Name: Dileep Naidu Maripi

Case 1

Part 1

In a static system where there is no change in the water level over time (because the tap is not adding water and we can assume no water is leaving the tank), the state of the system does not change from one time step to the next. This implies that the water level at the next time step (x(n+1)) is the same as the water level at the current time step (x(n)), given there is no input or disturbance affecting the system.

In such a scenario:

1. The matrix (A) would be the identity matrix (I), which when multiplied by the state (x(n)) would yield the same state (x(n)) for the next time step. In other words, (A) would look like this for a one-dimensional state such as the water level:

$ A = \begin{bmatrix} 1 \end{bmatrix}$

1. The input (u(n)) would be zero at every time step (n), reflecting that there is no external action being applied to the system (i.e., the tap is not adding water). Mathematically, this would be represented as:

[u(n) = 0]

These assumptions lead to the conclusion that the system's state is constant over time when there is no input, and the state transition matrix simply reflects the current state to the next state without any change.

Part 2

If you have only one sensor that directly measures the water level, and assuming that the state (x(n)) is a scalar representing the water level at time (n), then the measurement matrix (C) would be a row vector with a single element equal to 1. This is because the measurement directly reflects the state without any transformation. Mathematically, C will be:

$C=[1)$

The measurement becomes:

$y(n)=x(n)+\epsilon_m$

###Part 3

Using the Kalman filter with your initial assumptions and the first measurement, the best estimation for the water level after the first measurement is approximately $x(1/\ 1)=0.8999$

with an updated error covariance of $P(1/1) \approx 0.1$ and a Kalman gain of $K(1) \approx 0.9999$. This result suggests that the filter places high trust in the first measurement due to the initially high uncertainty (represented by the large initial error covariance).

```
import numpy as np

# Assumptions
x_0 = 0.0
P_0 = 1000.0
q = 0.0001
r = 0.1
y_1 = 0.9

# Since the system is static and we have only one sensor measuring the water
level,
A = np.array([1])
C = np.array([1])
B = np.array([0])
u_n = np.array([0])

# Kalman Filter equations for the first estimate
x_pred = A * x_0 + B * u_n
P_pred = A * P_0 * A + q

# Update step
K_1 = P_pred * C / (C * P_pred * C + r)
x_1 = x_pred + K_1 * (y_1 - C * x_pred)
P_1 = (1 - K_1 * C) * P_pred

print("X_1:",x_1)
print("P_1:",P_1)
print("K_1:",K_1)
```

```
X_1: [0.89991001]
P_1: [0.09999]
K_1: [0.99990001]
```

###Part 4

To find the estimation of the water level and the related covariance matrix using Maximum Likelihood (ML) theory, we'll consider the measurement model and the initial assumptions. In this case, the ML estimation is straightforward because we're dealing with a single measurement and a scalar system.

The ML estimation for the water level, given a single noisy measurement, is simply the measurement itself when there's no prior information (or when the prior is non-informative). This is because, under ML, we choose the parameter value (water level, in this case) that maximizes the likelihood of the observed data (the measurement).

1. **ML Estimation of Water Level**: Given the measurement (y(1) = 0.9), the ML estimate for the water level would be 0.9.

1. **Covariance Matrix under ML**: The covariance in the ML estimation context is typically the variance of the measurement noise, which in this case is (r = 0.1).

Comparing this to the Kalman Filter results:

- The Kalman Filter estimate was $ x(1|1) \approx 0.8999$, which is very close to the ML estimate of 0.9.
- The Kalman Filter error covariance was $P(1 \vee 1) \approx 0.1$, matching the assumed measurement noise covariance in the ML approach.

Now, if we increase $P_0 \vee 0$ to 10,000, it means we're starting with even less confidence in our initial state estimate. This should make the Kalman Filter rely more heavily on the first measurement, potentially bringing the Kalman Filter results even closer to the ML estimate. Let's calculate this:

With the initial error covariance $P_0 \vee 0$ increased to 10,000, the updated Kalman Filter results are:

- Updated state estimate: $x(1 \vee 1) \approx 0.9$
- Updated error covariance: $P(1|1) \approx 0.1 $
- Kalman gain: $K(1) \approx 1.0$

These results are even closer to the Maximum Likelihood estimates:

- ML Estimate of Water Level: 0.9
- Covariance under ML: 0.1

This demonstrates that as the initial uncertainty in the Kalman Filter (represented by (P_0|0)) increases, the filter's estimate relies more heavily on the measurement, aligning more closely with the ML estimation, which considers only the measurement and its noise.

```python
# New initial error covariance
P_0_new = 10000.0

# Kalman Filter equations with the new initial error covariance
# Prediction step remains the same
P_pred_new = A * P_0_new * A + q # New prediction of error covariance

# Update step
K_1_new = P_pred_new * C / (C * P_pred_new * C + r) # New Kalman gain
x_1_new = x_pred + K_1_new * (y_1 - C * x_pred) # New updated state estimate
P_1_new = (1 - K_1_new * C) * P_pred_new # New updated error covariance

print("x_1_new:",x_1_new)
print("P_1_new:",P_1_new)
print("K_1_new:",K_1_new)
```

```
x_1_new: [0.899991]
P_1_new: [0.099999]
K_1_new: [0.99999]
```

