

Νευρο-ασαφής Έλεγχος

9ο Εξάμηνο 2023 – 2024

Assignment 4 – Solutions

Ζαρίφης Στέλιος – el20435

Email: el20435@mail.ntua.gr

Contents

1 Model Predictive Control	3
1.1 Το Σύστημα	3
1.2 Συνάρτηση Κόστους και Περιορισμοί	3
1.3 Υλοποίηση	3
2 Υπολογισμός του π με Monte Carlo Προσομοίωση	8
3 Parking Problem	10
4 Controlled Random Walk	11

1 Model Predictive Control

Θα χρησιμοποιήσουμε Model Predictive Control για τον έλεγχο της διεύθυνσης ενός αυτοκινήτου. Ο στόχος είναι να ελέγξουμε τη γωνία a μεταξύ της ταχύτητας του οχήματος και της κατεύθυνσης της λωρίδας.

1.1 Το Σύστημα

Η δυναμική του συστήματος δίνεται από τις ακόλουθες εξισώσεις:

$$\begin{aligned}x' &= V \cos a \\y' &= V \sin a \\a' &= u, u \text{ ανάλογο γωνίας στροφής}\end{aligned}$$

Εξετάζουμε μόνο τη δεύτερη και την τρίτη εξίσωση και ύστερα από γραμμικοποίηση και διακριτοποίηση στο χρόνο, έχουμε:

$$x_{k+1} = Ax_k + Bu_k$$

όπου $x_k = [a_k, y_k]^T$.

1.2 Συνάρτηση Κόστους και Περιορισμοί

Η συνάρτηση κόστους ορίζεται ως

$$J = \lambda_1|y_{t+3}| + \lambda_2|a_{t+3}| + \sum_{k=0}^T \left(\lambda_1|y_{t+k}| + \lambda_2|a_{t+k}| + |u_{t+k}| \right)$$

Και οι περιορισμοί του προβλήματος ορίζονται μέσω της δυναμικής του συστήματος:

$$\begin{aligned}|u_{t+k}| &\leq U \\x_{t+k+1} &= Ax_{t+k} + Bu_k\end{aligned}$$

1.3 Υλοποίηση

Με Python ορίζουμε τους περιορισμούς σε κάθε θέση (που προκύπτουν από τη δυναμική του συστήματος) και τη συνάρτηση ελαχιστοποίησης. Με τη βοήθεια της βιβλιοθήκης cvxpy ελαχιστοποιούμε το κόστος, ικανοποιώντας τους περιορισμούς:

```
1 import cvxpy as cp
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Parameters
6 T = 100      # horizon length
7 lambda1 = 1 # weight for y
8 lambda2 = 1 # weight for a
9 U = 1         # maximum absolute value of u
10 Ts = 0.1     # time step
11 V = 1         # speed
12
13 # System matrices
```

```

14 A = np.array([[1, 0], [V*Ts, 1]])
15 B = np.array([Ts, 0.5*V*Ts**2])
16
17 # Variables
18 x = cp.Variable((2, T+1))      # states
19 u = cp.Variable(T)            # control inputs
20
21 # Initial state
22 x0 = np.array([0.5, 0.5])    # replace with your initial state
23
24 # Objective
25 objective = lambda1*cp.abs(x[1, T]) + lambda2*cp.abs(x[0, T])
26 for k in range(T):
27     objective += lambda1*cp.abs(x[1, k]) + lambda2*cp.abs(x[0, k]) + cp.
28     abs(u[k])
29
30 # Constraints
31 constraints = [cp.abs(u) <= U, x[:, 0] == x0]
32 for k in range(T):
33     constraints.append(x[:, k+1] == A @ x[:, k] + B * u[k])
34
35 # Problem
36 problem = cp.Problem(cp.Minimize(objective), constraints)
37
38 # Solve
39 problem.solve()
40
41 # Print optimal control inputs and states
42 print("Optimal control inputs:")
43 print(u.value)
44 print("Optimal states:")
45 print(x.value)
46
47 # Plot states over the horizon
48 plt.figure()
49 plt.subplot(2, 1, 1)
50 plt.suptitle(r'Model Predictive Control ($\lambda_1={}$, $\lambda_2={}$,
51 $T={}$)'.format(lambda1, lambda2, T))
52 plt.plot(x.value[0, :])
53 plt.ylabel('Angle (a)')
54 plt.subplot(2, 1, 2)
55 plt.plot(x.value[1, :])
56 plt.xlabel('Time step')
57 plt.show()

```

Και το αποτέλεσμα του ελέγχου φαίνεται εδώ:

Optimal control inputs:
[-1.0000000e+00 -1.0000000e+00 -1.0000000e+00 -1.0000000e+00]

```

-1.00000000e+00 -1.00000000e+00 -1.00000000e+00 -9.73684211e-01
-3.32874709e-12 -1.48240914e-12 -8.18502793e-13 -4.48458699e-13
-1.87400226e-13  2.96686994e-14  2.34522778e-13  4.48263408e-13
6.89978889e-13  9.82104158e-13  1.35626847e-12  1.86285866e-12
...
...
...
-6.09921831e-15 -5.46221213e-15 -4.78358642e-15 -4.06720390e-15
-3.31607426e-15 -2.53245569e-15 -1.71795231e-15 -8.73608371e-16]
Optimal states:
[[ 5.00000000e-01  4.0000000e-01  3.0000000e-01  2.0000000e-01
1.0000000e-01 -9.72906885e-13 -1.0000000e-01 -2.0000000e-01
-2.97368421e-01 -2.97368421e-01 -2.97368421e-01 -2.97368421e-01
-2.97368421e-01 -2.97368421e-01 -2.97368421e-01 -2.97368421e-01
...
...
...
9.74509856e-15  9.41375933e-15  9.16072439e-15  8.98907574e-15
8.90179130e-15]
[ 5.00000000e-01  5.45000000e-01  5.80000000e-01  6.05000000e-01
6.20000000e-01  6.25000000e-01  6.20000000e-01  6.05000000e-01
5.80131579e-01  5.50394737e-01  5.20657895e-01  4.90921053e-01
4.61184211e-01  4.31447368e-01  4.01710526e-01  3.71973684e-01
3.42236842e-01  3.12500000e-01  2.82763158e-01  2.53026316e-01
...
...
...
-7.03952196e-15 -6.08190860e-15 -5.15342046e-15 -4.24608094e-15
-3.35161089e-15]]

```

Παραθέτουμε στη συνέχεια αποτελέσματα για διάφορες τιμές των παραμέτρων:

Model Predictive Control ($\lambda_1 = 1, \lambda_2 = 1, T = 100$)

