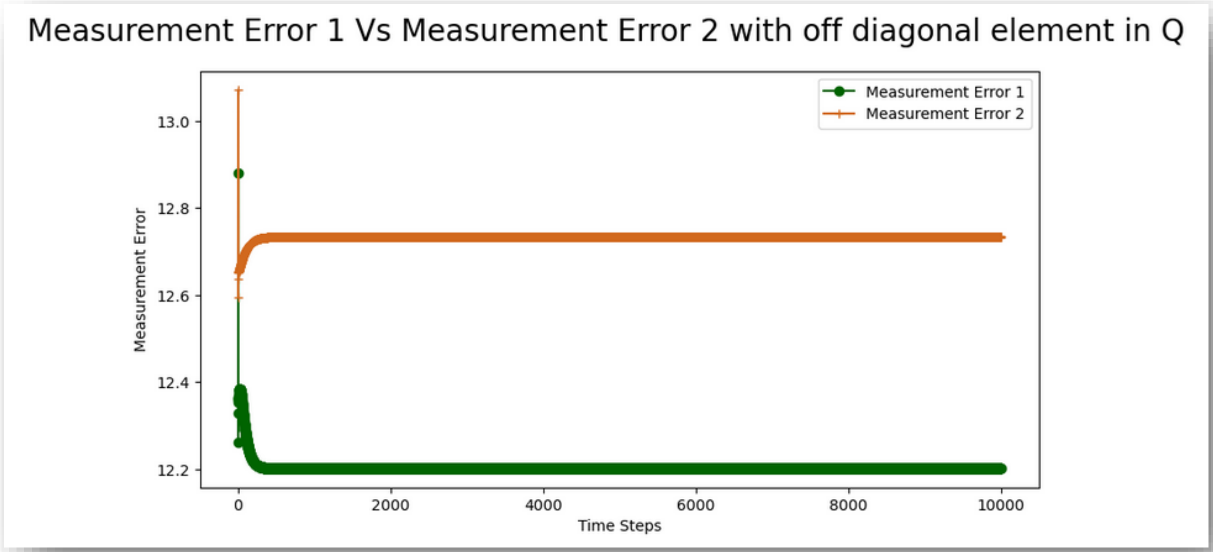




Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503

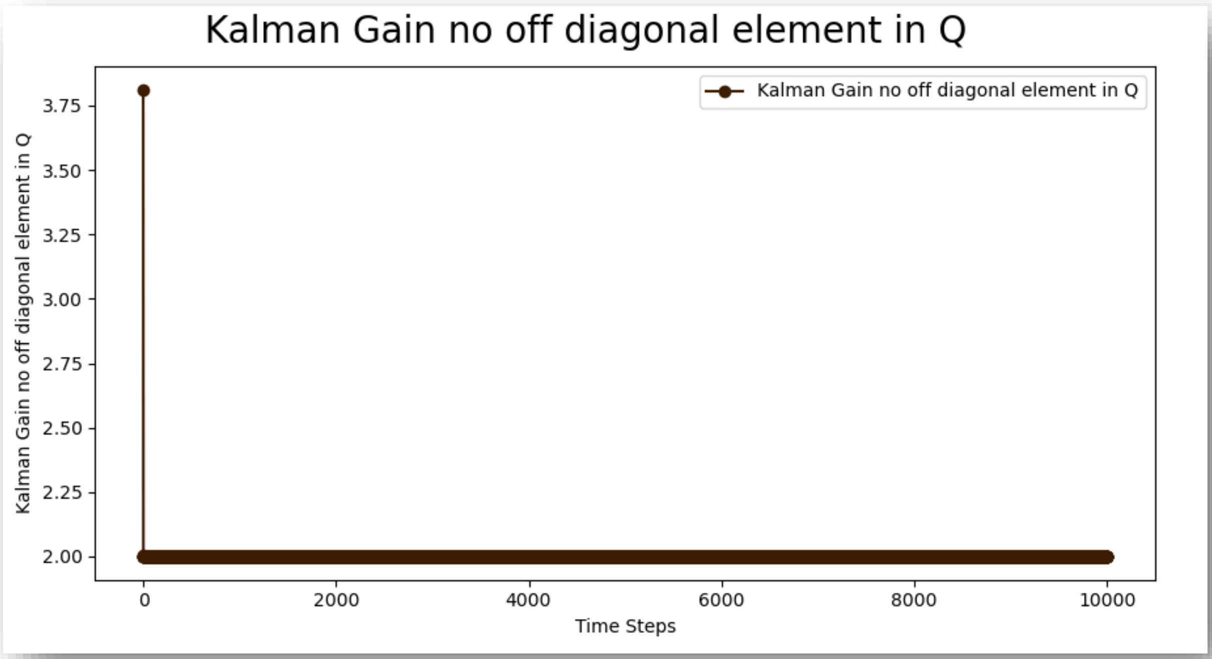


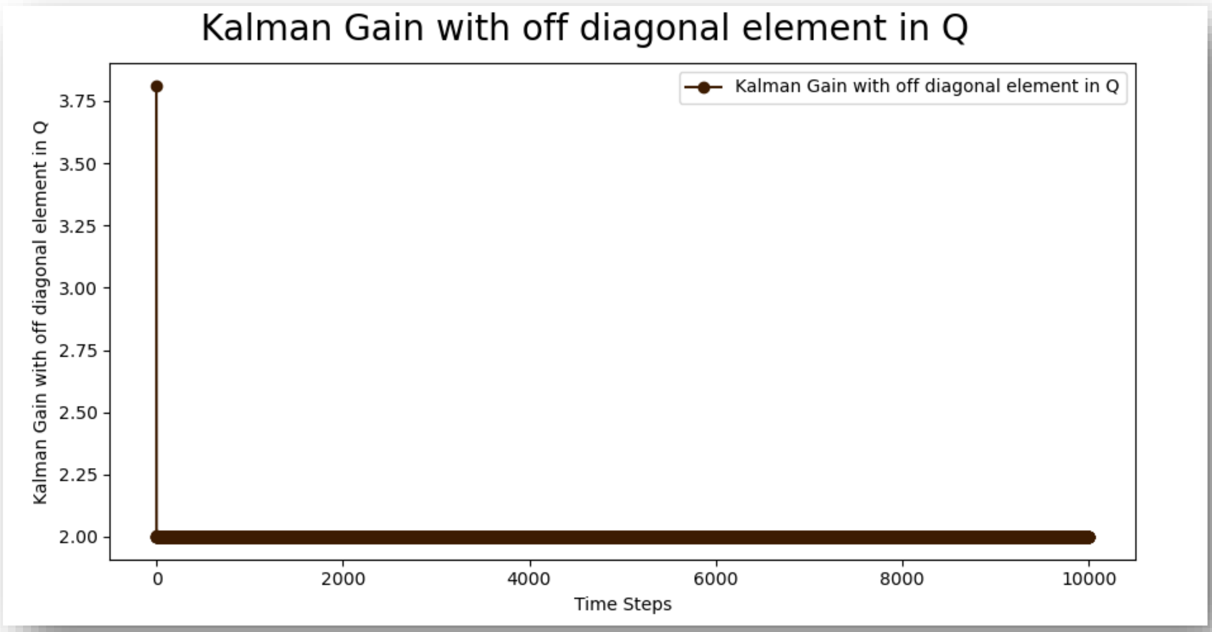


Mini Project – Kalman : Report

Author: Aloy Banerjee

Roll No.: CH22M503

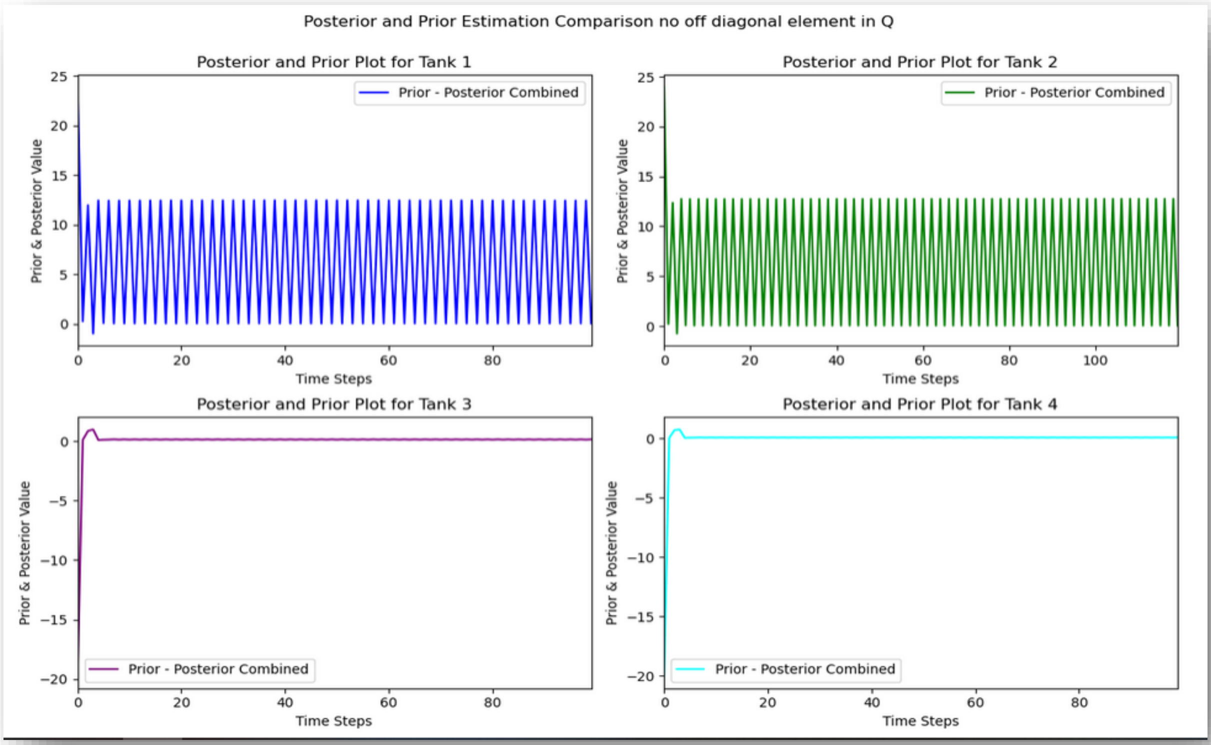


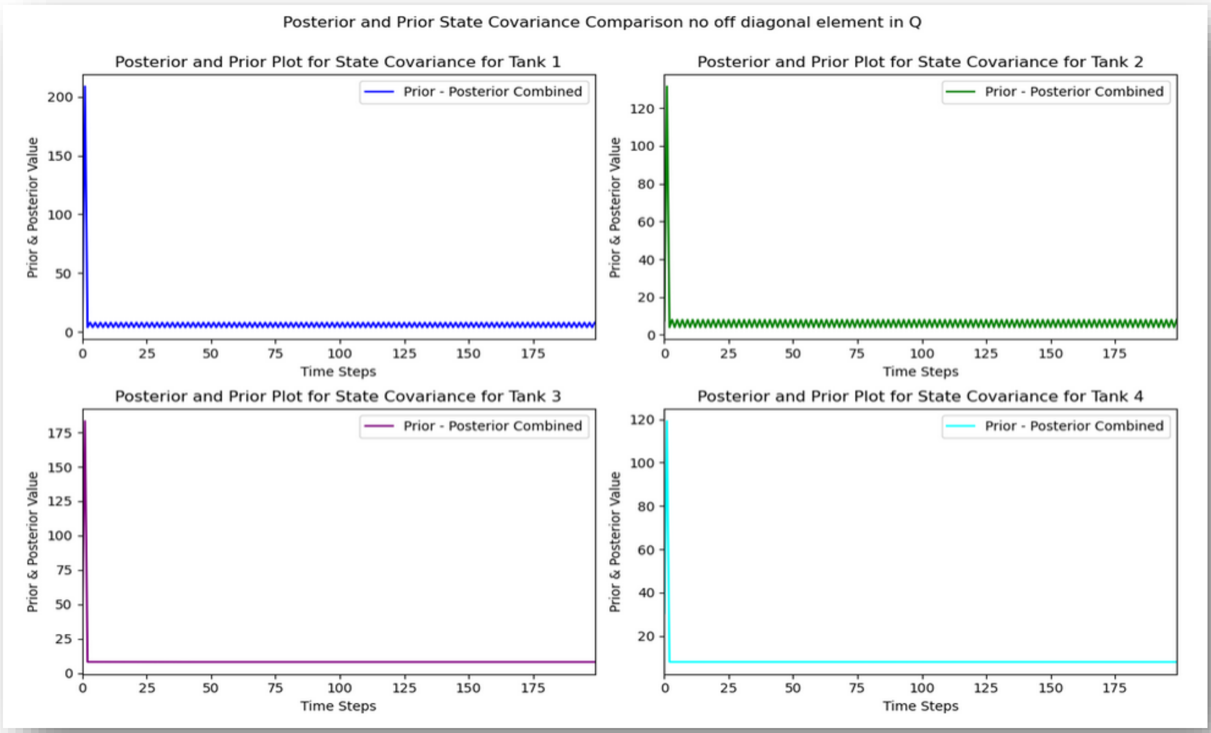


Mini Project – Kalman : Report

