5 Assignment 5

5.1 Unicycle Robot and Convex Optimization

Consider a robot moving along a straight line controlled according to the unicycle model with inputs v (forward velocity) and ω (angular velocity).

$$\mathbf{x}_{k+1} = \begin{bmatrix} y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} y_k \\ \theta_k \end{bmatrix} + T \begin{bmatrix} v_k \sin\theta_k \\ \omega_k \end{bmatrix} \tag{1}$$

Where the constraints on ω_k are as follows:

$$\omega_{min} \leq \omega_k \leq \omega_{max} \tag{2}$$

Our goal is to move the robot such that we are optimizing the following cost function:

$$\sum_{k=1}^{N} \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \sum_{k=0}^{N-1} r\omega_k^2 \tag{3}$$

where:

$$\mathbf{Q} \in \mathbb{R}^{2\times 2}, \qquad r \in \mathbb{R}, \qquad \mathbf{Q} = \mathbf{Q}^T, \qquad r > 0 \tag{4}$$

5.1.A Convert to Standard Linear Form

We will make the following assumptions in order to simplify our model:

1. $\sin\theta_k = \theta_k$
2. v is constant at each and every time step, so, $v_k = v$ for all k.

We begin by converting our robot dynamics into the form:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{5}$$

Expanding (1) and accounting for our assumptions, we have:

$$y_{k+1} = y_k + Tv\theta_k \tag{6}$$

$$\theta_{k+1} = \theta_k + T\omega_k \tag{7}$$

Rearranging, we get:

$$y_{k+1} = y_k + Tv\omega_k + 0 \tag{8}$$

$$\theta_{k+1} = \theta_k + T\omega_k \tag{9}$$

which we can rewrite as:

$$\begin{bmatrix} y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & Tv \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} 0 \\ T \end{bmatrix} \begin{bmatrix} \omega_k \end{bmatrix} \tag{10}$$

And so we have written our system model in the form of (5) with:

$$\mathbf{x}_{k+1} = \begin{bmatrix} y_{k+1} \\ \theta_{k+1} \end{bmatrix}, \qquad \mathbf{x}_k = \begin{bmatrix} y_k \\ \theta_k \end{bmatrix}, \qquad \mathbf{u}_k = \begin{bmatrix} \omega_k \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} 1 & Tv \\ 0 & 1 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 0 \\ T \end{bmatrix} \tag{11}$$

5.1.B Formulate the Minimization Problem

Next, we explicitly formulate our optimization problem where our cost function (3) is our **objective function**, our system dynamics are our **equality constraints**, and our bounds for ω_k are our **inequality constraints**.

$$\begin{aligned} \underset{\mathbf{x}_k, \mathbf{u}_k}{\text{minimize}} \quad & \sum_{k=1}^{N} \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \sum_{k=0}^{N-1} r\omega_k^2 \\ \text{subject to} \quad & \mathbf{X}' = \mathbf{A}'\mathbf{X} + \mathbf{B}'\mathbf{U} \\ & \Omega_{min} - \mathbf{U} \leq 0, \qquad (element-wise) \\ & \mathbf{U} - \Omega_{max} \leq 0, \qquad (element-wise) \end{aligned} \tag{12}$$

where:

$$\mathbf{X}' = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_N \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_{N-1} \end{bmatrix}, \qquad \mathbf{U} = \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{N-1} \end{bmatrix}, \qquad \mathbf{x}_k = \begin{bmatrix} y_k \\ \theta_k \end{bmatrix}, \qquad \mathbf{u}_k = \begin{bmatrix} \omega_k \end{bmatrix} \tag{13}$$

$$\mathbf{A}' = \begin{bmatrix} A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{bmatrix}, \qquad \mathbf{B}' = \begin{bmatrix} B & 0 & \cdots & 0 \\ 0 & B & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B \end{bmatrix}, \qquad \mathbf{A}' \in \mathbb{R}^{N \times N}, \qquad \mathbf{B}' \in \mathbb{R}^{N \times N} \tag{14}$$