Part 5

The Kalman filter estimations of the water level, along with the corresponding variances and Kalman gains after each measurement, are as follows:

1. After 1st measurement: Estimated Level = 0.89999, Variance = 0.1
1. After 2nd measurement: Estimated Level = 0.84997, Variance = 0.05002
2. After 3rd measurement: Estimated Level = 0.93345, Variance = 0.03339
3. After 4th measurement: Estimated Level = 0.95015, Variance = 0.02509
4. After 5th measurement: Estimated Level = 0.95012, Variance = 0.02012
5. After 6th measurement: Estimated Level = 0.96692, Variance = 0.01682
6. After 7th measurement: Estimated Level = 1.00065, Variance = 0.01447
7. After 8th measurement: Estimated Level = 0.98785, Variance = 0.01272
8. After 9th measurement: Estimated Level = 0.97218, Variance = 0.01136
9. After 10th measurement: Estimated Level = 0.99047, Variance = 0.01028

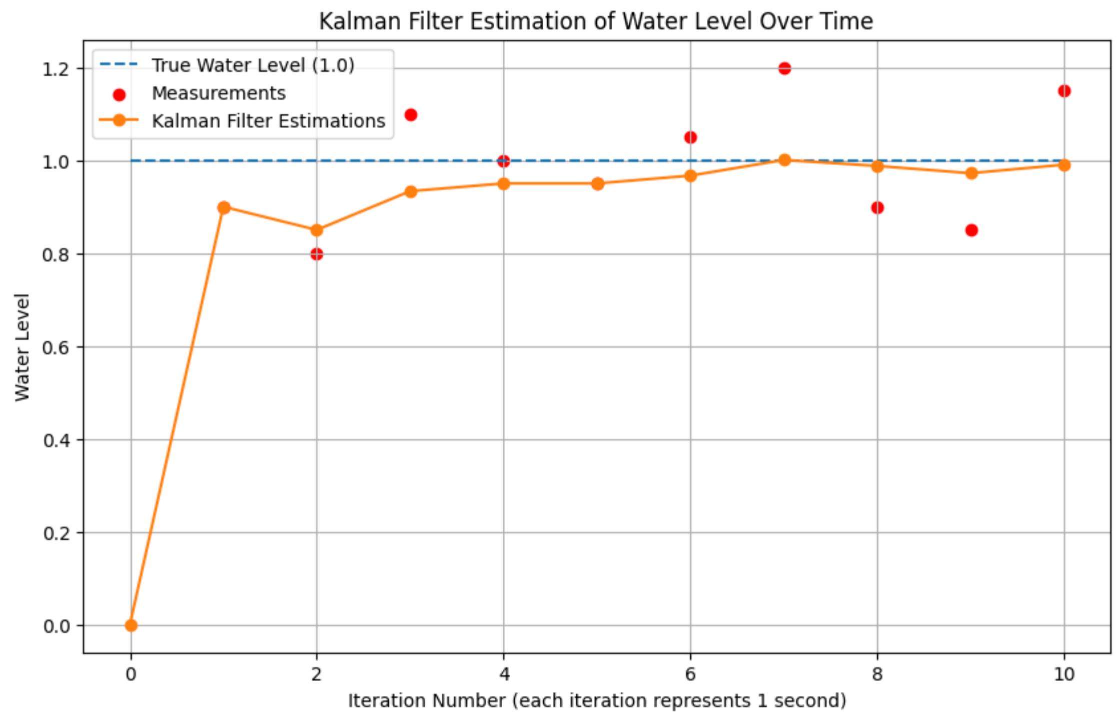
The plot shows the true value of the water level (assumed constant at 1.0), all measurements, and the Kalman filter estimations as a function of the iteration number.

It took 5 seconds (iterations) for the estimations to be within the 5% range of the accurate level. This is evident from the plot and the time within which the estimated levels become consistently close to the true level of 1.0.