Author: Aloy Banerjee

Roll No.: CH22M503

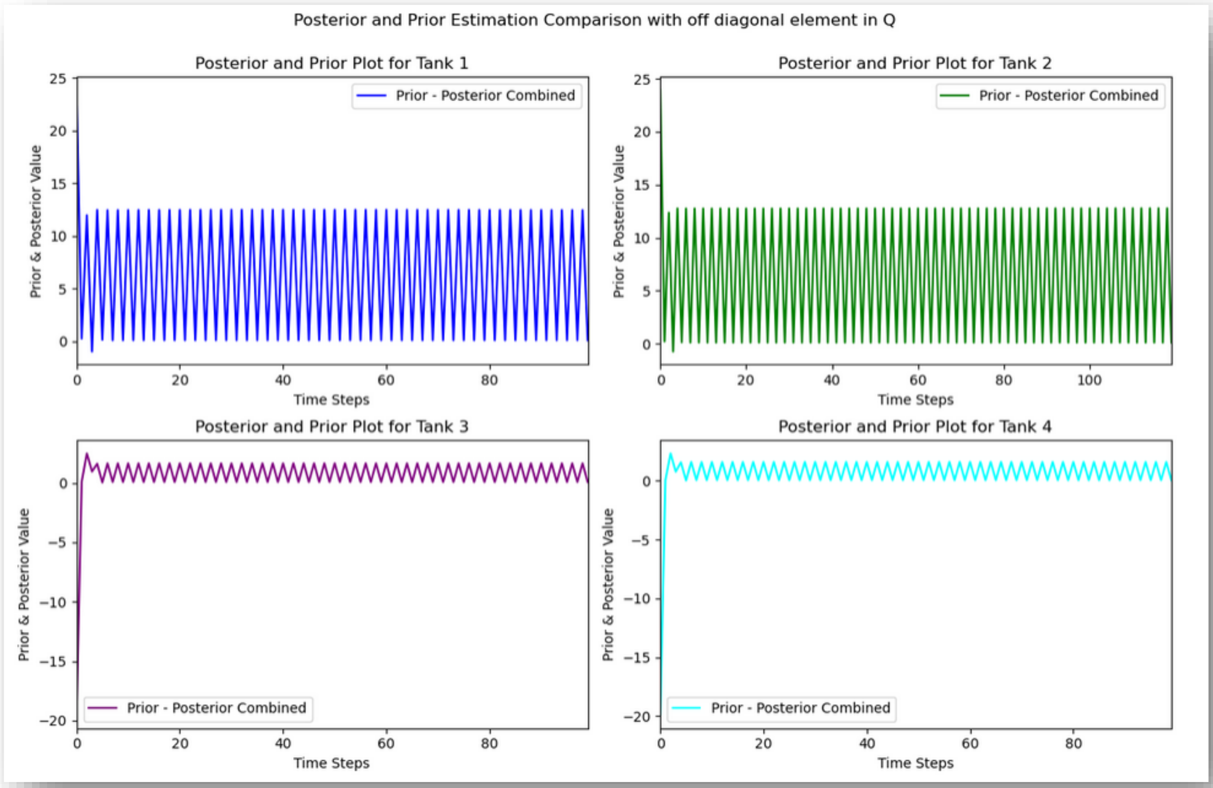


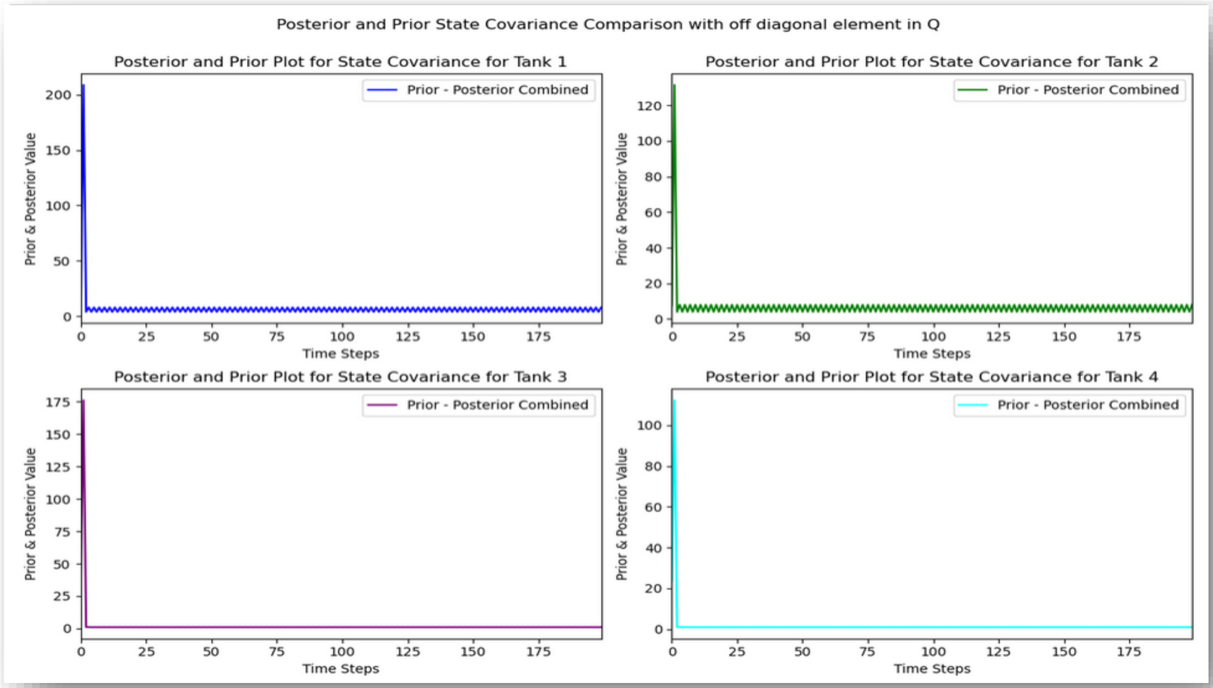


Mini Project – Kalman : Report

Author: Aloy Banerjee

Roll No.: CH22M503

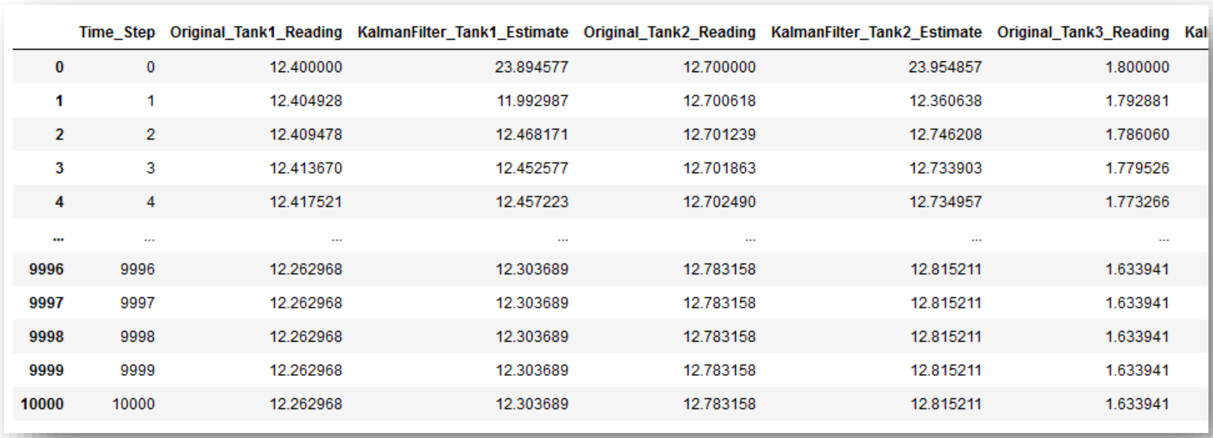




Mini Project – Kalman : Report

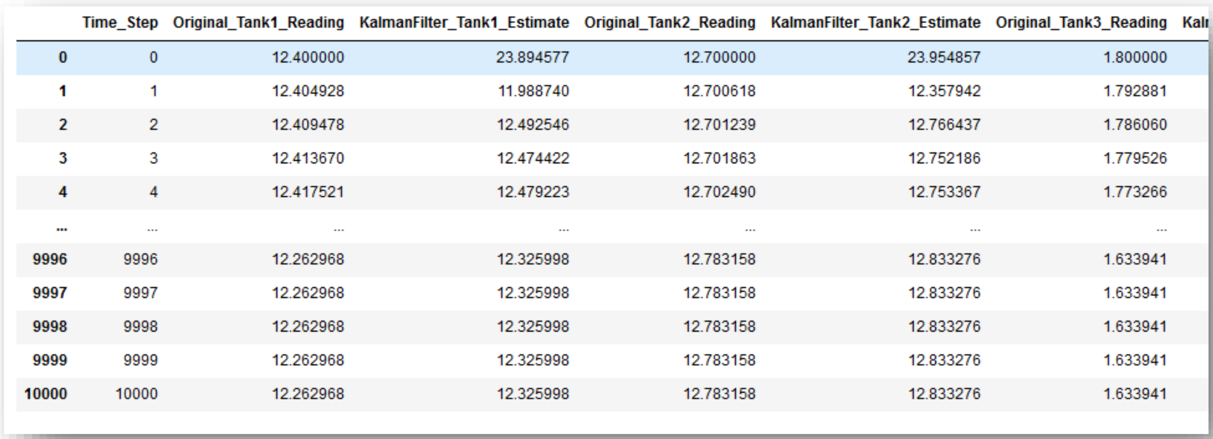
Author: Aloy Banerjee
Roll No.: CH22M503