$$\mathbf{A} = \begin{bmatrix} 1 & Tv \\ 0 & 1 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 0 \\ T \end{bmatrix}, \qquad \mathbf{Q} = \begin{bmatrix} Q_1 & 0 & \cdots & 0 \\ 0 & Q_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Q_N \end{bmatrix} \tag{15}$$

$$\Omega_{min} = \begin{bmatrix} [\omega_{min}] \\ [\omega_{min}] \\ \vdots \\ [\omega_{min}] \end{bmatrix}, \qquad \Omega_{max} = \begin{bmatrix} [\omega_{max}] \\ [\omega_{max}] \\ \vdots \\ [\omega_{max}] \end{bmatrix} \tag{16}$$

5.1.C Convexity

The objective function and all constraint functions are convex, thus, this is a **convex optimization** problem, and any local optimum we find can be considered a global optimum. The objective function is convex as it is the sum of two sums, each itself a sum of a series of convex functions. The following function $h(x)$ is convex so long as $f(x)$ and $g(x)$ are convex (as we proved in Assignment 4).

$$h(x) = \sum f(x) + \sum g(x) \tag{17}$$

Now we must show $f(x)$ and $g(x)$ are convex. Consider:

$$f(x) = \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k \tag{18}$$

$$g(x) = r\omega_k^2 \tag{19}$$

We know $f(x)$ is convex by the second derivative condition, where $f'(x) = 2x^T\mathbf{Q}$, and $f''(x) = 2\mathbf{Q}$, which is positive semi-definite, thus $f(x) = \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k$ is convex.

We know w_k^2 is convex as we know graphically the positive square of a variable is convex since the set of points above the graph of a positive square is convex. And scaling a convex function by a constant preserves the convexity of the original function (as we learned in lecture), thus, $g(x) = r\omega_k^2$ is convex. With that, we have shown our objective function is convex.

For our equality constraint, we see that it is simply the sum of two instances of affine scaling (which we know are convex), $\mathbf{A'X}$ and $\mathbf{B'U}$ (within which we have nested Ax_k and Bu_k), and so the sum is also convex (as shown in Assignment 4).

Lastly, our inequality constraints are convex as, when we consider them element-wise (the constraint on each ω_k), they are 1-dimensional constraints bounded above and below. By the intuitive geometric definition of a convex set, if you were to draw a line between any possible ω_k within the bounds of ω_{min} and ω_{max}, any point on the line drawn would also be within the bounds of ω_{min} and ω_{max} and would thus be viable values for ω_k. Thus, our inequality constraints are also convex.

And with that we have shown that our optimization problem we formulated in **5.1.B** is a convex optimization problem since the objective function and all constraint equations are convex.

5.1.D Solve with Python and cvxpy

We will now solve the optimization problem numerically using the `cvxpy` python library.

```
import numpy as np
import cvxpy as cp
from scipy.linalg import block_diag
import matplotlib.pyplot as plt
```

```
    # 5.1.D

def optimize_system(A, B, N, v, T, Q, r):
  X = cp.Variable((2, N))
  U = cp.Variable((1, N-1))
  w = cp.Variable((1, N-1))

  # min (objective)
  objective = cp.Minimize(
    sum(cp.quad_form(X[:, k], Q) for k in range(1, N)) +
    sum(r * cp.square(w[:, k]) for k in range(0, N-1)))

  # s.t. (constraints)
  constraints = []
  ic_input = np.array([1.5, 0])
  ic_output = np.array([0])
  constraints.append(X[:, 0] == ic_input)
  constraints.append(U[:, 0] == ic_output)

  # equality constraints (system dynamics)
  for k in range(N-1):
    constraints.append(X[:, k] == A @ X[:, k-1] + B @ U[:, k-1])
    constraints.append(w[:, k-1] == U[:, k-1])

  # inequality constraints (constrants on w)
  w_min = -np.pi/4
  w_max = np.pi/4
  constraints += [w >= w_min, w <= w_max]

  # solve with cvxpy
  problem = cp.Problem(objective, constraints)
  problem.solve()

  y_values = X.value[0, 0:-1]
  theta_values = X.value[1, 0:-1]
  w_values = U.value[0]

  time_steps = np.arange(N-1)

  return time_steps, y_values, theta_values, w_values
```

```
    # find optimal solution

N = 1000
v = 1
T = 0.01

A = np.array([[1, T * v], [0, 1]])
B = np.array([[0], [T]])