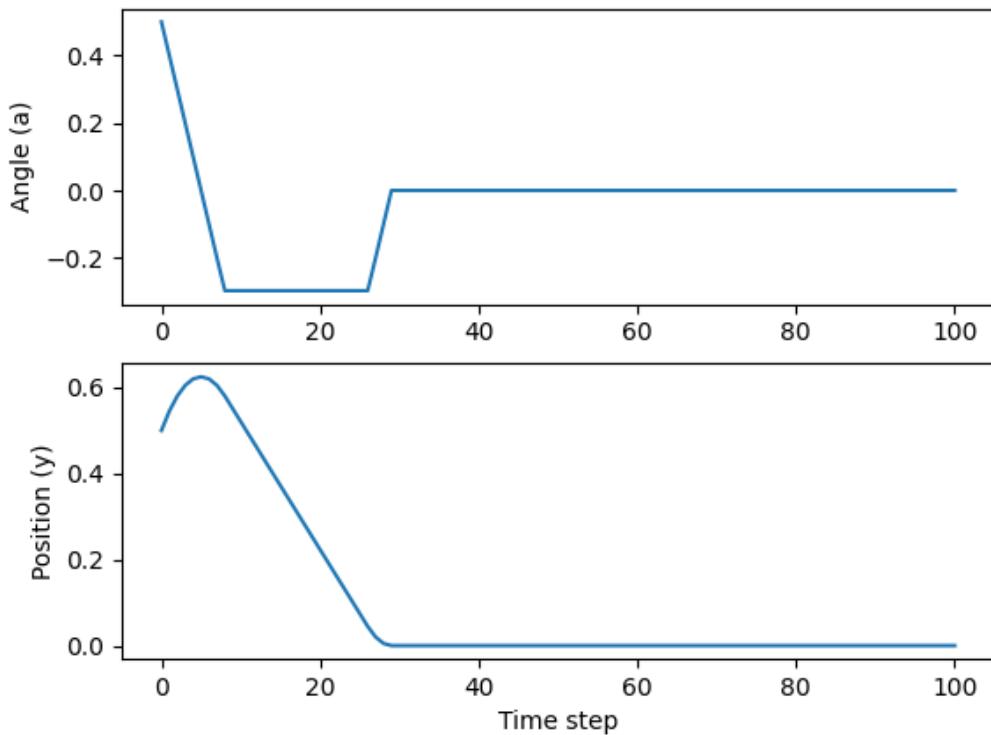


Figure 1: Controlled Position and Angle

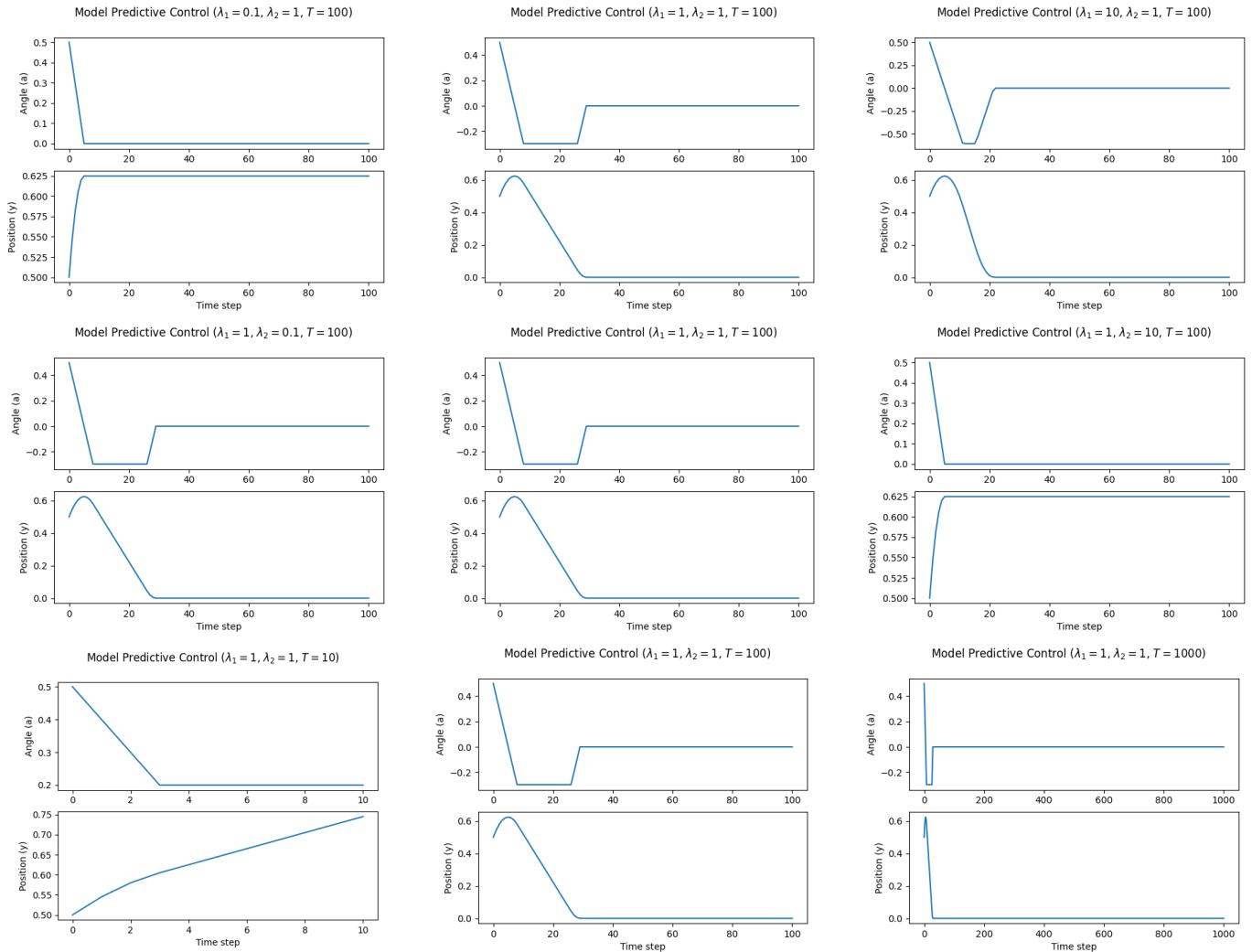


Figure 2: MPC for various parameter values

2 Υπολογισμός του π με Monte Carlo Προσομοίωση

Η μέθοδος Monte Carlo προσεγγίζει λύσεις σε προβλήματα μέσω τυχαίας δειγματοληψίας. Για την προσέγγιση του αριθμού π θεωρούμε ένα τετράγωνο με πλευρά μήκους 2 και τον μοναδιαίο κύκλο με κέντρα την αρχή $(0, 0)$. Παράγουμε ένα μεγάλο αριθμό τυχαίων σημείων (x, y) εντός του τετραγώνου με τα x και y ομοιόμορφα κατανευμένα στο διάστημα $[-1, 1]$. Για κάθε σημείο που δημιουργούμε, εξετάζουμε αν βρίσκεται μέσα στο μοναδιαίο κύκλο. Ο λόγος του αριθμού των σημείων εντός του κύκλου προς τον συνολικό αριθμό των σημείων προσεγγίζει τον λόγο του εμβαδού του κύκλου προς το εμβαδόν του τετραγώνου, δηλαδή δίνει:

$$\frac{S_{circle}}{S_{square}} = \frac{\pi \cdot 1^2}{2^2} = \frac{\pi}{4}$$

Οπότε ο αριθμός π προσεγγίζεται από το τετραπλάσιο του λόγου αυτού. Παραθέτουμε το Python script για τον υπολογισμό:

```
1 import random
2 import numpy
3 import matplotlib.pyplot as plt
4
5 def monte_carlo_pi(num_points):
6     inside_circle = 0
7     for _ in range(num_points):
8         x = random.uniform(0, 1)
9         y = random.uniform(0, 1)
10        if x**2 + y**2 <= 1:
11            inside_circle += 1
12    pi_estimate = 4 * inside_circle / num_points
13    return pi_estimate
14
15 # Actual value of pi
16 actual_pi = numpy.pi
17
18 # List of number of points to try
19 num_points_list = [10, 100, 1000, 10000, 100000, 1000000, 10000000,
20   100000000]
21
22 # Calculate estimated pi and error for each number of points
23 estimated_pi_list = []
24 error_list = []
25 for num_points in num_points_list:
26     estimated_pi = monte_carlo_pi(num_points)
27     estimated_pi_list.append(estimated_pi)
28     error = abs(estimated_pi - actual_pi)
29     error_list.append(error)
30
31 print(f"Estimated value of pi using 100000000 random points: {estimated_pi}")
32
33 # Plotting
34 plt.plot(num_points_list, error_list, marker='o')
```

```

34 plt.xscale('log')
35 plt.yscale('log')
36 plt.xlabel('Number of Points')
37 plt.ylabel('Error')
38 plt.title('Error of Estimated vs. Number of Points')
39 plt.grid(True)
40 plt.show()

```

Και το αποτέλεσμα φαίνεται εδώ:

Estimated value of pi using 100000000 random points: 3.14144392

Ακόμα, βλέπουμε πως το σφάλμα συγκλίνει στο 0 όσο μεγαλώνει ο αριθμός επαναλήψεων των πειραμάτων:

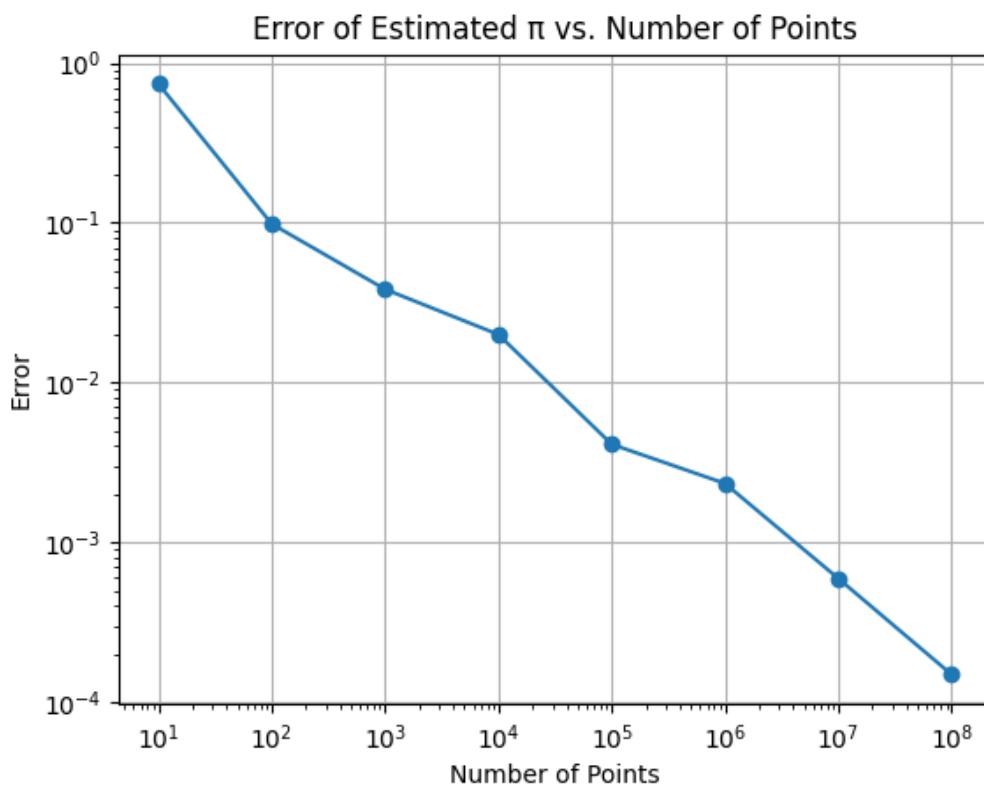


Figure 3: Error of Monte Carlo π Approximation for Various Numbers of Experiments.