Kalman Filter Result - 1st Approach with no off diagonal element in Q



	Time_Step	Original_Tank1_Reading	KalmanFilter_Tank1_Estimate	Original_Tank2_Reading	KalmanFilter_Tank2_Estimate	Original_Tank3_Reading	Kal
0	0	12.400000	23.894577	12.700000	23.954857	1.800000	
1	1	12.404928	11.992987	12.700618	12.360638	1.792881	
2	2	12.409478	12.468171	12.701239	12.746208	1.786060	
3	3	12.413670	12.452577	12.701863	12.733903	1.779526	
4	4	12.417521	12.457223	12.702490	12.734957	1.773266	
...	...	...	...	...	...	...	
9996	9996	12.262968	12.303689	12.783158	12.815211	1.633941	
9997	9997	12.262968	12.303689	12.783158	12.815211	1.633941	
9998	9998	12.262968	12.303689	12.783158	12.815211	1.633941	
9999	9999	12.262968	12.303689	12.783158	12.815211	1.633941	
10000	10000	12.262968	12.303689	12.783158	12.815211	1.633941	

Kalman Filter Result - 1st Approach with off diagonal element in Q



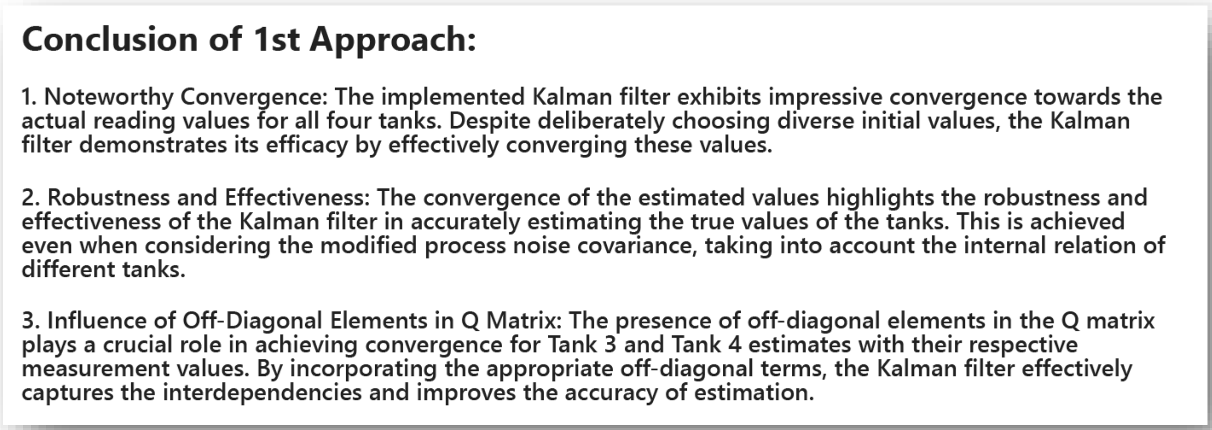
	Time_Step	Original_Tank1_Reading	KalmanFilter_Tank1_Estimate	Original_Tank2_Reading	KalmanFilter_Tank2_Estimate	Original_Tank3_Reading	Kal
0	0	12.400000	23.894577	12.700000	23.954857	1.800000	
1	1	12.404928	11.988740	12.700618	12.357942	1.792881	
2	2	12.409478	12.492546	12.701239	12.766437	1.786060	
3	3	12.413670	12.474422	12.701863	12.752186	1.779526	
4	4	12.417521	12.479223	12.702490	12.753367	1.773266	
...	...	...	...	...	...	...	
9996	9996	12.262968	12.325998	12.783158	12.833276	1.633941	
9997	9997	12.262968	12.325998	12.783158	12.833276	1.633941	
9998	9998	12.262968	12.325998	12.783158	12.833276	1.633941	
9999	9999	12.262968	12.325998	12.783158	12.833276	1.633941	
10000	10000	12.262968	12.325998	12.783158	12.833276	1.633941	

Comparison_Kalma
n_Approach1.csv

Figure 1: Result for Kalman Gain with no off-diagonal element in Q

Comparison_Kalman_Approach1_Q.csv

Figure 2: Result for Kalman Gain with off-diagonal element in Q



Conclusion of 1st Approach:

1. Noteworthy Convergence: The implemented Kalman filter exhibits impressive convergence towards the actual reading values for all four tanks. Despite deliberately choosing diverse initial values, the Kalman filter demonstrates its efficacy by effectively converging these values.

2. Robustness and Effectiveness: The convergence of the estimated values highlights the robustness and effectiveness of the Kalman filter in accurately estimating the true values of the tanks. This is achieved even when considering the modified process noise covariance, taking into account the internal relation of different tanks.

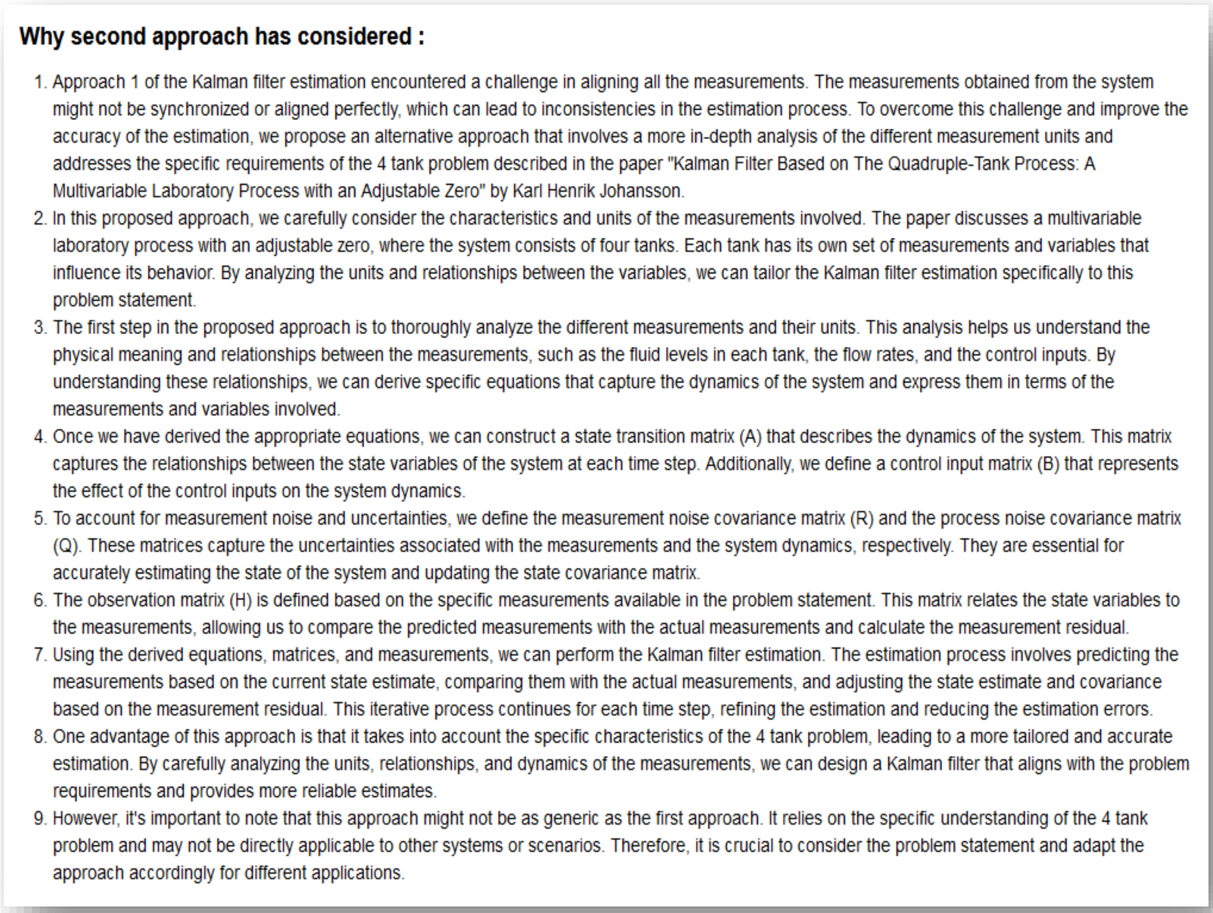
3. Influence of Off-Diagonal Elements in Q Matrix: The presence of off-diagonal elements in the Q matrix plays a crucial role in achieving convergence for Tank 3 and Tank 4 estimates with their respective measurement values. By incorporating the appropriate off-diagonal terms, the Kalman filter effectively captures the interdependencies and improves the accuracy of estimation.

<u>Implementation of Kalman Filter using 2nd approach</u>

Though we have received good convergence with varying the process and measurement noise covariance matrix but still due to below problem we have move to approach 2.

Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503

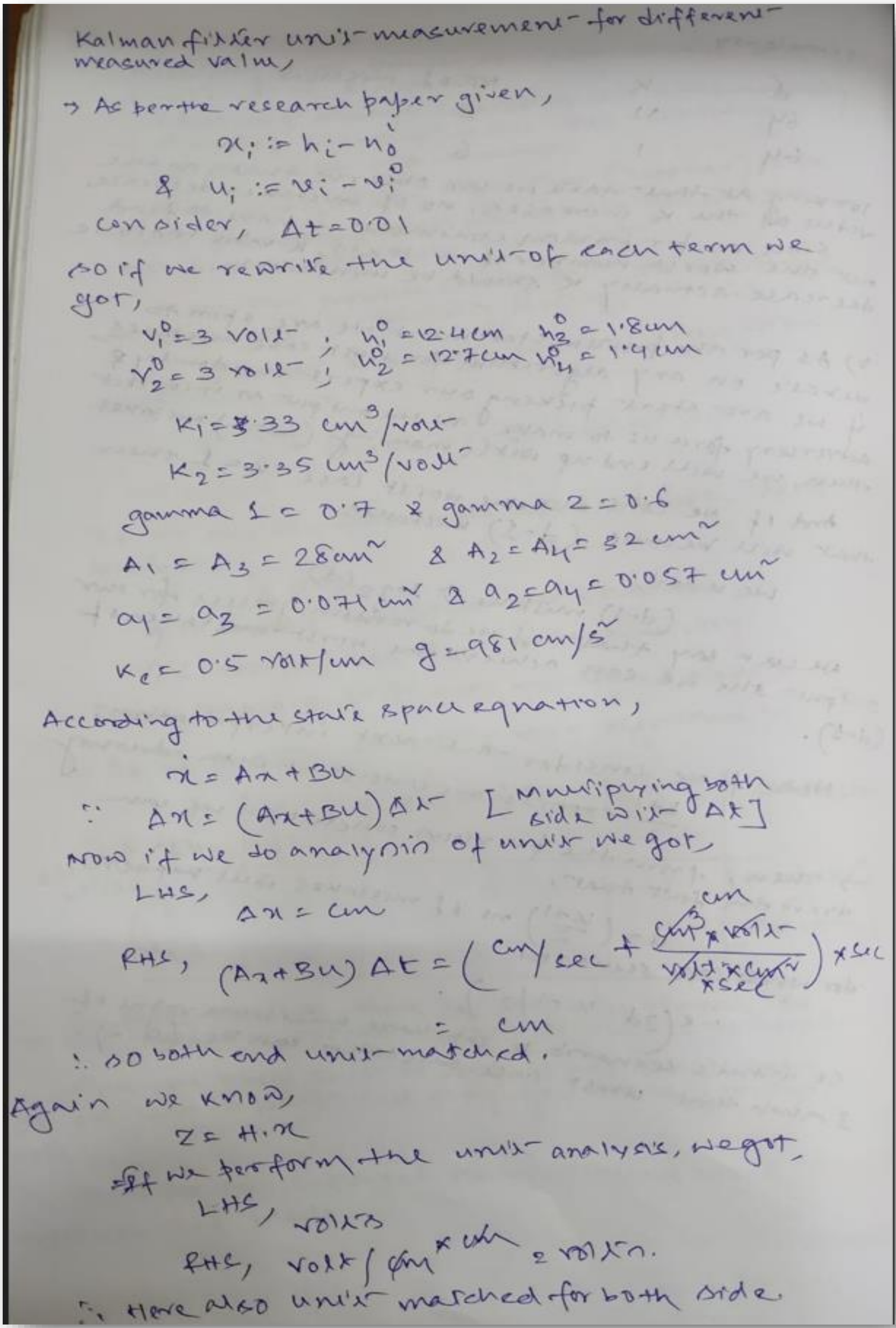


Why second approach has considered :

1. Approach 1 of the Kalman filter estimation encountered a challenge in aligning all the measurements. The measurements obtained from the system might not be synchronized or aligned perfectly, which can lead to inconsistencies in the estimation process. To overcome this challenge and improve the accuracy of the estimation, we propose an alternative approach that involves a more in-depth analysis of the different measurement units and addresses the specific requirements of the 4 tank problem described in the paper "Kalman Filter Based on The Quadruple-Tank Process: A Multivariable Laboratory Process with an Adjustable Zero" by Karl Henrik Johansson.
2. In this proposed approach, we carefully consider the characteristics and units of the measurements involved. The paper discusses a multivariable laboratory process with an adjustable zero, where the system consists of four tanks. Each tank has its own set of measurements and variables that influence its behavior. By analyzing the units and relationships between the variables, we can tailor the Kalman filter estimation specifically to this problem statement.
3. The first step in the proposed approach is to thoroughly analyze the different measurements and their units. This analysis helps us understand the physical meaning and relationships between the measurements, such as the fluid levels in each tank, the flow rates, and the control inputs. By understanding these relationships, we can derive specific equations that capture the dynamics of the system and express them in terms of the measurements and variables involved.
4. Once we have derived the appropriate equations, we can construct a state transition matrix (A) that describes the dynamics of the system. This matrix captures the relationships between the state variables of the system at each time step. Additionally, we define a control input matrix (B) that represents the effect of the control inputs on the system dynamics.
5. To account for measurement noise and uncertainties, we define the measurement noise covariance matrix (R) and the process noise covariance matrix (Q). These matrices capture the uncertainties associated with the measurements and the system dynamics, respectively. They are essential for accurately estimating the state of the system and updating the state covariance matrix.
6. The observation matrix (H) is defined based on the specific measurements available in the problem statement. This matrix relates the state variables to the measurements, allowing us to compare the predicted measurements with the actual measurements and calculate the measurement residual.
7. Using the derived equations, matrices, and measurements, we can perform the Kalman filter estimation. The estimation process involves predicting the measurements based on the current state estimate, comparing them with the actual measurements, and adjusting the state estimate and covariance based on the measurement residual. This iterative process continues for each time step, refining the estimation and reducing the estimation errors.
8. One advantage of this approach is that it takes into account the specific characteristics of the 4 tank problem, leading to a more tailored and accurate estimation. By carefully analyzing the units, relationships, and dynamics of the measurements, we can design a Kalman filter that aligns with the problem requirements and provides more reliable estimates.
9. However, it's important to note that this approach might not be as generic as the first approach. It relies on the specific understanding of the 4 tank problem and may not be directly applicable to other systems or scenarios. Therefore, it is crucial to consider the problem statement and adapt the approach accordingly for different applications.

Before going into simulation of Kalman filter implementation using second approach, briefly explain the unit of measurement of different parameters,

Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503



Kalman filter unit measurement for different measured value,

→ As per the research paper given,

$$x_i := h_i - h_0^i$$
$$\& \quad u_i := v_i - v_i^0$$

consider, $\Delta t = 0.01$

so if we rewrite the unit of each term we got,

$v_1^0 = 3$ volt ; $h_1^0 = 12.4$ cm $h_3^0 = 1.8$ cm
$v_2^0 = 3$ volt ; $h_2^0 = 12.7$ cm $h_4^0 = 1.4$ cm

$K_1 = ~~3~~.33$ cm^3/volt
$K_2 = 3.35$ cm^3/volt

gamma 1 = 0.7 & gamma 2 = 0.6

$A_1 = A_3 = 28$ cm^2 & $A_2 = A_4 = 32$ cm^2

$a_1 = a_3 = 0.071$ cm^2 & $a_2 = a_4 = 0.057$ cm^2

$K_c = 0.5$ volt/cm $g = 981$ cm/s^2

According to the state space equation,

$$\dot{x} = Ax + Bu$$
$$\therefore \Delta x = (Ax + Bu)\Delta t \quad \text{[Multiplying both side with } \Delta t]$$

now if we do analysis of unit we got,

LHS, $\Delta x = \text{cm}$

RHS, $$(Ax + Bu)\Delta t = \left(\text{cm/sec} + \frac{\overset{\text{cm}}{\cancel{\text{cm}^3}} \times \cancel{\text{volt}}}{\cancel{\text{volt}} \times \cancel{\text{cm}^2} \times \text{sec}} \right) \times \text{sec}$$
$$= \text{cm}$$

∴ so both end unit matched.

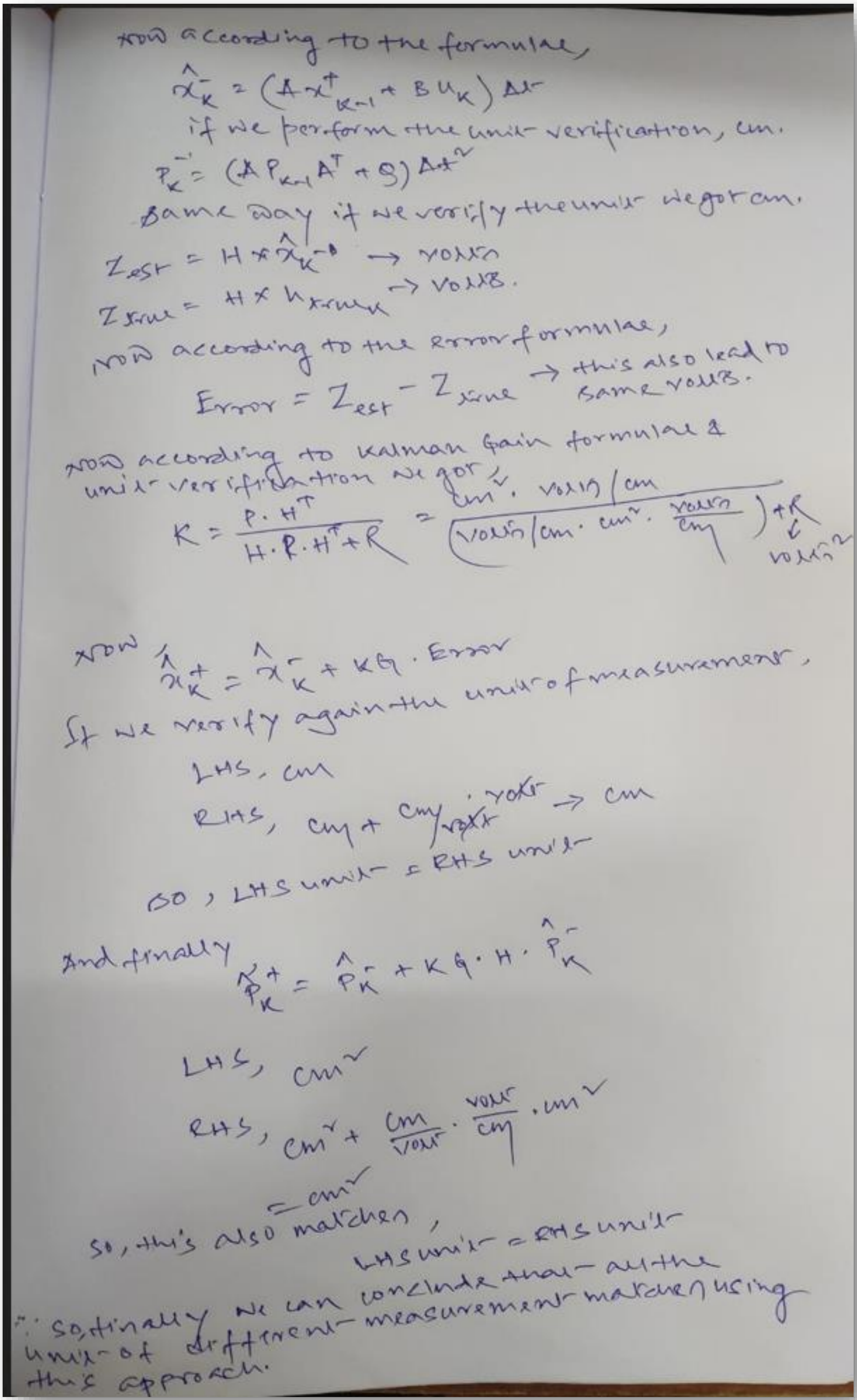
Again we know,
$$z = H.x$$
If we perform the unit analysis, we got,

LHS, volts

RHS, $\text{volt}/\cancel{\text{cm}} \times \cancel{\text{cm}} = \text{volts}$.

∴ Here also unit matched for both side.

Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503



Now according to the formulae,

$\hat{x}_K^- = (Ax_{K-1}^+ + Bu_K)\Delta t$

if we perform the unit verification, cm.

$P_K^- = (AP_{K-1}A^T + Q)\Delta t^2$

Same way if we verify the unit we got cm.