```

```
Q = np.array([[1, 0], [0, 1]])
r = 1

time_steps, y_values, theta_values, w_values = optimize_system(A, B, N, v, T, Q, r)
```

```
    # plot
plt.figure()
plt.plot(time_steps, y_values, color='red')
plt.title("y_k over Time")
plt.xlabel("Time Step")
plt.ylabel("y_k")
plt.show()

plt.figure()
plt.plot(time_steps, theta_values, color='blue')
plt.title("theta_k over Time")
plt.xlabel("Time Step")
plt.ylabel("theta_k")
plt.show()
```

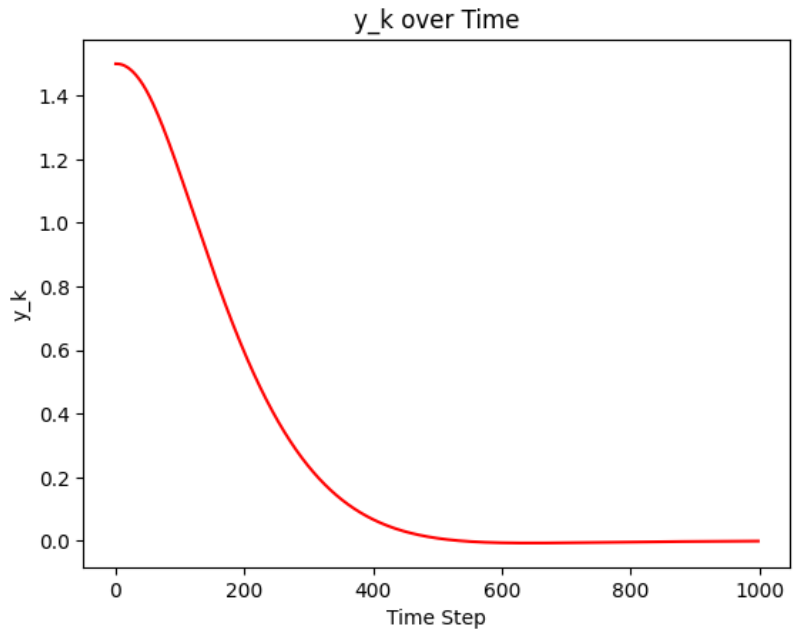

Figure 1: y_k over time

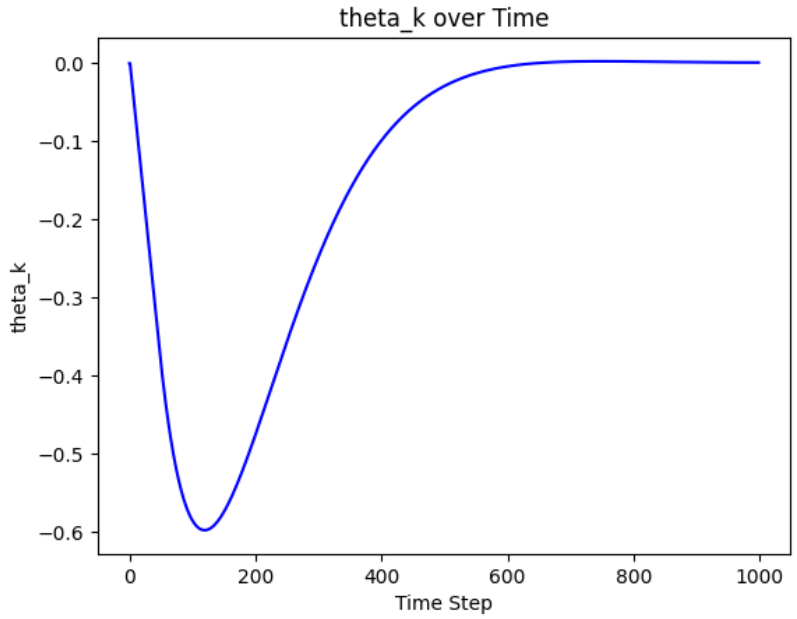

Figure 2: θ_k over time

5.1.E Different Values for Q

We will analyze the impact of changing the diagonal values of $\mathbf{Q}$ on the optimal control sequence. The diagonal values of $\mathbf{Q}$ in the cost function control the weight (or rather, priority) with which each term (y, θ) is considered when minimizing; as we adjust $\mathbf{Q}$, we should see a difference in which values are being minimized faster.

```
    # 5.1.E (different Q)

Q1 = np.array([[1, 0], [0, 1]]) # baseline
Q2 = np.array([[1, 0], [0, 0]]) # optimize only y
Q3 = np.array([[0, 0], [0, 1]]) # optimize only theta
Q4 = np.array([[0, 0], [0, 0]]) # optimize only w
Q5 = np.array([[1, 0], [0, 0.5]]) # double weight y