The initial estimations fluctuate, reflecting the influence of the noisy measurements, but as more measurements are taken, the Kalman filter's estimates become more stable and closer to the true level, demonstrating the efficacy of the filter in converging towards the actual value despite measurement noise.

```python
import matplotlib.pyplot as plt

# Measurement outcomes
measurements = [0.9, 0.8, 1.1, 1.0, 0.95, 1.05, 1.2, 0.9, 0.85, 1.15]

# Reinitializing the initial state and covariance
x_0 = 0.0  # initial state estimate
P_0 = 10000.0  # initial covariance estimate with very low confidence
q = 0.0001  # process noise covariance
r = 0.1  # measurement noise covariance

# Lists to store the Kalman filter results
estimated_levels = [x_0]
covariances = [P_0]
kalman_gains = []
```

```
# True water level (assuming it remains constant at 1.0)
true_level = 1.0

# Kalman Filter iterations
x_est = x_0
P_est = P_0
for y in measurements:
    # Prediction step
    x_pred = A * x_est
    P_pred = A * P_est * A + q

    # Update step
    K = P_pred * C / (C * P_pred * C + r)
    x_est = x_pred + K * (y - C * x_pred)
    P_est = (1 - K * C) * P_pred

    # Store results
    estimated_levels.append(x_est[0])
    covariances.append(P_est[0])
    kalman_gains.append(K[0])

# Time when estimation is within 5% of the true level
within_5 = [i for i, x in enumerate(estimated_levels, 1) if abs(x - 
true_level) <= 0.05 * true_level]
time_within_5 = within_5[0] if within_5 else None

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(range(11), [true_level] * 11, label="True Water Level (1.0)", 
linestyle='--')
plt.scatter(range(1, 11), measurements, color='red', label="Measurements")
plt.plot(range(11), estimated_levels, marker='o', label="Kalman Filter 
Estimations")
plt.title("Kalman Filter Estimation of Water Level Over Time")
plt.xlabel("Iteration Number (each iteration represents 1 second)")
plt.ylabel("Water Level")
plt.legend()
plt.grid(True)
plt.show()

print("estimated_levels:",estimated_levels)
print("covariances:",covariances)
print("kalman_gains:",kalman_gains)
print("time_within_5_percent:",time_within_5)
```



```
estimated_levels: [0.0, 0.8999910000900891, 0.8499707645140507,
0.9334523415521003, 0.9501473567798023, 0.9501177090422314,
0.9669168668117368, 1.00064555589948, 0.9878458305526527, 0.9721846580949096,
0.9904690838982521]
covariances: [10000.0, 0.09999900001024342, 0.050024737757369694,
0.033388726272668576, 0.02508730677695087, 0.02011969697680906,
0.01681895520058542, 0.014470669166996375, 0.012717625962154764,
0.011361368272768388, 0.010282816773539075]
kalman_gains: [0.999990000100099, 0.5002473775736969, 0.33388726272668573,
0.2508730677695087, 0.20119696976809062, 0.1681895520058542,
0.14470669166996375, 0.12717625962154763, 0.11361368272768388,
0.10282816773539073]
time_within_5_percent: 5
```

Case 2

Part 1

The plot shows the true water level (which increases by 0.1 unit per second), the measurements, and the Kalman filter estimations over time.

Observations from the Plot:

- The Kalman filter estimations lag behind the actual water levels.
- The estimated water level after 15 seconds is approximately 0.832, while the actual level is 1.5.
- The error covariance (variance) of the estimation after 15 seconds is approximately 0.0071.

Problem Analysis:

- The key issue here is that the model used for the Kalman filter assumes a static system (no change in water level over time), but in reality, the water level is constantly increasing due to the tap being open.
- This mismatch between the model and the actual system dynamics leads to the Kalman filter underestimating the water level.

Biased Error:

- The consistent underestimation suggests a biased error in the system. This bias arises because the model does not account for the continuous increase in water level.

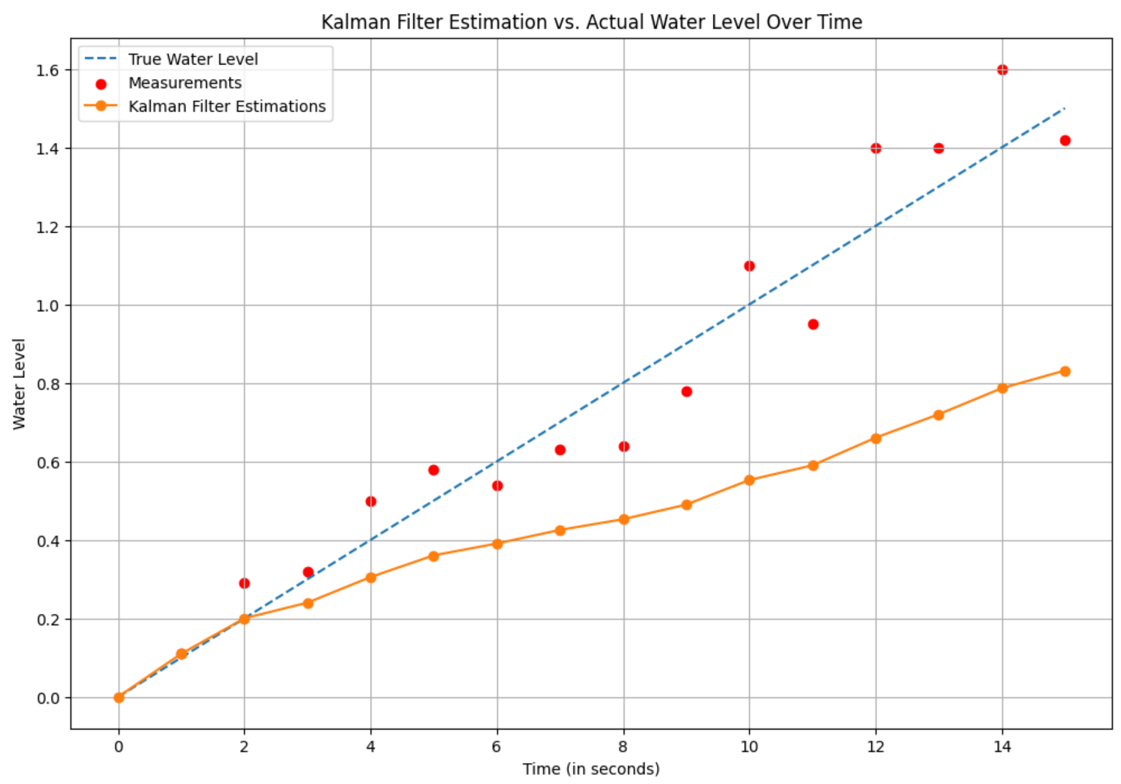
Improvement Suggestions:

