ΡΟΜΠΟΤΙΚΑ ΣΥΣΤΗΜΑΤΑ Ι

1η ΑΣΚΗΣΗ 2023-2024

Ονοματεπώνυμο: Γεώργιος Παπουτσάς

ΑΜ: 1083738

Έτος: 4ο

Contents

GitHub repository: https://github.com/Papiqulos/Ergasia_1_RSI

Modeling the Differential Drive Robot:

Differential Kinematics:

Η διαφορική κινηματική του Ρομπότ υλοποιήθηκε με την συνάρτηση **modeling.diffkin**:

```python
def diffkin(x:np.ndarray, u:np.ndarray)->np.ndarray:
    """
    Differential kinematics model of the differential drive robot

    Args:
        x: state of the robot
        u: control input
    Returns:
        continuous kinematics of the robot
    """
    a = np.array([[-r / ( 2. * d ), r / ( 2. * d )],
                  [( r / 2. ) * np.cos(x[0, 0]), ( r / 2. ) * np.cos(x[0, 0])],
                  [( r / 2. ) * np.sin(x[0, 0]), ( r / 2. ) * np.sin(x[0, 0])]])

    return a @ u
```

Visualization of the states:

Για την οπτικοποίηση του συστήματος ζωγραφίζεται ένα ορθογώνιο με διαστάσεις 4d, 2d (για να φαίνεται καλύτερα, το simulation πραγματοποιείται με τις κανονικές διαστάσεις). Η συνάρτηση που αναπτύχθηκε είναι η **visuals.visualize_states**:

```
def visualize_states(states:list, obstacles:list=[])->None:
    """
    Visualize the states of the robot

    Args:
        states: list of states of the robot
        obstacles: list of obstacles
    Returns: None
    """
    if states:
        fig = plt.figure()
        ax = fig.add_subplot(111)
        # Visualize every second state
        for state in states[::2]:
            # Draw the robot
            rect = Rectangle((state[1,0], state[2,0])
                             , rectangle_width
                             , rectangle_height
                             , edgecolor = "black"
                             , fill=False
                             , angle=state[0, 0] * 180. / np.pi)
            ax.add_patch(rect)

        # Draw the obstacles if any
        if obstacles:
            for o in obstacles:
                ax.add_patch(Circle([o[0], o[1]]
                                    , radius=o[2], fill=False, zorder=3))

        plt.xlim(-5, 5)
        plt.ylim(-5, 5)
        plt.show()
    else:
        return
```

Simulating using Runge-Kutta 4th Order integration

Για το simulation γίνεται χρήση της **modeling.simulate** η οποία χρησιμοποιεί ολοκλήρωση Euler ή Runge Kutta 4ης τάξης:

```
def simulate(x0:np.ndarray, u:np.ndarray, dt:float, T:float, method:str = "rk")-
>list:
    """
    Simulate the robot with Euler or Runge-Kutta 4th order Integration

    Args:
        x0: initial state of the robot
        u: control input
        dt: time step
        T: total time
        method: integration method

    Returns:
        list of states of the robot
    """
    # assert np.any(u < 0.5)
    # Check if control input is valid
    if np.all(u <= 0.5):
        # Number of time steps
        K = int(T/dt) + 1
        # Initial state
        states = [x0]
        # Euler Integration
        if method == "euler":
            for k in range(K):
                x = states[k] + diffkin(states[k], u) * dt
                states.append(x)
        # Runge-Kutta 4th order Integration
        elif method == "rk":
            for k in range(K):
                x = states[k]
                k1 = diffkin(x, u)
                k2 = diffkin(x + k1 * dt / 2, u)
                k3 = diffkin(x + k2 * dt / 2, u)
                k4 = diffkin(x + k3 * dt, u)
                x = x + (k1 + 2 * k2 + 2 * k3 + k4) * dt / 6
                states.append(x)
        return states
    else:
        print("Invalid control input, must be less than 0.5")
        return
```