$Z_{est} = H * \hat{x}_K^-$ → volts

$Z_{true} = H \times h_{true}$ → volts.

Now according to the error formulae,

$\text{Error} = Z_{est} - Z_{true}$ → this also lead to same volts.

Now according to kalman Gain formulae & unit verification we got,

$K = \frac{P \cdot H^T}{H \cdot P \cdot H^T + R} = \frac{cm^2 \cdot volts/cm}{(volts/cm \cdot cm^2 \cdot \frac{volts}{cm}) + \underset{volts^2}{R}}$

Now $\hat{x}_K^+ = \hat{x}_K^- + KG \cdot \text{Error}$

If we verify again the unit of measurement,

LHS, cm

RHS, $cm + \frac{cm}{volt} \cdot volt$ → cm

So, LHS unit = RHS unit

And finally $\hat{P}_K^+ = \hat{P}_K^- + KG \cdot H \cdot \hat{P}_K^-$

LHS, cm^2

RHS, $cm^2 + \frac{cm}{volt} \cdot \frac{volt}{cm} \cdot cm^2$

$= cm^2$

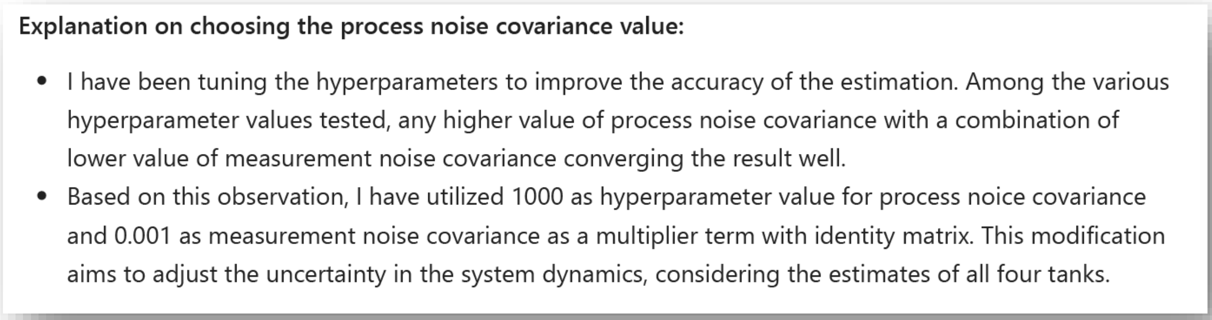
So, this's also matches,
LHS unit = RHS unit

∴ So, finally we can conclude that all the unit of different measurement matches using this approach.

Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503

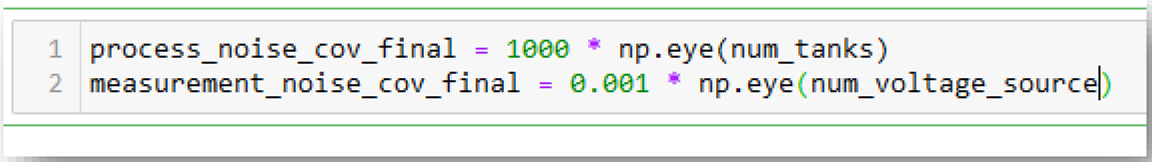
<u>Step1 – Simulate the Kalman Filter implementation:</u> The process noise covariance matrix and the measurement noise covariance matrix are hyperparameters that can be adjusted to obtain the convergence of the Tank height measurement, as was described in the methodology description. I am changing those parameters as a first step to see the convergence of the tank height measurement in this second approach also.



Explanation on choosing the process noise covariance value:

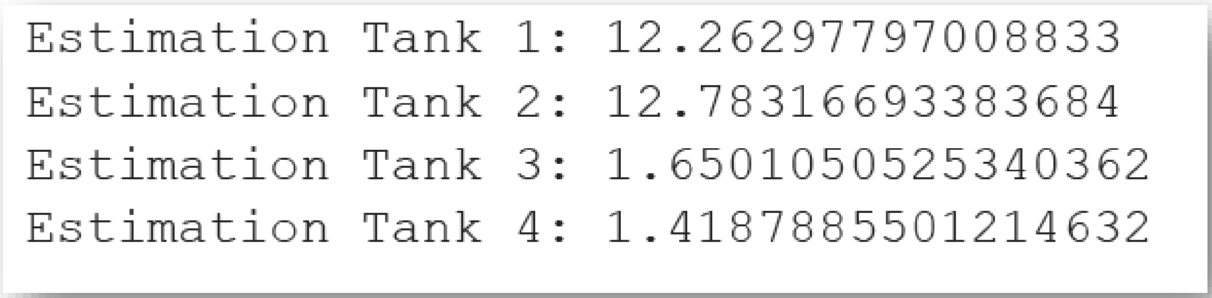
- I have been tuning the hyperparameters to improve the accuracy of the estimation. Among the various hyperparameter values tested, any higher value of process noise covariance with a combination of lower value of measurement noise covariance converging the result well.
- Based on this observation, I have utilized 1000 as hyperparameter value for process noice covariance and 0.001 as measurement noise covariance as a multiplier term with identity matrix. This modification aims to adjust the uncertainty in the system dynamics, considering the estimates of all four tanks.

So, we have finalized below value,



```
process_noise_cov_final = 1000 * np.eye(num_tanks)
measurement_noise_cov_final = 0.001 * np.eye(num_voltage_source)
```

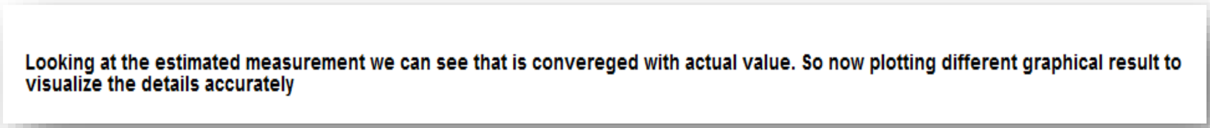
Calling Kalman filter function for estimation with new process noise covariance matrix and measurement noice covaraince



```
Estimation Tank 1: 12.262977970088 33
Estimation Tank 2: 12.7831669338368 4
Estimation Tank 3: 1.650105052534036 2
Estimation Tank 4: 1.418788550121463 2
```

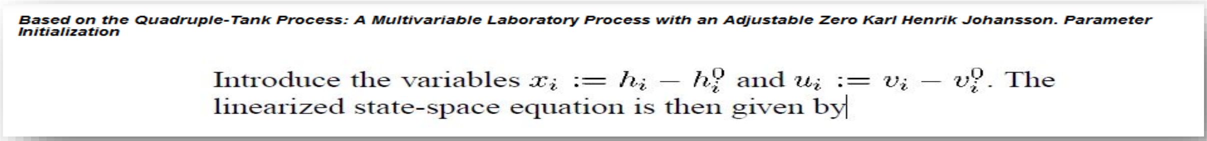
Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503



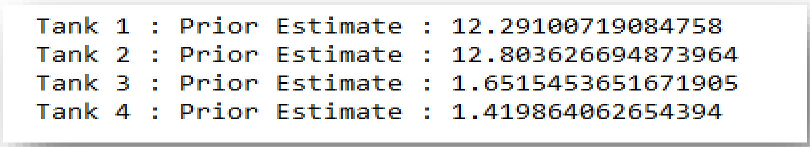
Looking at the estimated measurement we can see that is convereged with actual value. So now plotting different graphical result to visualize the details accurately

<u>Prior Plots:</u>

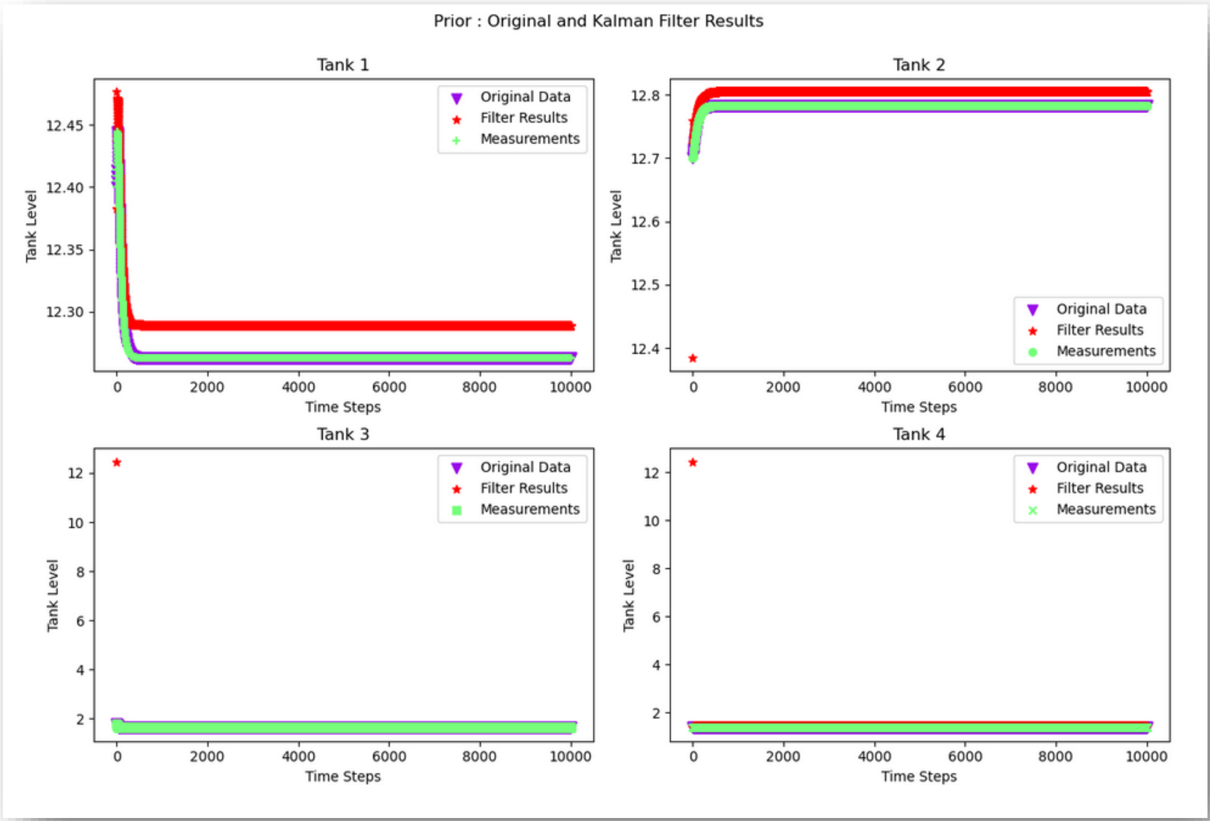


Based on the Quadruple-Tank Process: A Multivariable Laboratory Process with an Adjustable Zero Karl Henrik Johansson. Parameter Initialization

Introduce the variables $x_i := h_i - h_i^0$ and $u_i := v_i - v_i^0$. The linearized state-space equation is then given by