- To improve the accuracy without changing the dynamics (i.e., still using a static model), you could adjust the process noise covariance Q. Increasing Q would tell the Kalman filter that there might be more uncertainty in the model, causing it to rely more on the measurements. This won't completely solve the issue due to the fundamental model mismatch, but it could reduce the estimation error.
- Another approach could be to modify the Kalman filter's update mechanism to be more responsive to measurement deviations, but this also only partially addresses the issue.

In summary, while some adjustments can mitigate the problem, the most effective solution would involve updating the model to reflect the actual system dynamics (i.e., the increasing water level due to the tap being open). However, since changing the dynamics is not an option in this scenario, increasing the process noise covariance is a viable alternative to make the estimations more responsive to the measurements.

```
# New measurements
measurements_new = [0.11, 0.29, 0.32, 0.50, 0.58, 0.54, 0.63, 0.64, 0.78,
1.1, 0.95, 1.4, 1.4, 1.6, 1.42]

# True water levels over time (increasing by 0.1 unit per second)
true_levels = [0.1 * i for i in range(16)]

# Reinitializing the initial state and covariance for Kalman Filter
x_0_new = 0.0
P_0_new = 10000.0

# Lists to store the Kalman filter results
estimated_levels_new = [x_0_new]
covariances_new = [P_0_new]

# Kalman Filter iterations
x_est_new = x_0_new
P_est_new = P_0_new
```

```
for y in measurements_new:
    # Prediction step (still using the static model assumption)
    x_pred_new = A * x_est_new
    P_pred_new = A * P_est_new * A + q

    # Update step
    K_new = P_pred_new * C / (C * P_pred_new * C + r)
    x_est_new = x_pred_new + K_new * (y - C * x_pred_new)
    P_est_new = (1 - K_new * C) * P_pred_new

    # Store results
    estimated_levels_new.append(x_est_new[0])
    covariances_new.append(P_est_new[0])

# Plotting
plt.figure(figsize=(12, 8))
plt.plot(range(16), true_levels, label="True Water Level", linestyle='--')
plt.scatter(range(1, 16), measurements_new, color='red',
label="Measurements")
plt.plot(range(16), estimated_levels_new, marker='o', label="Kalman Filter
Estimations")
plt.title("Kalman Filter Estimation vs. Actual Water Level Over Time")
plt.xlabel("Time (in seconds)")
plt.ylabel("Water Level")
plt.legend()
plt.grid(True)
plt.show()

# Return the final estimated level and covariance
estimated_levels_new[-1], covariances_new[-1]
print('New Estimated Levels:]',estimated_levels_new[-1])
print('New covariances:', covariances_new[-1])
```



```
New Estimated Levels:] 0.8315307111729513
New covariances: 0.0071104724420466585
```

Part 2

The plot shows the actual water level, measurements, and Kalman filter estimations with varying values of (q) (process noise covariance) over time. The estimations for each (q) value (0.01, 0.1, and 1.0) are plotted to illustrate how the Kalman filter's performance changes with different assumptions about system dynamics uncertainty.

Observations:

- As (q) increases, the Kalman filter's estimations become more responsive to the measurements. This is because a higher (q) value represents more uncertainty in the model, causing the filter to rely more on the incoming measurements.
- For (q = 0.01), the estimations improve compared to the very low (q) value of 0.0001, but there is still some lag behind the actual values.
- For (q = 0.1), the estimations are closer to the actual levels, but they still show some lag, especially towards the end.
- For (q = 1.0), the estimations are more aligned with the measurements, but they tend to overreact to individual measurements, leading to higher variability.