Q6 = np.array([[0.5, 0], [0, 1]]) # double weight theta
Q_list = [Q1, Q2, Q3, Q4, Q5, Q6]
```

```
plt.figure(figsize=(10, 6))  # Adjust figure size for better visibility

for i, Q in enumerate(Q_list):
    time_steps, y_values, theta_values, w_values = optimize_system(A, B, N, v, T, Q, r)

    if i == 2:  # Q3, using line width 5 for Q3
      plt.plot(time_steps, w_values, label=f'w_k (Q{i+1})', linewidth=5)
    else:
      plt.plot(time_steps, w_values, label=f'w_k (Q{i+1})')

plt.title("w_k over Time for different Q matrices")
plt.xlabel("Time Step")
plt.ylabel("w_k")
plt.legend()
plt.grid(True)
plt.show()


plt.figure(figsize=(10, 6))
for i, Q in enumerate(Q_list):
    time_steps, y_values, theta_values, w_values = optimize_system(A, B, N, v, T, Q, r)
    plt.plot(time_steps, y_values, label=f'y_k (Q{i+1})')

plt.title("y_k over Time for different Q matrices")
plt.xlabel("Time Step")
plt.ylabel("y_k")
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
for i, Q in enumerate(Q_list):
    time_steps, y_values, theta_values, w_values = optimize_system(A, B, N, v, T, Q, r)
    plt.plot(time_steps, theta_values, label=f' _k  (Q{i+1})')

plt.title(" _k  over Time for different Q matrices")
plt.xlabel("Time Step")
plt.ylabel(" _k ")
plt.legend()
plt.grid(True)
plt.show()
```