Στο Figure 1 φαίνεται η τροχιά που πραγματοποιεί το ρομπότ με αρχικό

state = (0, -1, 0),

control input = (0.5, 0.4),

dt = 0.5,

T = 100

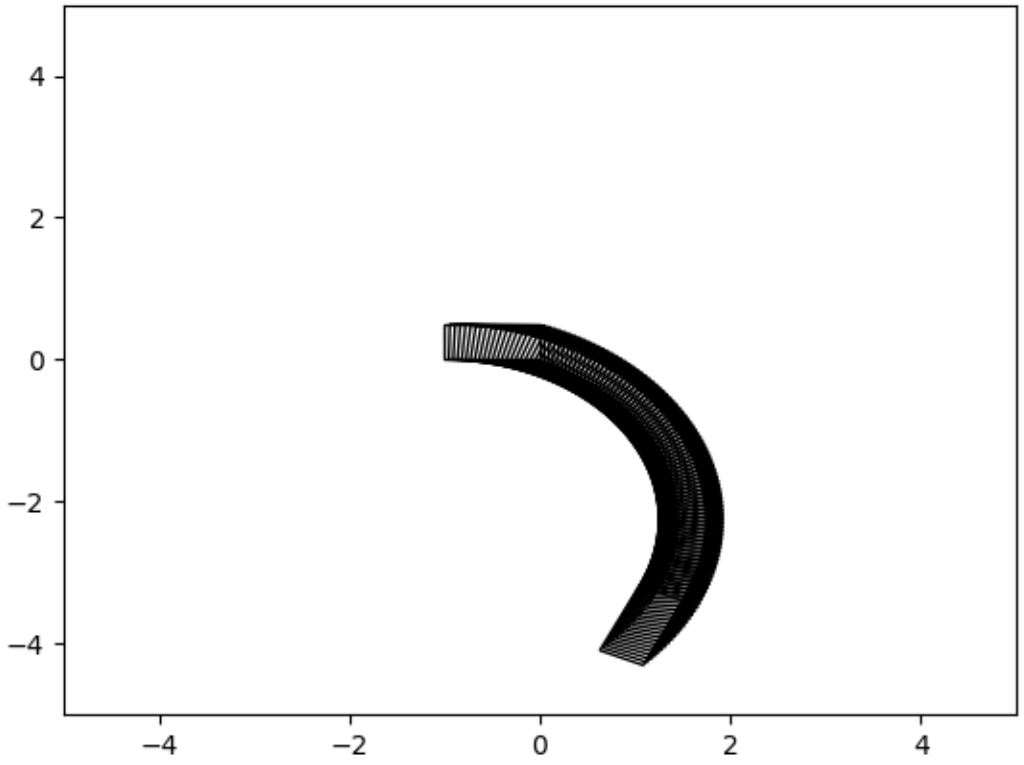

Figure 1: Simulation για δεδομένο control input

RRT Algorithm:

Ο αλγόριθμος υλοποιήθηκε με τα εξής βήματα:

$1:$ Initialize search tree with $\boldsymbol{x}_{\text{start}}$
$2:$ while *some stopping criteria is not met*
$3:$ sample $\boldsymbol{x}_{\text{sample}} \sim \mathcal{X}$
$4:$ find $\boldsymbol{x}_{\text{nearest}}$ nearest node of $\boldsymbol{x}_{\text{sample}}$ in tree
$5:$ connect $\boldsymbol{x}_{\text{nearest}}$ to $\boldsymbol{x}_{\text{new}}$ in direction of $\boldsymbol{x}_{\text{sample}}$
$6:$ **if** success
$7:$ add $\boldsymbol{x}_{\text{new}}$ to the tree with an edge from $\boldsymbol{x}_{\text{nearest}}$
$8:$ **if** $\boldsymbol{x}_{\text{new}} \in \mathcal{X}_{\text{goal}}$ return SUCCESS
$9:$ return FAILURE