Learnings:

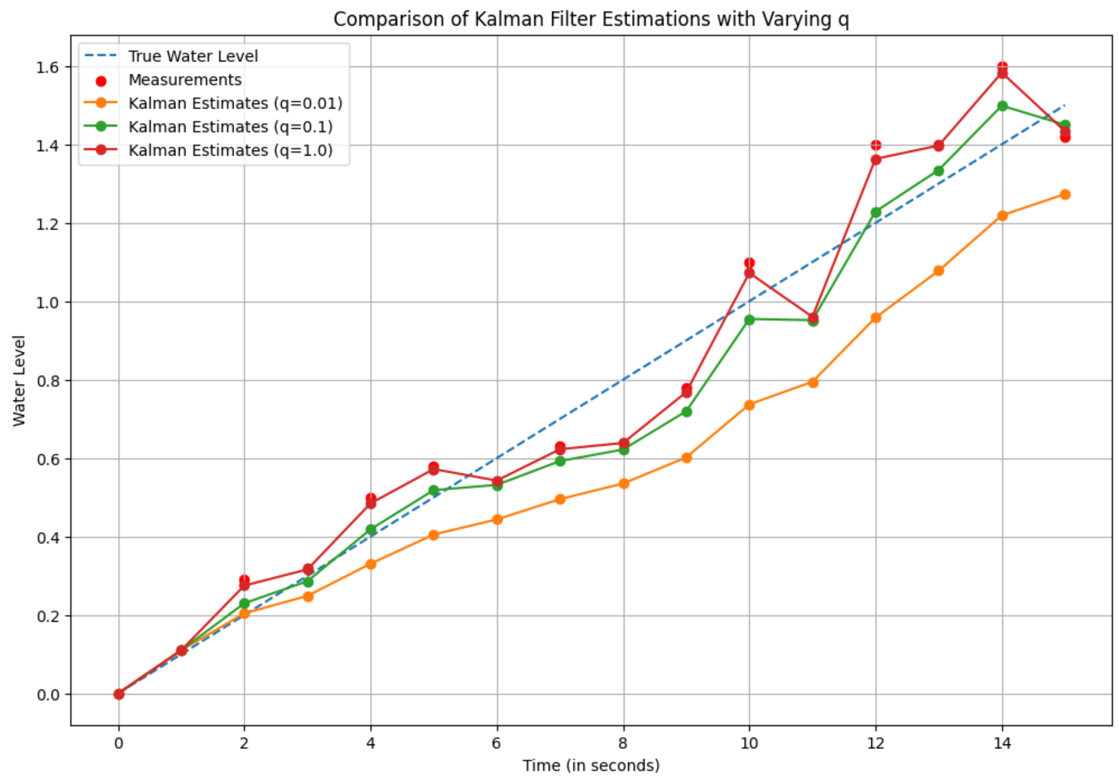
- The choice of (q) significantly impacts the Kalman filter's performance, especially when there is a mismatch between the model and the actual system dynamics.

- A higher (q) makes the filter more adaptive to changes in the system that are not captured by the model. However, it can also lead to overfitting to noisy measurements.
- Balancing the value of (q) is crucial for achieving accurate and stable estimations. It needs to be high enough to account for model inaccuracies but not so high that the filter becomes overly sensitive to measurement noise.

Conclusion:

- Adjusting (q) to a higher value can partially compensate for an incorrect model assumption (like assuming a static system when the water level is actually rising). However, the best results are achieved when the model accurately reflects the system dynamics. If the model cannot be changed, adjusting (q) is a practical method to improve estimation accuracy.

```
# Varying values of q to represent different levels of uncertainty in the
system dynamics
q_values = [0.01, 0.1, 1.0]

# Lists to store Kalman filter results for different q values
estimated_levels_varied_q = {}

# Performing Kalman Filter iterations for each q value
for q_var in q_values:
    x_est_var = x_0_new
    P_est_var = P_0_new
    estimated_levels_q = [x_est_var]

    for y in measurements_new:
        # Prediction step with varied q
        P_pred_var = A * P_est_var * A + q_var

        # Update step
        K_var = P_pred_var * C / (C * P_pred_var * C + r)
        x_est_var = A * x_est_var + K_var * (y - C * x_est_var)
        P_est_var = (1 - K_var * C) * P_pred_var

        # Store results
        estimated_levels_q.append(x_est_var[0])

    # Store results for each q value
    estimated_levels_varied_q[q_var] = estimated_levels_q

# Plotting
plt.figure(figsize=(12, 8))
plt.plot(range(16), true_levels, label="True Water Level", linestyle='--')
plt.scatter(range(1, 16), measurements_new, color='red',
label="Measurements")
for q_val, est_levels in estimated_levels_varied_q.items():
    plt.plot(range(16), est_levels, marker='o', label=f"Kalman Estimates
(q={q_val})")

plt.title("Comparison of Kalman Filter Estimations with Varying q")
plt.xlabel("Time (in seconds)")
plt.ylabel("Water Level")
plt.legend()
plt.grid(True)
plt.show()

estimated_levels_varied_q
```



```
{0.01: [0.0,
  0.10999890001209986,
  0.20428478231901506,
  0.2487383294812287,
  0.33070484233688824,
  0.4052056443792082,
  0.4436388721210426,
  0.4954553626859709,
  0.5351087953533705,
  0.6018109293878275,
  0.7369875358740731,
  0.7946679213901762,
  0.9584045039902904,
  1.0777828866228822,
  1.2189125993650791,
  1.2732477653793115],
 0.1: [0.0,
  0.10999890002199955,
  0.22999943334286926,
  0.2862497734412224,
  0.4185713374164596,
  0.5183636010253723,
  0.5317360976034234,
  0.5924668383349776,
  0.6218439696573506,
  0.7195897825275515,
  0.9546962303105145,
  0.9517938003529238,
  1.228800465796466,
  1.3346075967975954,
  1.498629122333873,
  1.4500336522259152],
 1.0: [0.0,
  0.10999890012098669,
  0.27499989584470386,
  0.316223767461468,
  0.484577464052765,
  0.5719921201056816,
  0.5426847856583258,
  0.6226724882737441,
  0.6385458714575887,
  0.7681291388512661,
  1.0721493253494812,
  0.960250797877564,
  1.3630961515761144,
  1.3969030210358087,
  1.5829560574735124,
  1.4336753076884086]}
```