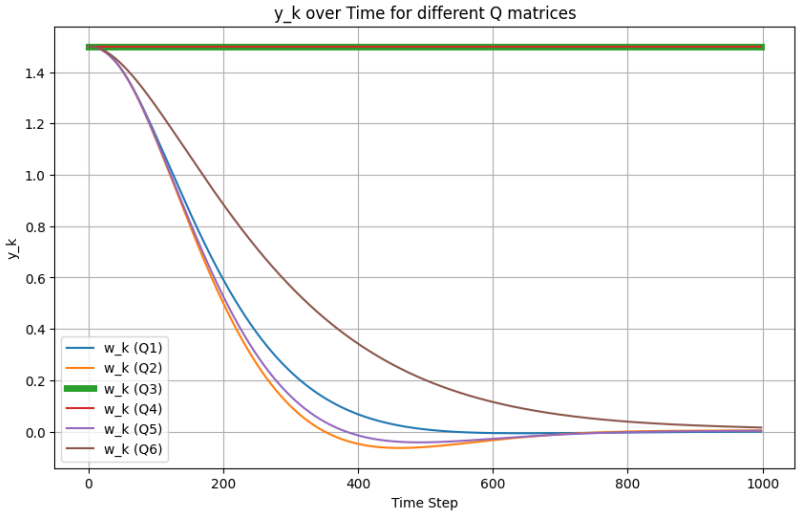

Figure 3: y_k over time for varying Q

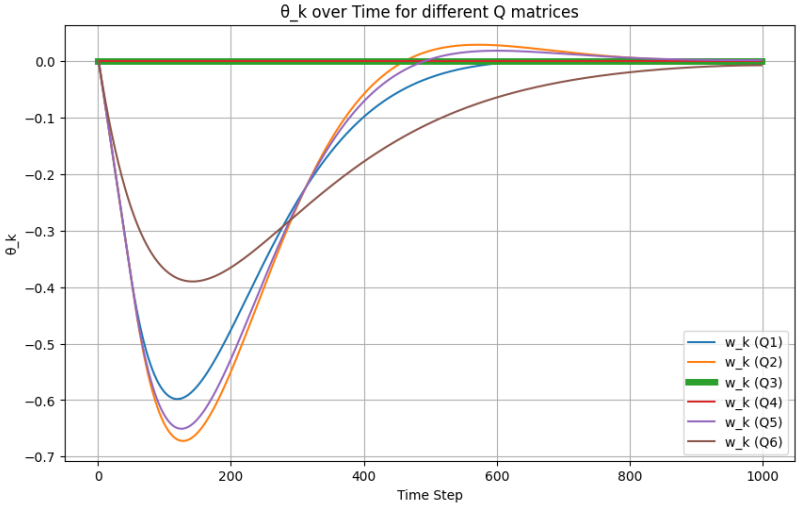

Figure 4: θ_k over time for varying Q

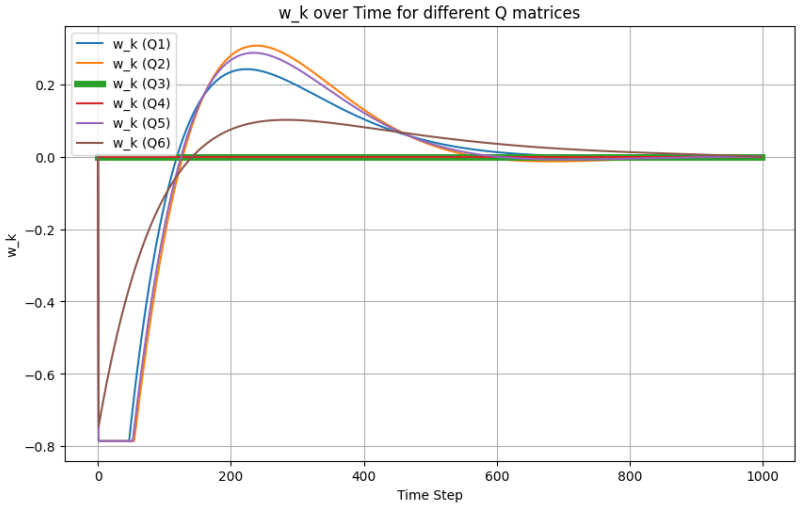

Figure 5: ω_k over time for varying Q

We begin our analysis by comparing the effect of $\mathbf{Q}_2$ (ignoring θ and minimizing only y and ω) and $\mathbf{Q}_3$ (ignoring y and minimizing only θ and ω) on the solution to our optimization problem. When we ran the optimizer with $\mathbf{Q}_2$, θ was essentially ignored in our cost function, and so we see in Figure 3 (plot of y) how the $\mathbf{Q}_2$ curve is declining the steepest, meanwhile in the plot of θ (Figure 4), the $\mathbf{Q}_2$ curve is the one which is allowed to stray the furthest from zero since the explicit θ term is being ignored in the cost function. We see a similar effect with $\mathbf{Q}_3$ in that, since y is ignored and θ and ω are the only things being minimized, θ and ω are initially zero and are thus already minimized, so there is no need to move the robot, and so y remains constant. We can see this phenomenon represented as the straight green line in each of Figure 3, 4, 5 (made thicker to be able to discern it from another line in the same location on each plot).