Το δέντρο είναι ένα λεξικό όπου κάθε κλειδί είναι ένας γονέας σε tuple και ως value είναι μια λίστα με τα παιδιά του σε διανυσματική μορφή. Οι συναρτήσεις που υλοποιούν τα παραπάνω βήματα είναι:

```
import numpy as np
from modules import distance_points, state_to_tuple, optimal_control, linear_path, collide_obstacles

> def sample_state(state_template:np.ndarray)->tuple: ···

> def connect(x_start:tuple, x_target:tuple, opt:bool, obstacles:list=[])->np.ndarray: ···

> def valid_state(x:np.ndarray, obstacles:list=[])->bool: ···

> def nearest(x_sample:tuple, tree:dict)->tuple: ···

# RRT algorithm
> def RRT(x_start:np.ndarray, x_goal:np.ndarray, opt:bool, max_dist:float, max_iters:int = 1000)->tuple: ···

# RRT algorithm with obstacles
> def RRT_obstacles(x_start:np.ndarray, x_goal:np.ndarray, opt:bool, obstacles:list, max_dist:float, max_iters:int = 1000)->tuple: ···
```

Η connect ενώνει τα δύο states με το optimization routine (για το differential drive robot) που αναφέρεται στην εκφώνηση ή με ένα γραμμικό μονοπάτι.

Εφόσον τρέξει ο αλγόριθμος και δημιουργηθεί το δέντρο μπορούμε να κάνουμε visualize όλο το δέντρο ή μόνο το βέλτιστο μονοπάτι προς το τελευταίο state με τις συναρτήσεις. Το βέλτιστο μονοπάτι βρίσκεται αρχίζοντας από το parent node του τελευταίου state και παίρνοντας τον γονέα του, έπειτα τον γονέα αυτουνού κ.ο.κ. μέχρι να φθάσουμε στην αρχή:

```
> def visualize_tree(tree:dict, obstacles:list=[])->None: ···

> def visualize_best_path(path:list, obstacles:list=[])->None: ···
```

Οι παραπάνω συναρτήσεις δεν δείχνουν την κατεύθυνση του συστήματος εφόσον υπάρχει, μόνο θέση. Μερικά παραδείγματα για ένα σύστημα με δύο μόνο μεταβλητές κατάστασης για την θέση, με αρχή (-2, 0) και στόχο (4, -2):