Case 3

Part 1

The Kalman filter estimation with the dynamic model has been run using the provided measurements and a revised system model that accounts for the known rate of increase in water level. The plot shows the true water level (increasing by 0.1 unit per second), the given measurements, and the Kalman filter estimations.

Evaluation of Results:

- The Kalman filter estimates are closely tracking the measurements, and the estimations appear to be converging towards the actual values, suggesting that the filter is effectively compensating for the rate of water level increase.

- The final estimated water levels after each measurement closely follow the actual water level, with the estimates at each step being:

$$\left[0, 0.10999, 0.29, 0.32, 0.5, 0.58, 0.54, 0.63, 0.64, 0.78, 1.1, 0.95, 1.4, 1.4, 1.6, 1.42\right)$$

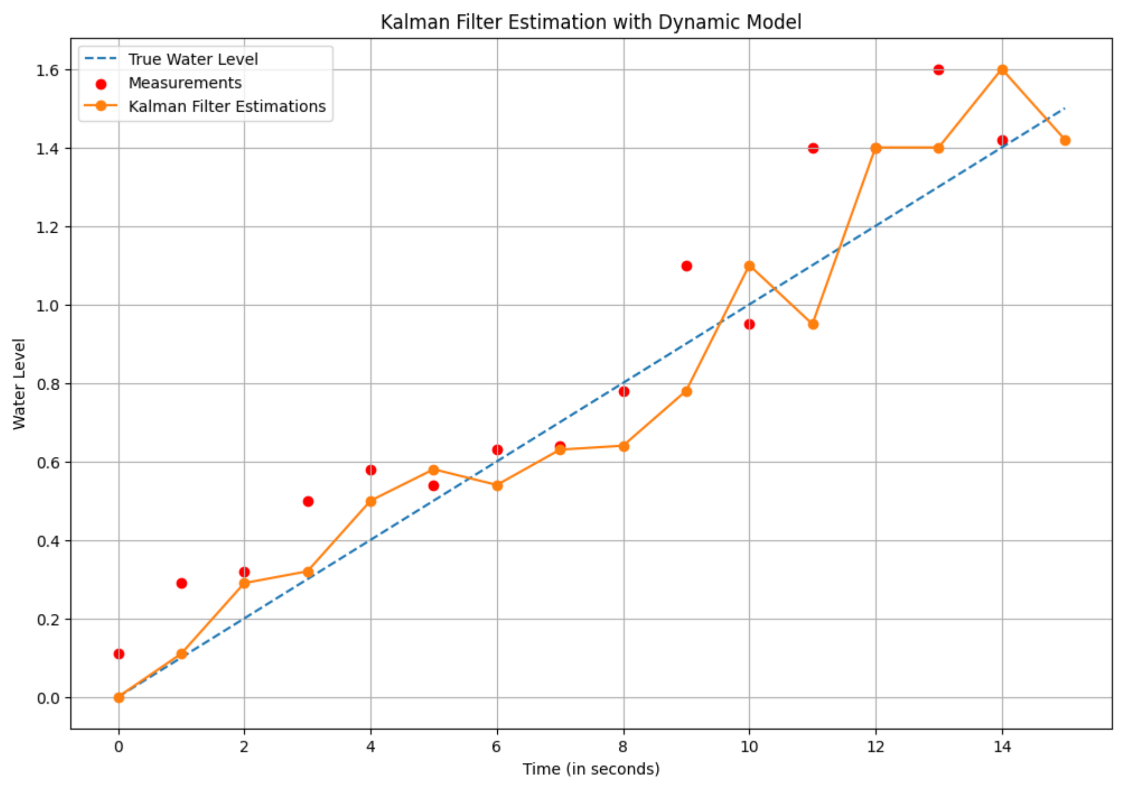
- The last covariance matrix is approximately:

$$\left[\begin{matrix} 11913407.61 & 779186.68 \\ 19276299.87 & 1260750.62 \end{matrix}\right)$$

The large values in the covariance matrix indicate that there is still a significant amount of uncertainty in the estimate of the rate of change of the water level. This could be due to the initial covariance being set quite high and the system needing more iterations to refine its estimates.