With $\mathbf{Q}_4$, we have essentially "turned off" y and θ in the cost function, and so we are only minimizing ω, which is initially at zero, and so once again (as with $\mathbf{Q}_3$) we see no change in the state of the robot over time as it is already in the optimal configuration according to this unique cost function with such a value of $\mathbf{Q}$.

With $\mathbf{Q}_5$ the weight we assigned to y was $4\times$ that of the weight assigned to θ. Looking at Figure 3 and 4 we see exactly what we would expect; compared to the baseline (equal weighting), y approaches 0 faster, and θ is allowed to stray further from 0 as it is prioritized less in the cost function. What is interesting to see is that once θ reaches its furthest point from 0, it begins to approach zero even faster than the baseline. This is likely because the $\mathbf{Q}_5$ system has allowed the θ to increase to a point where it is actually higher than the y term even considering the imbalanced weighting assignments, and so it is minimized quicker. Looking at $\mathbf{Q}_6$ we see something similar; y approaches 0 much slower due to the reduced weight, and θ does not stray as far from 0 as before as it is weighted much heavier in the objective function.

One last thing to notice is how in Figure 5, for the majority of trials (different values for $\mathbf{Q}$), ω is instantly sent to its minimum value ω_{min} as the starting position of the robot is far away from zero (and the starting θ and ω are zero), so the highest term in the cost function at time zero is the y term, and so it makes sense that the robot is making an effort to reduce the y term using its control of ω on the vehicle dynamics.

5.1.F Corrupted Dynamics

Now we would like to see how the optimal controls devised in part **5.1.D** would control the robot in an environment where its dynamics were different (corrupted). We will simulate the robot using the following dynamical system:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}'_k + \mathbf{C}\mathbf{x}'_k \tag{20}$$

Where $\mathbf{u}'_k$ is the sequence of optimal controls found in part **5.1.D**, and $\mathbf{x}'_k$ is the sequence of poses in the optimal solution of the robot dynamics in part **5.1.D**. We also have $\mathbf{x}_k$ which is the state of the robot in the new dynamics. $\mathbf{A}$ and $\mathbf{B}$ as the same as before, but $\mathbf{C}$ is new and is here to represent the corruption added to this dynamical system which was not present in part **5.1.D**.

$$\mathbf{C} = \begin{bmatrix} 0 & 0 \\ 0 & \frac{\pi}{25} \end{bmatrix} \tag{21}$$

The purpose of this simulation is to see how the robot will behave in the new (corrupted) environment with controls which were optimized for a different environment. We should expect the robot to exhibit non-optimal behaviour.

```
    # 5.1.F

A = np.array([[1, T * v], [0, 1]])
B = np.array([[0], [T]])
C = np.array([[0, 0], [0, np.pi/25]])
x_k_actual = np.array([[1.5], [0]])
x_k = [np.array([[1.5], [0]])]
u_k = []

# define inputs
for k in range(N-1):
    y = y_values[k]
    theta = theta_values[k]
    x_k.append(np.array([[y], [theta]]))
    w = w_values[k]
    u_k.append(np.array([w]))

# simulate
N = 1000

y_history = []
theta_history = []
for k in range(1, N-1):
    x_k_p_1 = A @ x_k_actual + C @ x_k[k] + B @ u_k[k-1]
    y_history.append(x_k_p_1[0, 0])
    theta_history.append(x_k_p_1[1, 0])
    x_k_actual = x_k_p_1

plt.figure()
plt.plot(time_steps[0:-1], y_history, time_steps, y_values)
plt.legend(["Ideal Environment", "Same Controls in Disturbed Environment"])
plt.title("y_k over Time")
plt.xlabel("Time Steps")
plt.ylabel("y_k")
plt.show()

plt.figure()
plt.plot(time_steps[0:-1], theta_history, time_steps, theta_values)
```

```
plt.legend(["Ideal Environment", "Same Controls in Disturbed Environment"])
plt.title("theta_k over Time")
plt.xlabel("Time Steps")
plt.ylabel("theta_k")
plt.show()
```