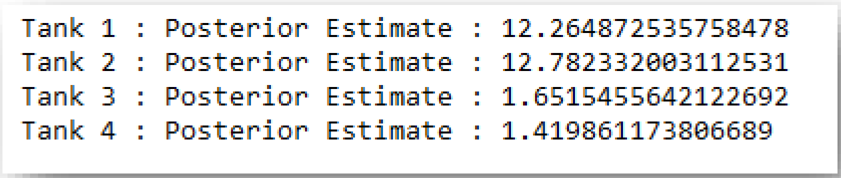


```
Tank 1 : Prior Estimate : 12.29100719084758
Tank 2 : Prior Estimate : 12.803626694873964
Tank 3 : Prior Estimate : 1.6515453651671905
Tank 4 : Prior Estimate : 1.419864062654394
```

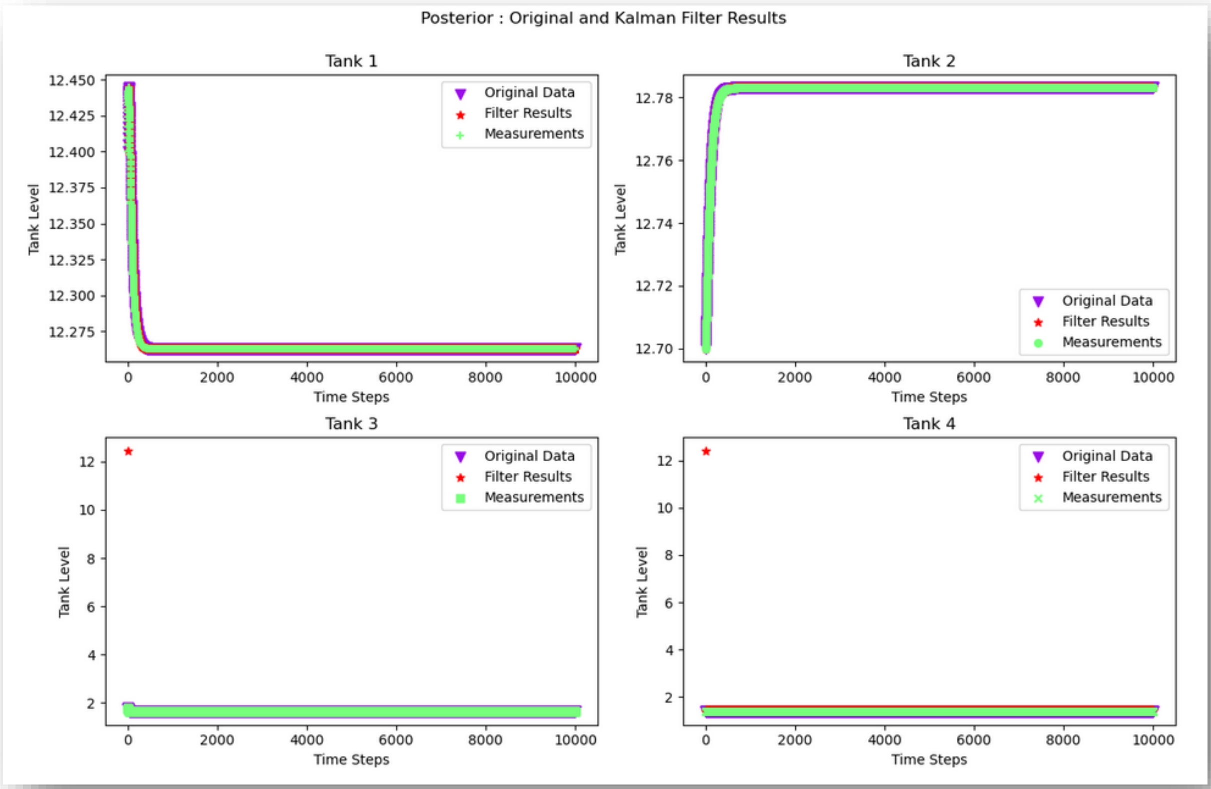


Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503

Posterior Plots:



```
Tank 1 : Posterior Estimate : 12.264872535758478
Tank 2 : Posterior Estimate : 12.782332003112531
Tank 3 : Posterior Estimate : 1.6515455642122692
Tank 4 : Posterior Estimate : 1.419861173806689
```

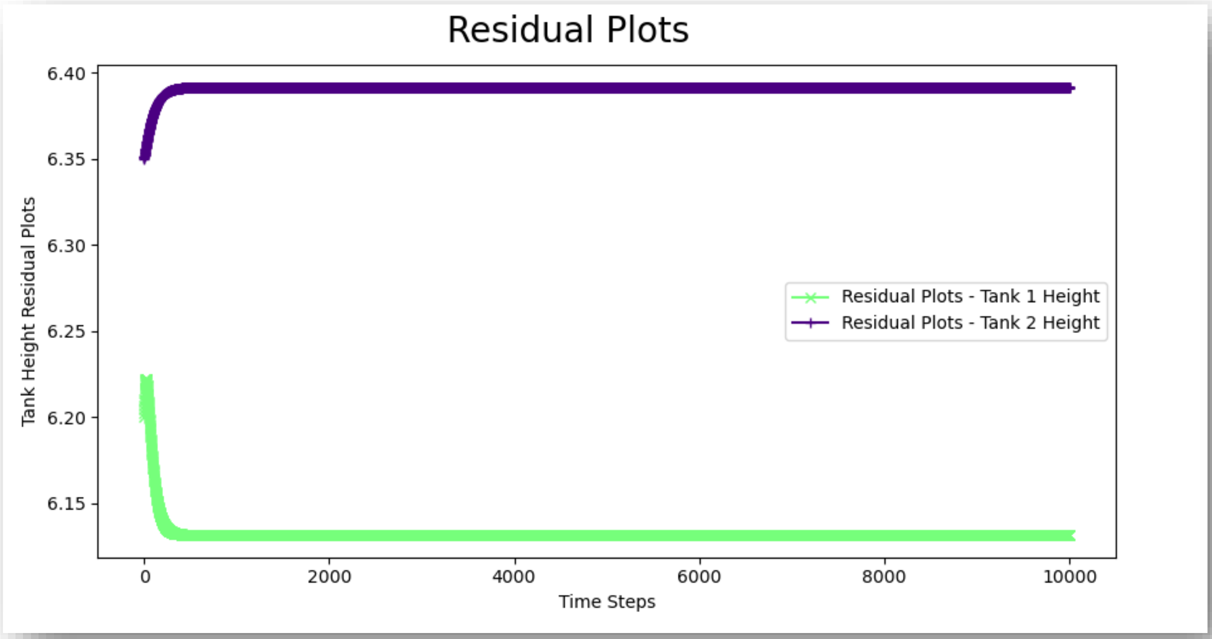


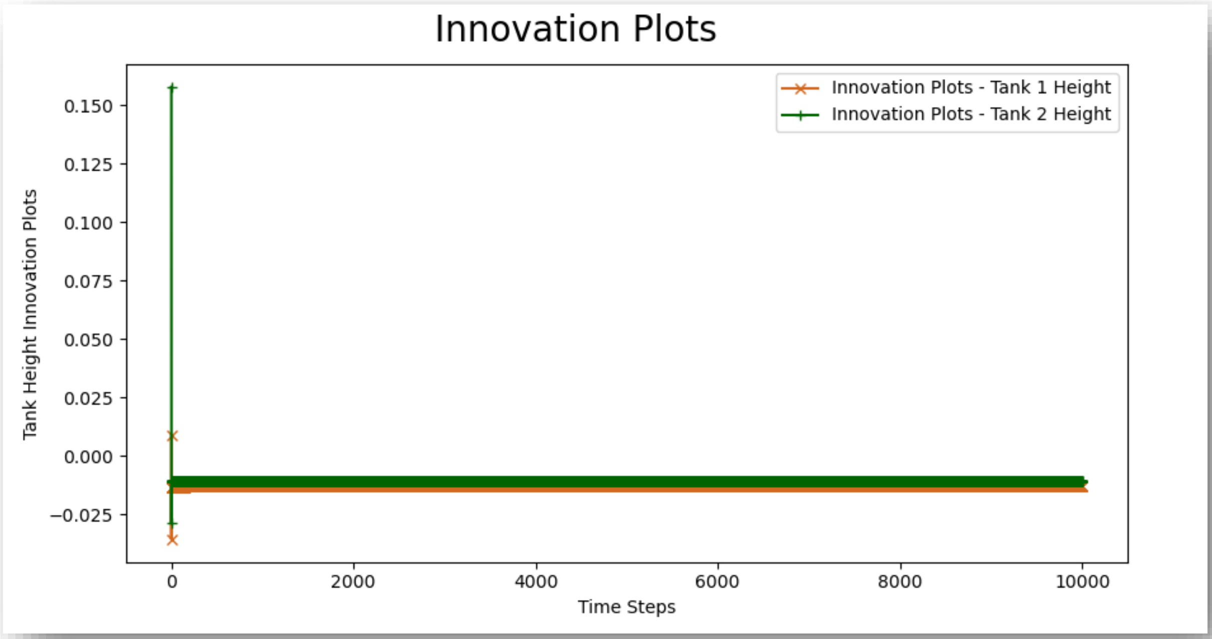
Upon careful examination of the plot, it becomes apparent that the estimated values for all four tanks exhibit a notable convergence with their corresponding measured values. This observation reinforces the efficacy of the second approach employed in the Kalman filter implementation. These findings further validate the effectiveness of our methodology in accurately estimating the tank values, ultimately contributing to the advancement of state estimation techniques in the context of the four-tank problem.

Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503

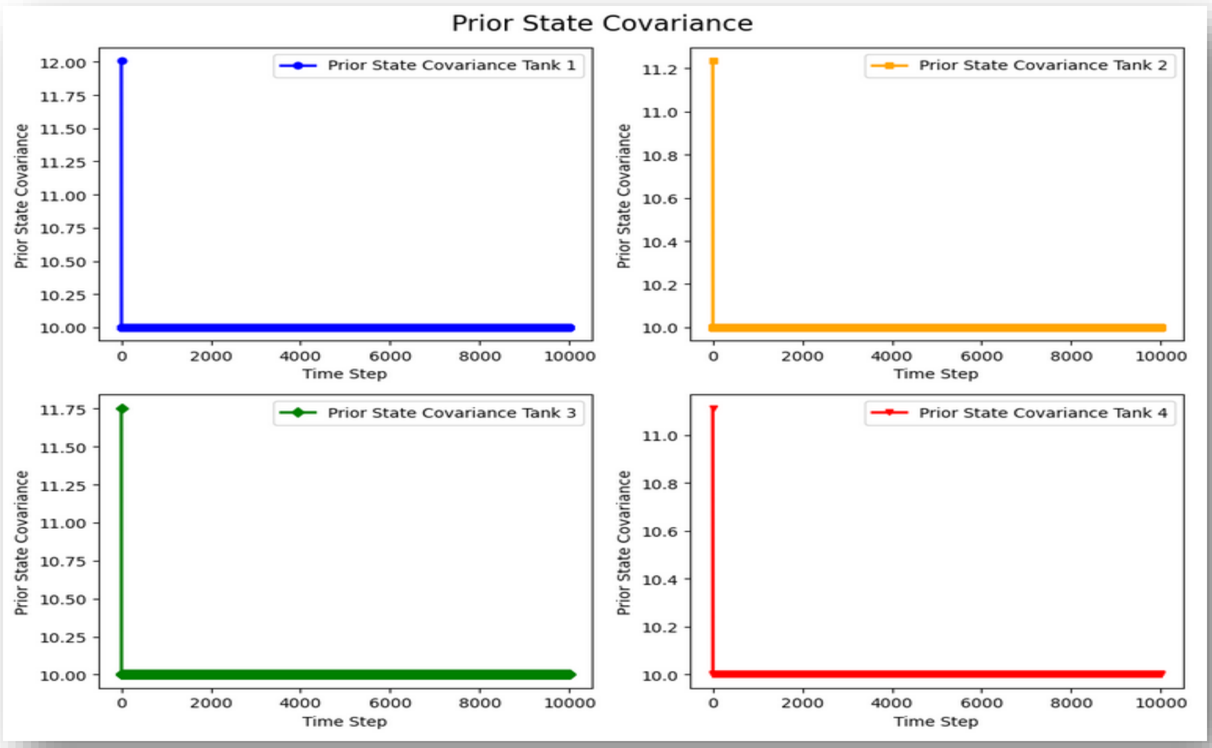
Different graphical plots for Kalman Filter using 2nd Approach