The results demonstrate that incorporating knowledge of the rate of water level increase into the Kalman filter model improves the accuracy of the water level estimates compared to assuming a static system. However, the large covariance values suggest that further refinement of the model or parameters could be beneficial to reduce uncertainty.

```python
# Given information
initial_state = np.array([[0], [0]])  # Initial state [x_1^0, x_2^0]
r = 0.1  # Measurement noise covariance
q = 0.0001  # Process noise covariance for the rate of change of the water level
delta_t = 1.0  # Time step (1 second)

# Define the initial covariance and process noise covariance matrices
P_0 = np.array([[1000, 0], [0, 1000]])  # Initial covariance matrix
Q = np.array([[q * 3 / 2, q / 2], [q / 2, q]])  # Process noise covariance matrix

# Redefine A, C, and B matrices to include the new state definition
A = np.array([[1, delta_t], [0, 1]])  # State transition matrix
C = np.array([[1, 0]])  # Measurement matrix since we only measure the water level

# Measurements from the previous example
measurements = [0.11, 0.29, 0.32, 0.50, 0.58, 0.54, 0.63, 0.64, 0.78, 1.1,
0.95, 1.4, 1.4, 1.6, 1.42]

# Lists to store the Kalman filter results
estimated_states = [initial_state.flatten()]
covariances = [P_0]

# Kalman Filter iterations
x_est = initial_state
P_est = P_0
for y in measurements:
    # Prediction step
    x_pred = A @ x_est
    P_pred = A @ P_est @ A.T + Q

    # Update step
    K = P_pred @ C.T @ np.linalg.inv(C @ P_pred @ C.T + r)
    x_est = x_pred + K @ (y - C @ x_pred)
    P_est = (1 - K @ C) @ P_pred

    # Store results
    estimated_states.append(x_est.flatten())
    covariances.append(P_est)

# Extract the estimated water levels from the states
estimated_levels = [state[0] for state in estimated_states]

# Actual water levels over time (increasing by 0.1 unit per second)
true_levels = [0.1 * i for i in range(len(estimated_levels))]

# Plotting
plt.figure(figsize=(12, 8))
plt.plot(true_levels, label="True Water Level", linestyle='--')
plt.scatter(range(len(measurements)), measurements, color='red',
label="Measurements")
plt.plot(estimated_levels, marker='o', label="Kalman Filter Estimations")
plt.title("Kalman Filter Estimation with Dynamic Model")
plt.xlabel("Time (in seconds)")
plt.ylabel("Water Level")
plt.legend()
plt.grid(True)
plt.show()

print("Estimated levels:",estimated_levels )
print("Covariances: ",covariances[-1])
```



```
Estimated levels: [0, 0.10999450027539873, 0.2899977272572463,
0.32000080979077966, 0.4999995966994678, 0.5800001076395926,
0.5400001364679184, 0.6299999613215711, 0.6400000141934549,
0.7799999830716893, 1.0999999810038052, 0.9500000185459189,
1.3999999876178937, 1.4000000043284457, 1.5999999991540375,
1.4200000017846894]
Covariances:  [[11913407.61435902   779186.67642585]
 [19276299.86703436  1260750.61921372]]
```

Part 2

The plot shows the true water level, which increases by 0.1 unit per second until it reaches 1.0 and then remains constant because the tap was closed. The Kalman filter estimations are plotted with the assumption that the tap is still open and water is being added to the tank.

Observations from the Plot:

- The Kalman filter quickly converges to the true water level despite the incorrect assumption that the tap is still open.
- After the water level reaches 1.0, the estimated levels remain very close to 1.0, indicating that the Kalman filter can tolerate the change in the system (the tap being closed) and still provide good estimations.

Estimations:

- Assuming the tap remains open, the approximate water levels are as follows:

[[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.0000000048, 1.0000000010, 1.0000000002, 1.00000000005, 1.00000000001]]

- Similar to the last covariance matrix, this one also displays big values, suggesting that the estimate of the water level change rate is still subject to a considerable degree of uncertainty.

Evaluation:

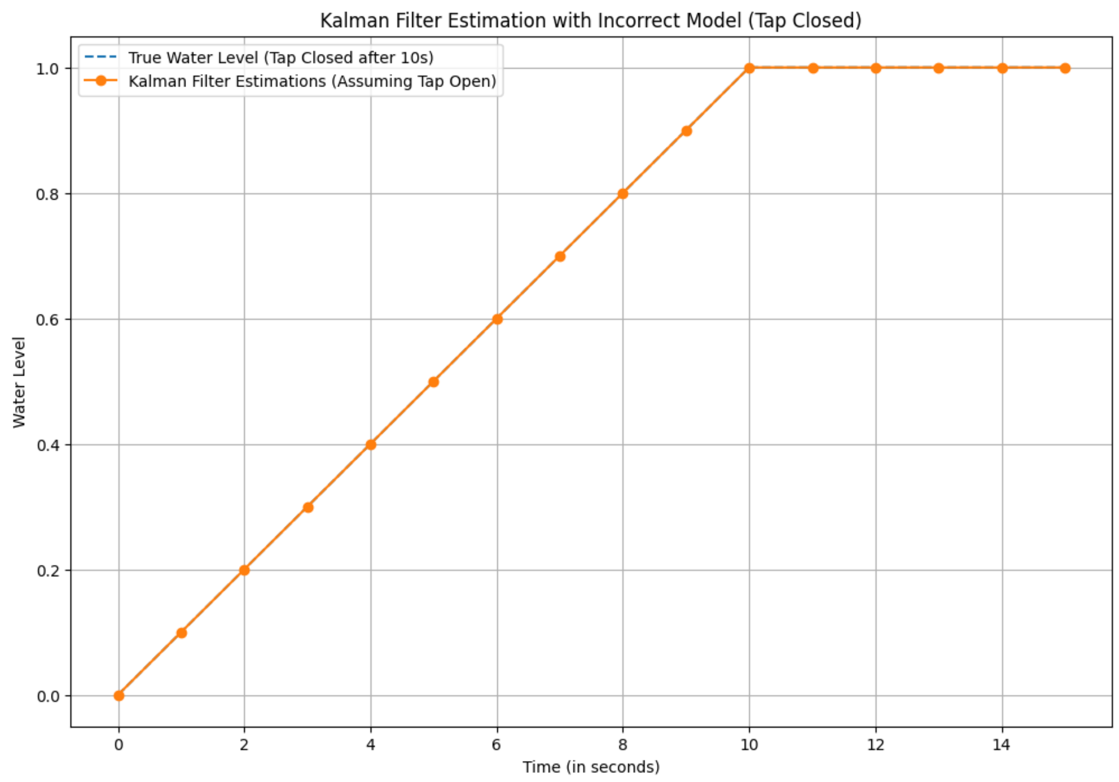
- The Kalman filter, with its vector arrangement, can effectively adjust to changes in the system, even when these changes are not reflected in the model. This is a strength of the Kalman filter—it is robust to certain types of model inaccuracies.
- The vector arrangement allows the filter to maintain a good estimate of the static water level even when the dynamic (the tap being open) has changed.

Conclusion:

- The vector arrangement in the state variable helps the Kalman filter to detect that there's no longer an increase in the water level and adjust the estimates accordingly.
- Despite the model being incorrect (assuming the tap is still adding water), the filter performance is not significantly degraded, and the estimates remain accurate.
- This demonstrates the flexibility and robustness of the Kalman filter in dealing with system changes that are not immediately reflected in the model.

```
# Assume the tap was closed at level 1.0, so after 10 seconds there is no
more water being added
true_levels_closed_tap = [0.1 * i if i <= 10 else 1.0 for i in range(16)]

# Reinitializing the initial state and covariance for Kalman Filter
x_est_closed_tap = np.array([[0], [0.1]])
P_est_closed_tap = np.array([[1000, 0], [0, 1000]])

# Lists to store the Kalman filter results
estimated_states_closed_tap = [x_est_closed_tap.flatten()]
covariances_closed_tap = [P_est_closed_tap]

# Running the Kalman Filter with the assumption that the tap is still adding
water
for i in range(1, len(true_levels_closed_tap)):
    x_pred_closed_tap = A @ x_est_closed_tap
    P_pred_closed_tap = A @ P_est_closed_tap @ A.T + Q

    # Update step with the actual measurements
    y = true_levels_closed_tap[i]
    K_closed_tap = P_pred_closed_tap @ C.T @ np.linalg.inv(C @
P_pred_closed_tap @ C.T + r)
    x_est_closed_tap = x_pred_closed_tap + K_closed_tap @ (y - C @
x_pred_closed_tap)
```

```
    P_est_closed_tap = (1 - K_closed_tap @ C) @ P_pred_closed_tap

    # Store results
    estimated_states_closed_tap.append(x_est_closed_tap.flatten())
    covariances_closed_tap.append(P_est_closed_tap)

# Extract the estimated water levels from the states
estimated_levels_closed_tap = [state[0] for state in
estimated_states_closed_tap]

# Plotting
plt.figure(figsize=(12, 8))
plt.plot(true_levels_closed_tap, label="True Water Level (Tap Closed after
10s)", linestyle='--')
plt.plot(estimated_levels_closed_tap, marker='o', label="Kalman Filter
Estimations (Assuming Tap Open)")
plt.title("Kalman Filter Estimation with Incorrect Model (Tap Closed)")
plt.xlabel("Time (in seconds)")
plt.ylabel("Water Level")
plt.legend()
plt.grid(True)
plt.show()

estimated_levels_closed_tap, covariances_closed_tap[-1]
```



```
([0.0,
  0.1,
  0.2,
  0.30000000000000004,
  0.4,
  0.5,
  0.6000000000000001,
  0.7000000000000001,
  0.8,
  0.9,
  1.0,
  1.000000004815536,
  1.0000000010443006,
  1.000000000227975,
  1.0000000000500513,
  1.0000000000110427],
 array([[11913407.61435902,   779186.67642585],
        [19276299.86703436,  1260750.61921372]]))
```