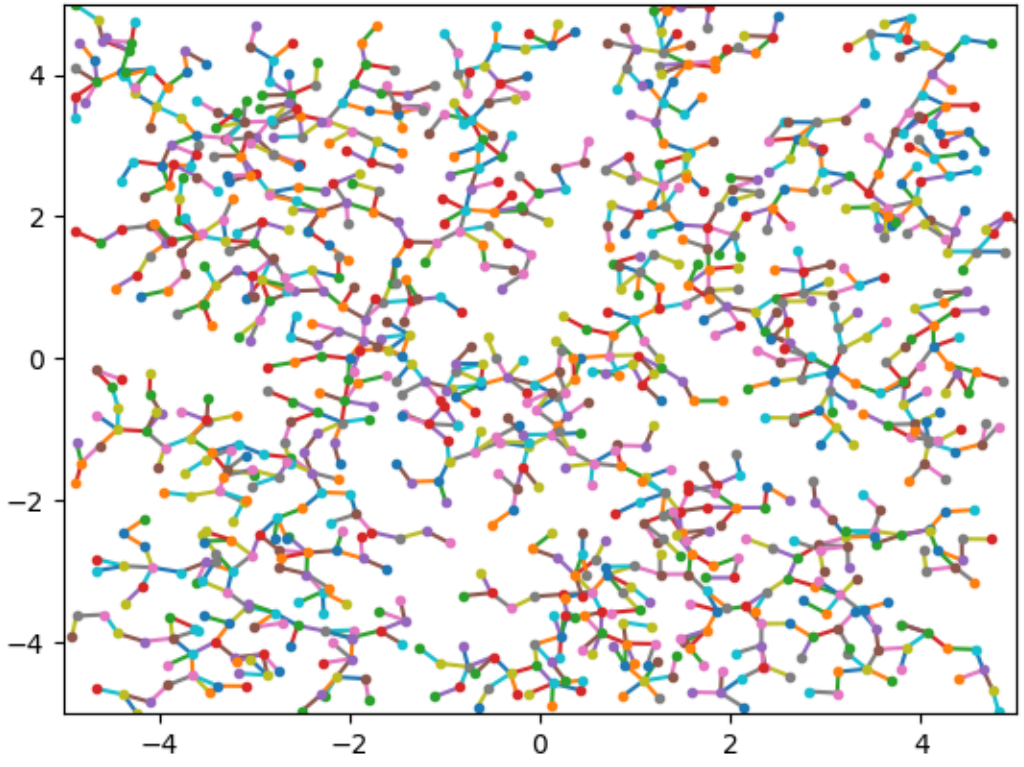

Figure 2: RRT χωρίς εμπόδια

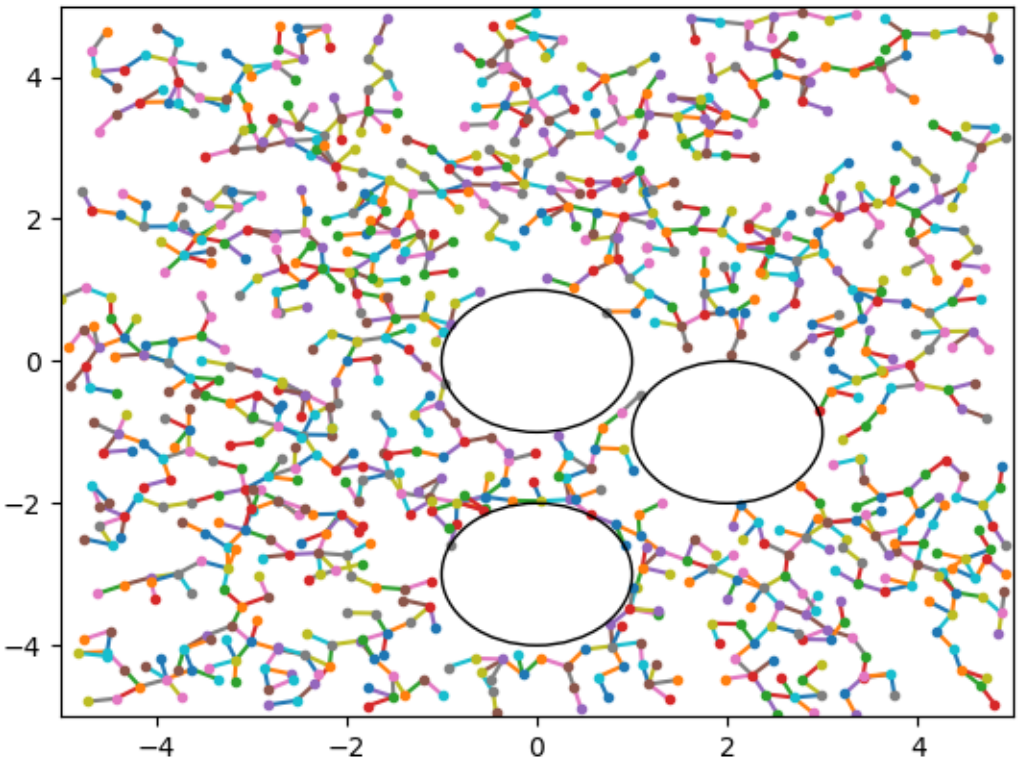

Figure 3: RRT με εμπόδια

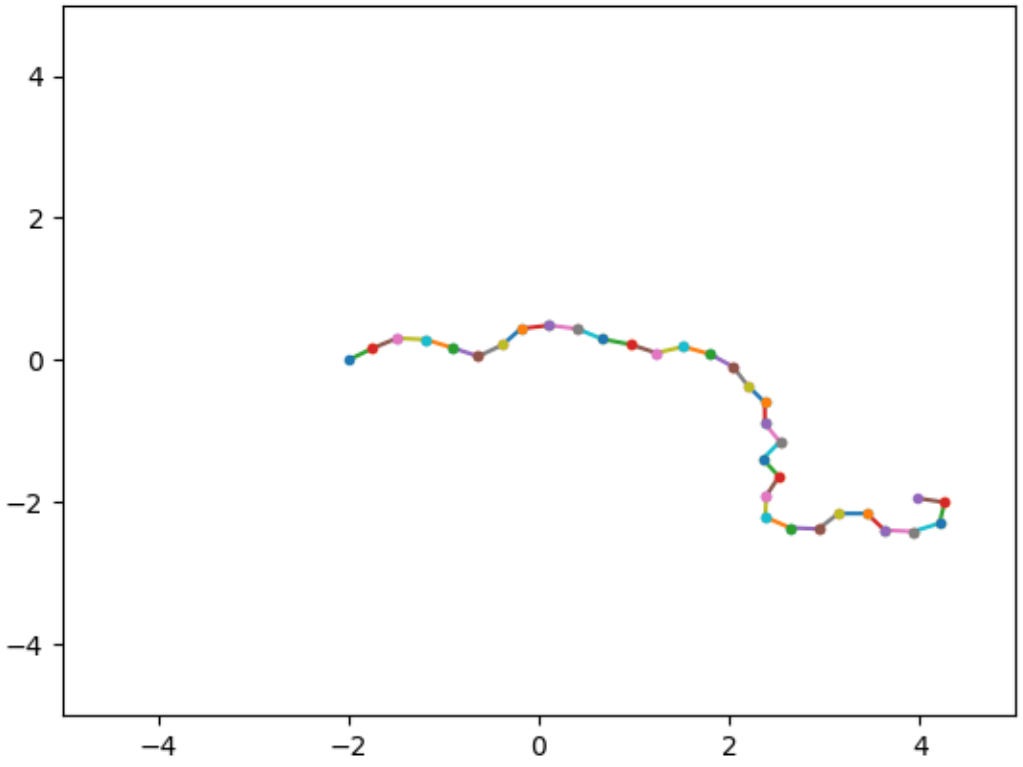

Figure 4: Βέλτιστο μονοπάτι χωρίς εμπόδια

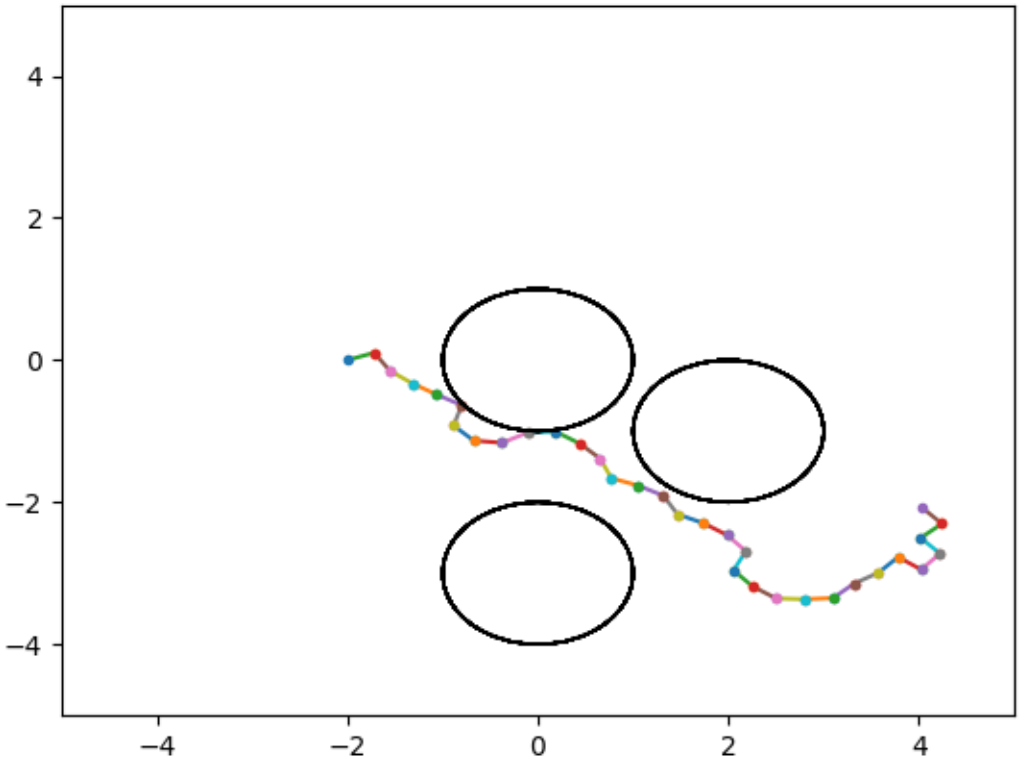

Figure 5: Βέλτιστο μονοπάτι με εμπόδια

Differential Drive Robot Path Following using RRT:

Χρησιμοποιώντας όλες τις μεθόδους που παρουσιάστηκαν παραπάνω, μπορούμε να εφαρμόσουμε το RRT αλγόριθμο για το Robot του Task 1 για ένα αρχικό state προς έναν στόχο. Στην περίπτωση εφαρμογής του RRT στο Robot υπάρχει δυνατότητα οπτικοποίησης κάθε τροχιάς προς κάθε node του δέντρου, όμως δεν έχουν βρεθεί οι βέλτιστες τροχιές και επίσης μερικές τροχιές δεν αποφεύγουν τα εμπόδια. Μερικά παραδείγματα με και χωρίς εμπόδια με αρχικό state (-6, 2, 0) και στόχο (-2, -2, 0):