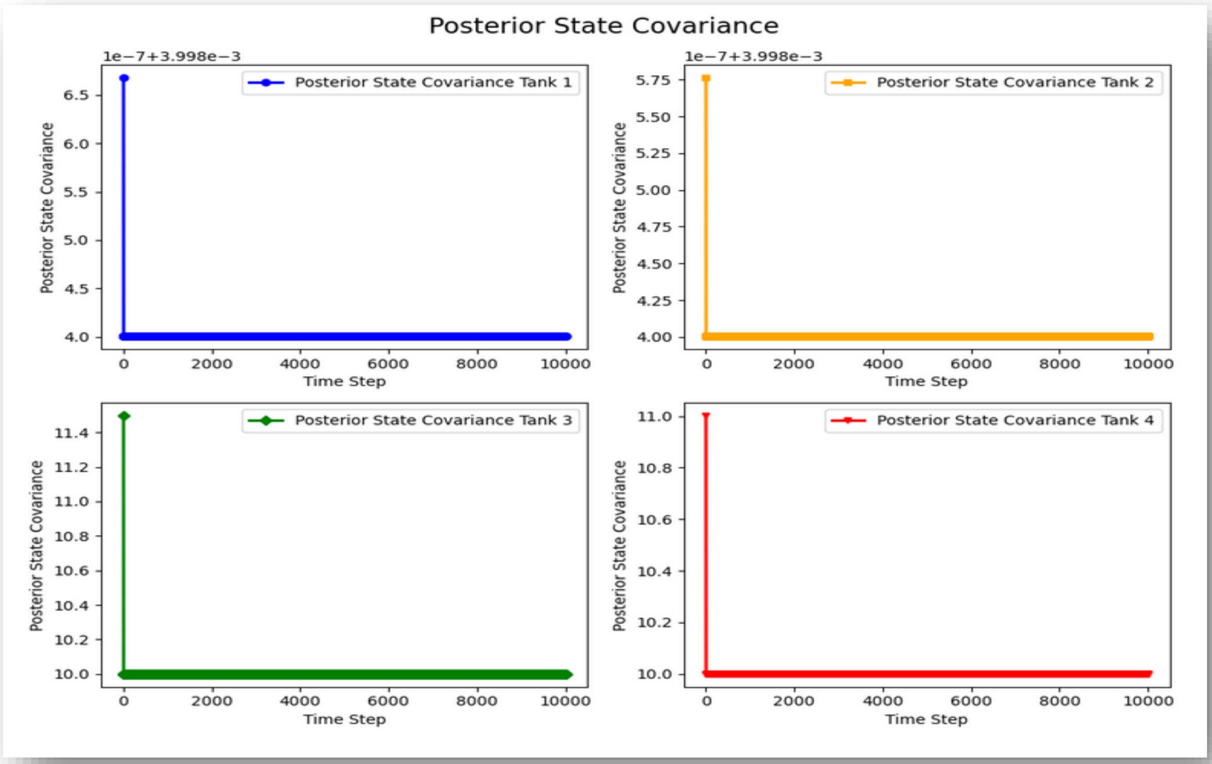




Mini Project – Kalman : Report

Author: Aloy Banerjee
Roll No.: CH22M503

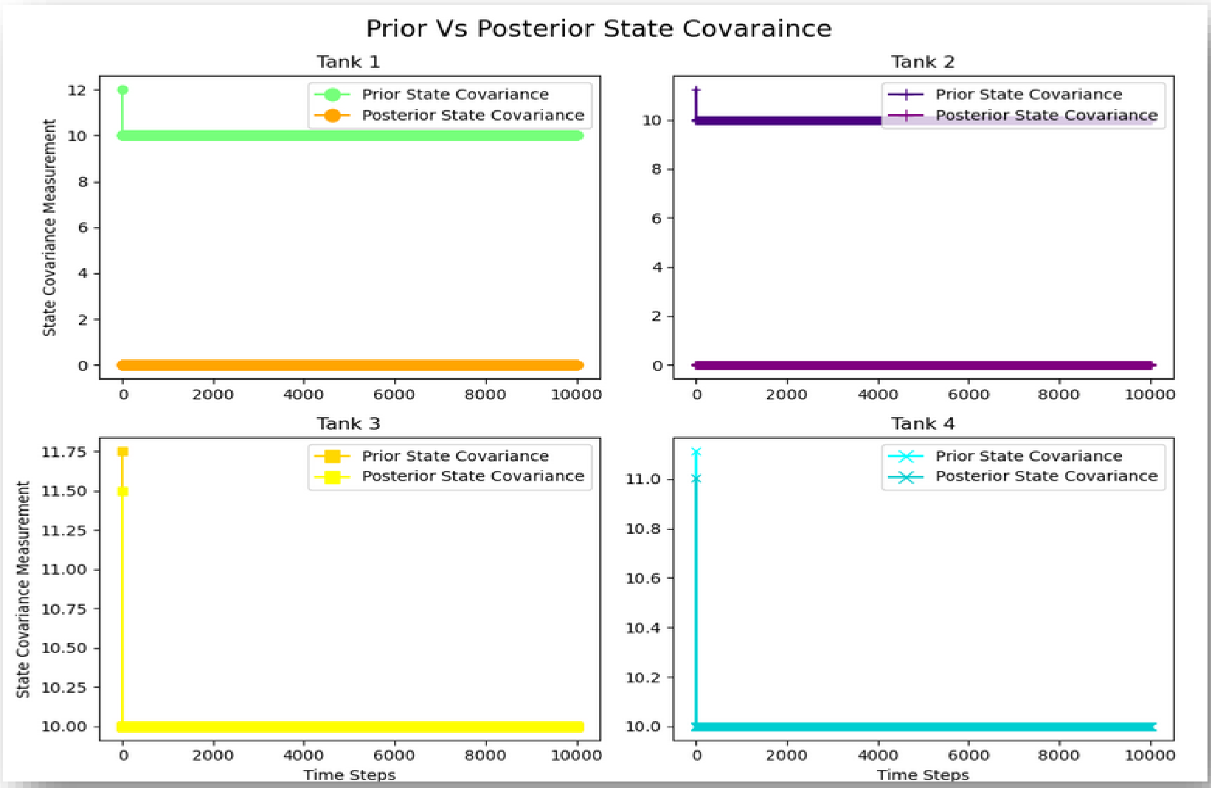


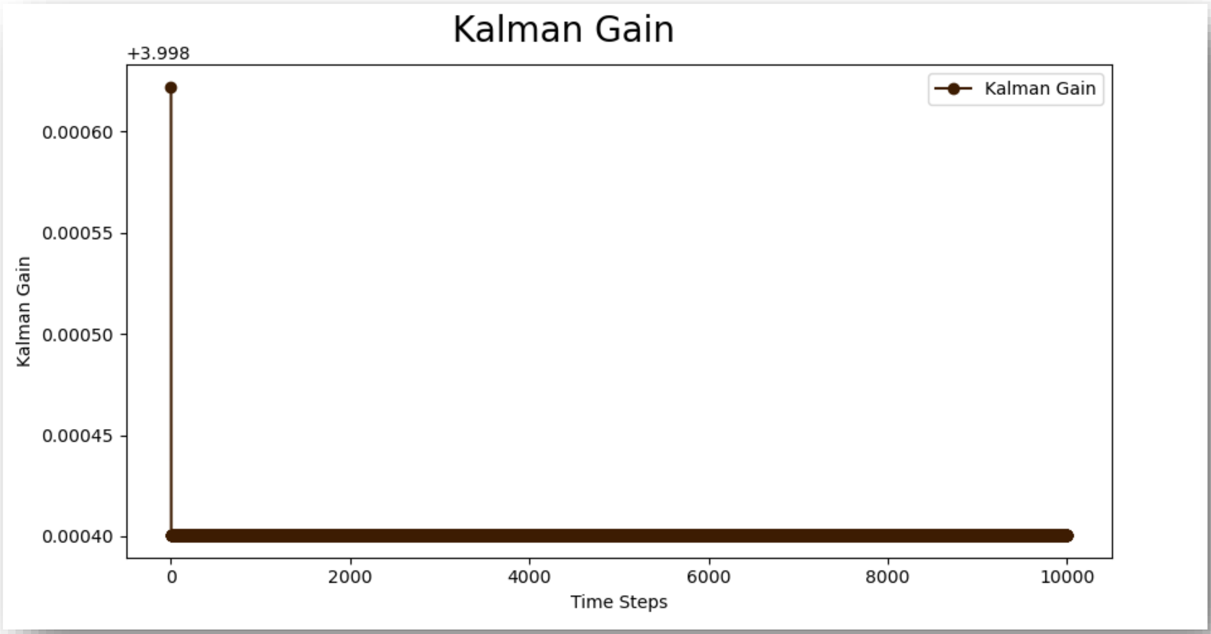


Mini Project – Kalman : Report

Author: Aloy Banerjee

Roll No.: CH22M503

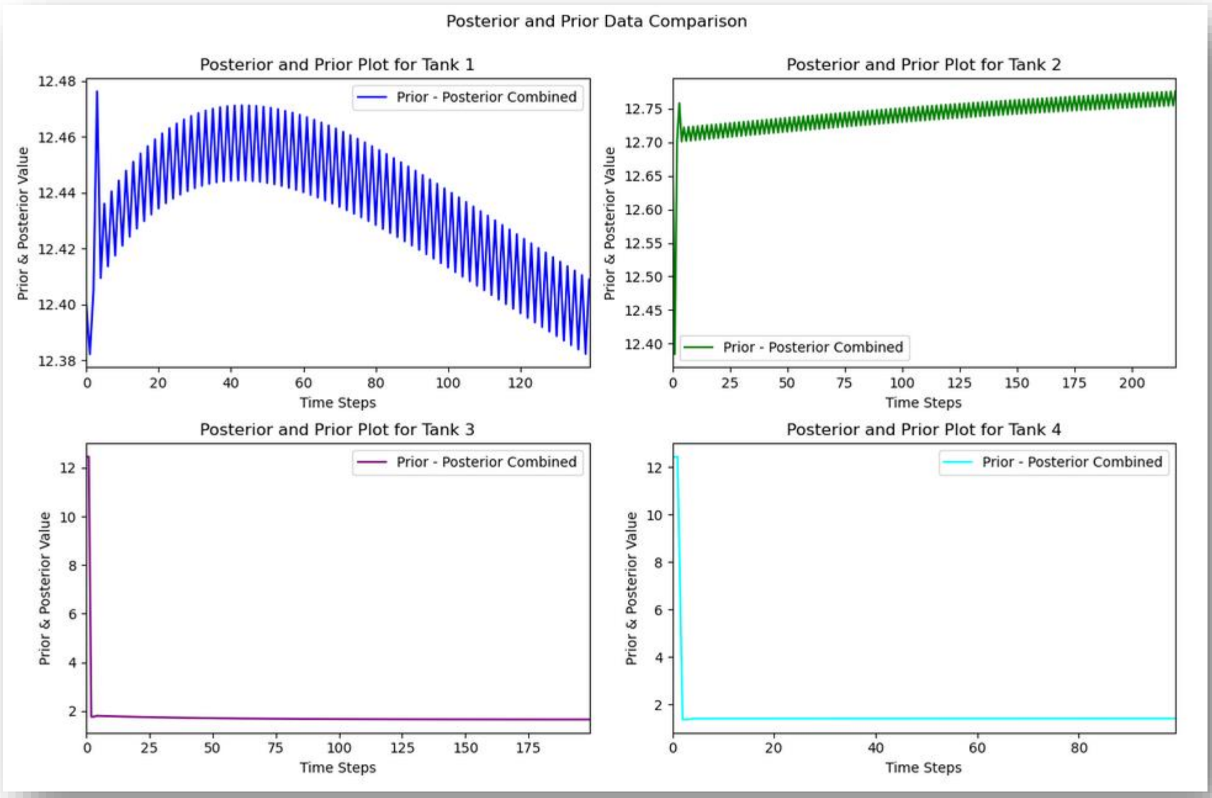


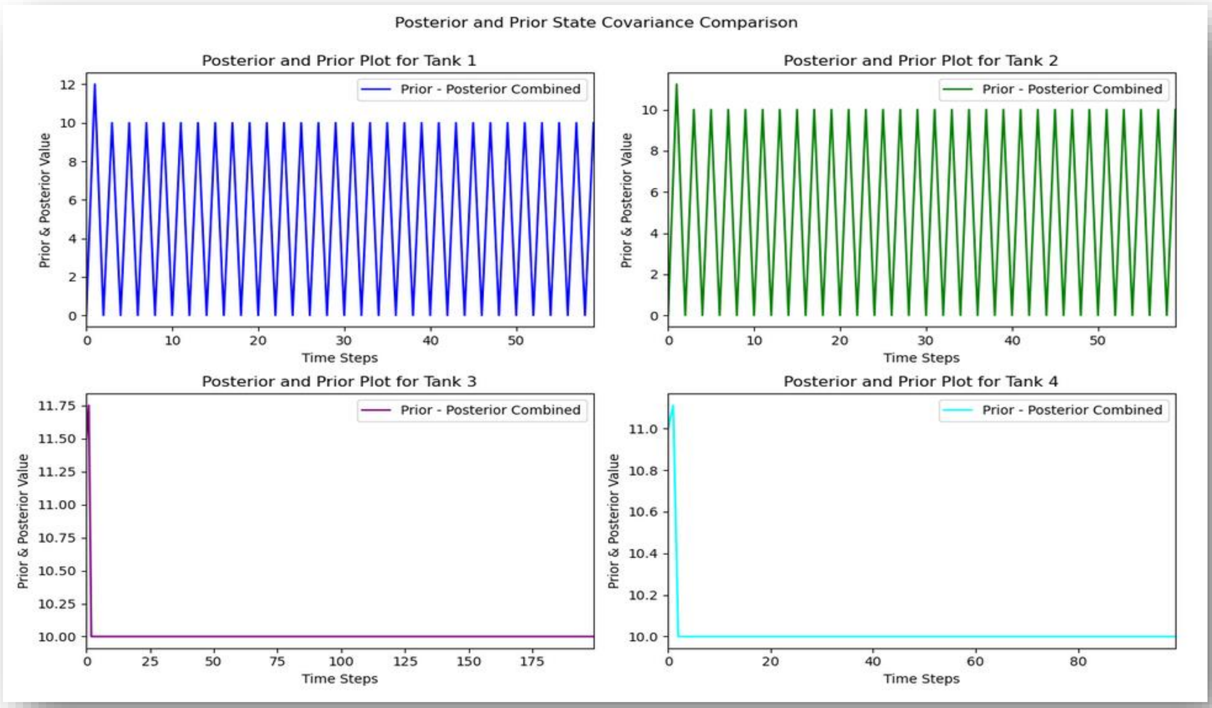


Mini Project – Kalman : Report

Author: Aloy Banerjee

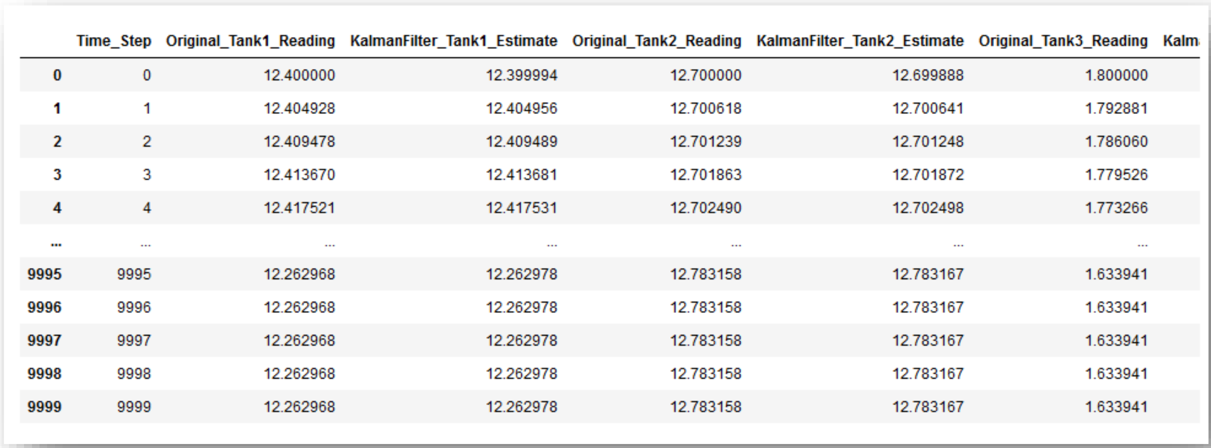
Roll No.: CH22M503





Mini Project – Kalman : Report
Author: Aloy Banerjee
Roll No.: CH22M503

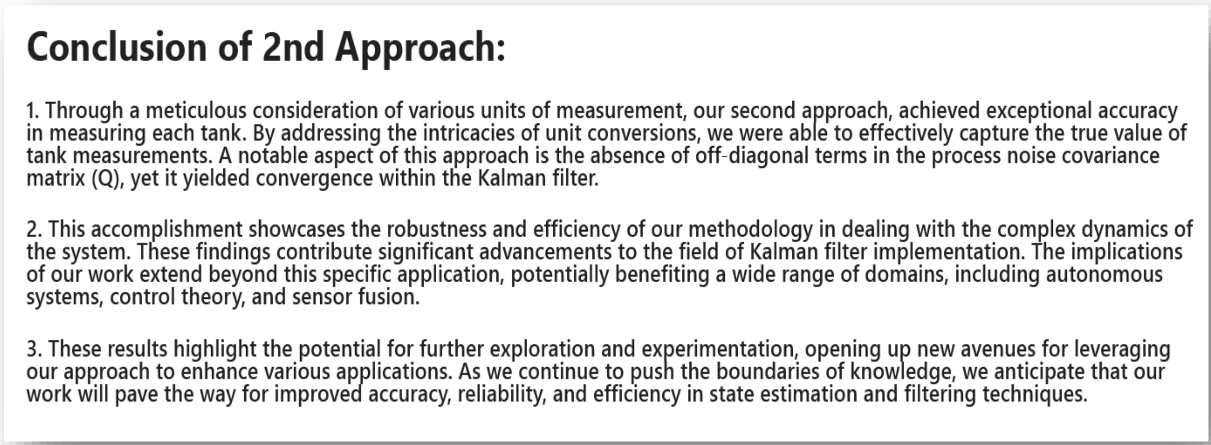
Kalman Filter Result - 2nd Approach



	Time_Step	Original_Tank1_Reading	KalmanFilter_Tank1_Estimate	Original_Tank2_Reading	KalmanFilter_Tank2_Estimate	Original_Tank3_Reading	Kalm
0	0	12.400000	12.399994	12.700000	12.699888	1.800000	
1	1	12.404928	12.404956	12.700618	12.700641	1.792881	
2	2	12.409478	12.409489	12.701239	12.701248	1.786060	
3	3	12.413670	12.413681	12.701863	12.701872	1.779526	
4	4	12.417521	12.417531	12.702490	12.702498	1.773266	
...	...	...	...	...	...	...	
9995	9995	12.262968	12.262978	12.783158	12.783167	1.633941	
9996	9996	12.262968	12.262978	12.783158	12.783167	1.633941	
9997	9997	12.262968	12.262978	12.783158	12.783167	1.633941	
9998	9998	12.262968	12.262978	12.783158	12.783167	1.633941	
9999	9999	12.262968	12.262978	12.783158	12.783167	1.633941	

Comparison_Kalman_Approach2.csv

Figure 3Result of Kalman Gain in Approach 2



Conclusion of 2nd Approach:

1. Through a meticulous consideration of various units of measurement, our second approach, achieved exceptional accuracy in measuring each tank. By addressing the intricacies of unit conversions, we were able to effectively capture the true value of tank measurements. A notable aspect of this approach is the absence of off-diagonal terms in the process noise covariance matrix (Q), yet it yielded convergence within the Kalman filter.

2. This accomplishment showcases the robustness and efficiency of our methodology in dealing with the complex dynamics of the system. These findings contribute significant advancements to the field of Kalman filter implementation. The implications of our work extend beyond this specific application, potentially benefiting a wide range of domains, including autonomous systems, control theory, and sensor fusion.

3. These results highlight the potential for further exploration and experimentation, opening up new avenues for leveraging our approach to enhance various applications. As we continue to push the boundaries of knowledge, we anticipate that our work will pave the way for improved accuracy, reliability, and efficiency in state estimation and filtering techniques.

<u>End of Report</u>