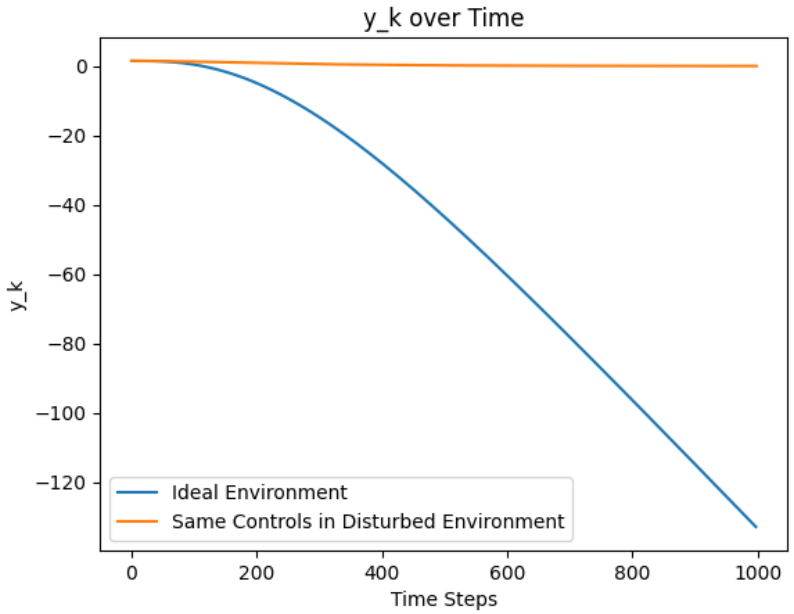

Figure 6: ω_k over time in corrupted dynamics

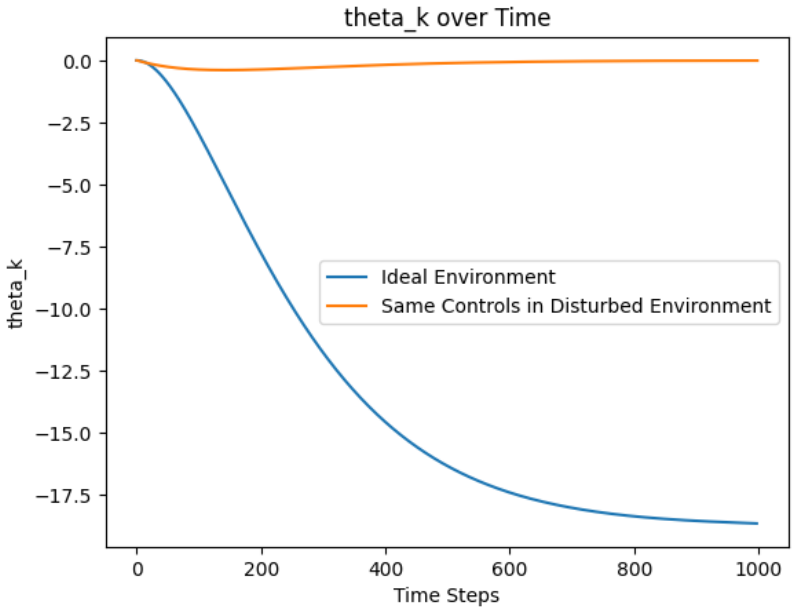

Figure 7: ω_k over time in corrupted dynamics

The problem here is that the robot is seriously diverging from the optimal pose. The corruption acts

explicitly on the θ term, and so even as the angular velocity eventually levels out over time, the robot is pointed in the wrong direction and is still moving away from the optimal y value even when the input angular velocity is close to zero (the dynamics in part **5.1.D** expected that the robot would be near the optimal y and θ values by that time, so it levels out the ω term to ensure it stays on the right track, but in this case, the robot is not on the right track, yet the ω term is not correcting its motion at all). We see the robot still is moving in a straight line (which was actually one of the main goals of the problem), it's just moving in the wrong direction as a result of the corrupted/offset angle measurement.