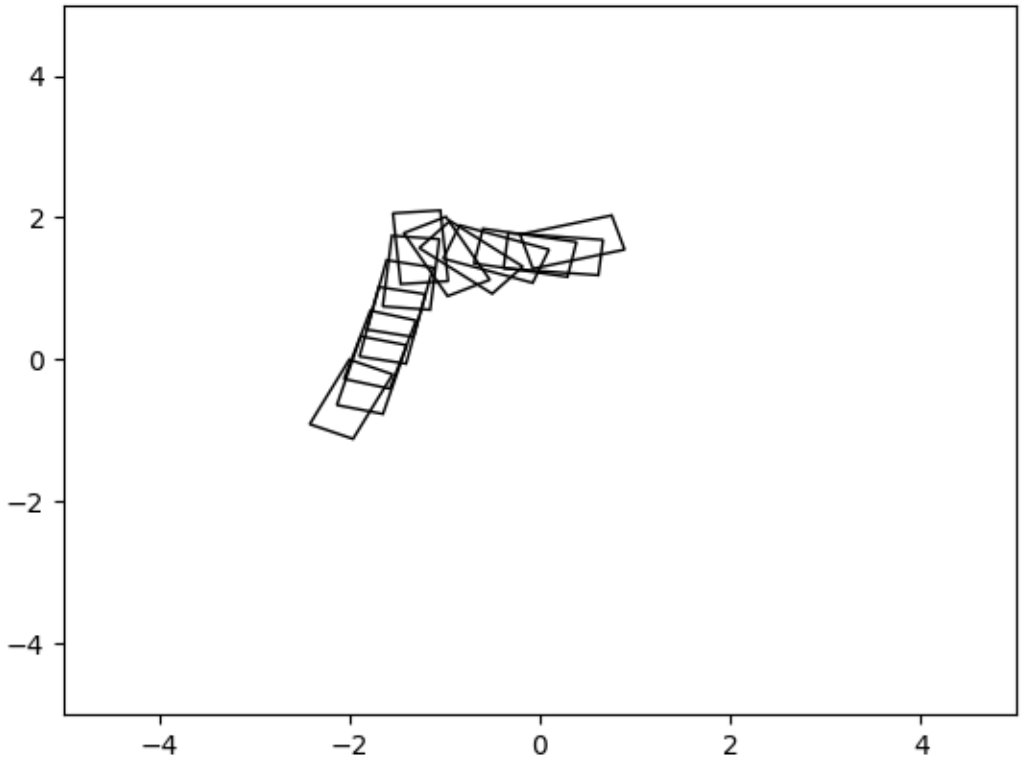

Figure 6: Path following χωρίς εμπόδια

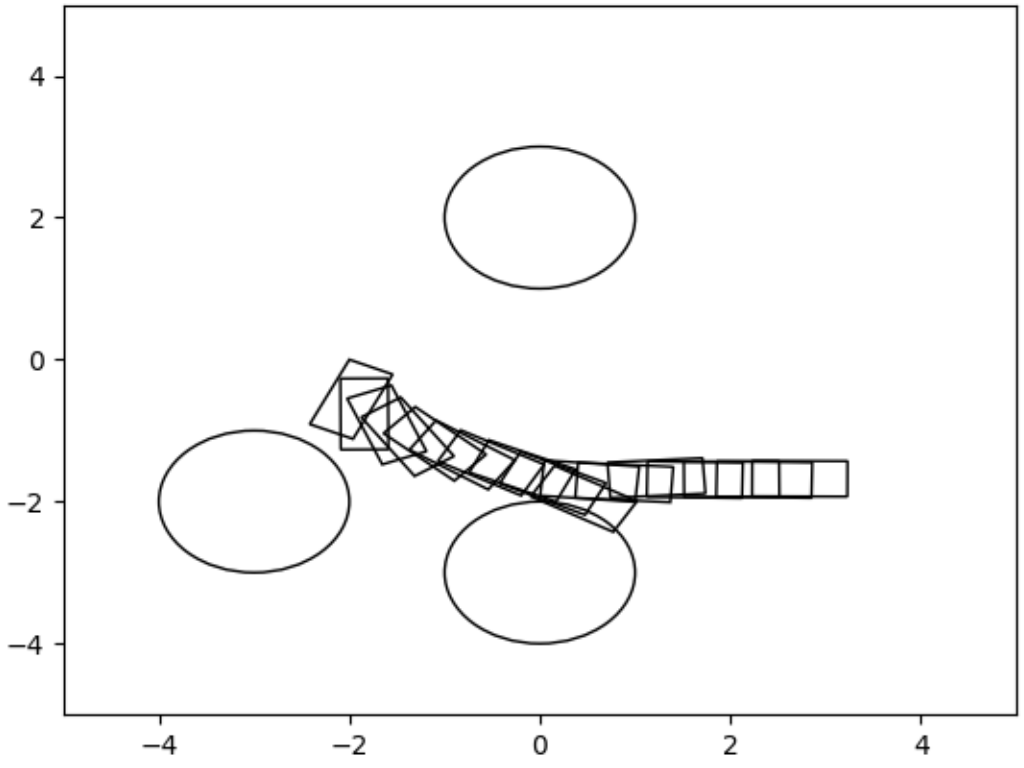

Figure 7: Path following με εμπόδια

Παρατηρούμε ότι χρησιμοποιώντας το optimization routine η εύρεση μονοπατιού είναι πάρα πολύ αργή και σπάνια καταλήγει σε state που είναι ικανοποιητικά κοντά στον στόχο.