Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρ. Μηχ/κών και Μηχ/κών Υπολογιστών
Τομέας Σημάτων, Ελέγχου και Ρομποτικής

« ΕΡΓΑΣΤΗΡΙΟ ΡΟΜΠΟΤΙΚΗΣ »

Άσκηση 4. Cobot

Υπεύθυνος Εργαστηρίου: Κ. Τζαφέστας
Email: ktzaf@cs.ntua.gr
Web: robotics.ntua.gr/members/ktzaf/

Μεταπτυχιακοί Συνεργάτες:
Ιωάννης Αλαμάνος, Email: johnalamanos@protonmail.com
Ιωάννης Καραμπίνας, Email: ioannis.karampinas3773@gmail.com

Παρατήρηση:
Η Άσκηση θα διεξαχθεί στο χώρο του Εργαστηρίου Ρομποτικής και Αυτοματισμού στο Κτήριο Β (Γενικές Έδρες), 2ος Όροφος, τηλ. 210-772-1546.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ/ΚΩΝ & ΜΗΧ/ΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

«Εργαστήριο Ρομποτικής»

Άσκηση 4

MyCobot 280 Jetson Nano

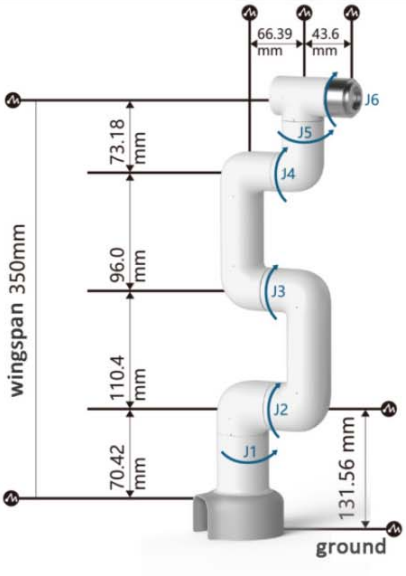
ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ:
MyCobot 280 Jetson Nano

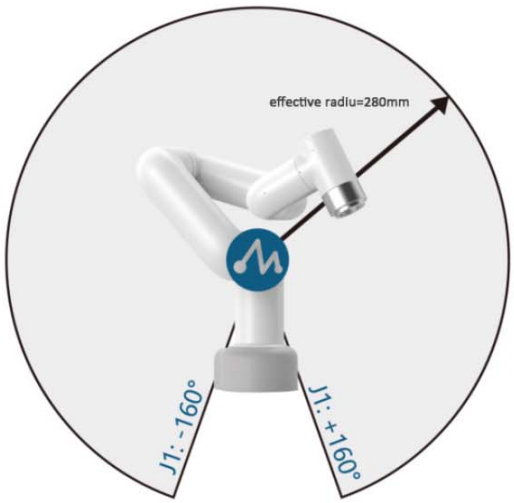
1. Εισαγωγή

Το myCobot 280 Jetson Nano είναι ένας συνεργατικός ρομποτικός βραχίονας έξι βαθμών ελευθερίας που αναπτύχθηκε με βάση τη σειρά myCobot 280. Είναι ένα επίσημο συνεργατικό ρομπότ της NVIDIA και υιοθετεί διπύρηνο έλεγχο του JETSONNANO + ATOM. Το myCobot 280 Jetson Nano ζυγίζει 1030g, αντέχει φορτίο 250g, έχει επαναληψιμότητα κινήσεων με ακρίβεια 5 χιλιοστών, και ακτίνα χεριού 280mm. Παρόλο που είναι μικρό σε μέγεθος, διαθέτει πολλαπλές λειτουργίες. Όχι μόνο λειτουργεί με διάφορους τύπους end effectors για να προσαρμοστεί σε διάφορες εφαρμογές, αλλά υποστηρίζει επίσης τη δευτερογενή ανάπτυξη βασισμένη σε λογισμικά πολλαπλών πλατφορμών για να ικανοποιήσει τις ανάγκες επιστημονικής έρευνας, εκπαίδευσης, smart home, κ.α

2. Χώρος Εργασίας και Εύρος Γωνιών των Αρθρώσεων

Στα ακόλουθα δύο σχήματα εικονίζονται οι κύριες διαστάσεις (μήκη συνδέσμων) του βραχίονα καθώς και τα επιτρεπτά όρια των κινήσεων (εύρος γωνιών) στις αρθρώσεις.

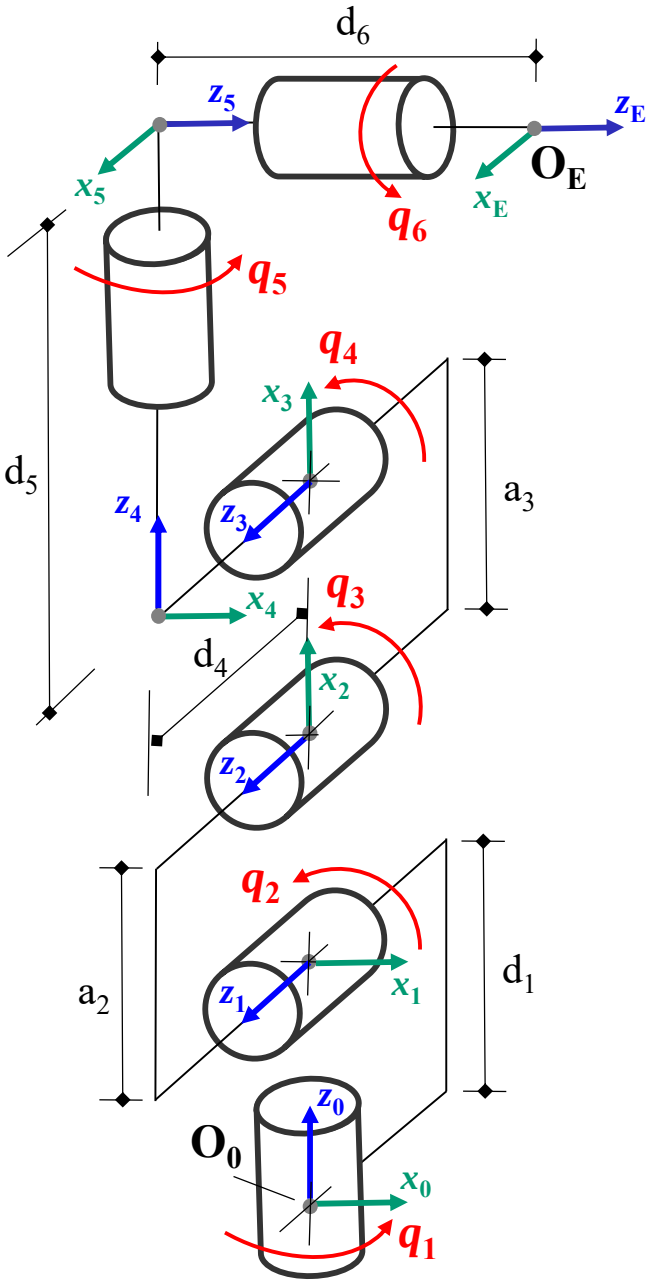




Joint	Angle
J1	-165 ~ +165
J2	-165 ~ +165
J3	-165 ~ +165
J4	-165 ~ +165
J5	-165 ~ +165
J6	-175 ~ +175

3. Διάγραμμα Κινηματικών Προδιαγραφών και Παράμετροι Denavit-Hartenberg

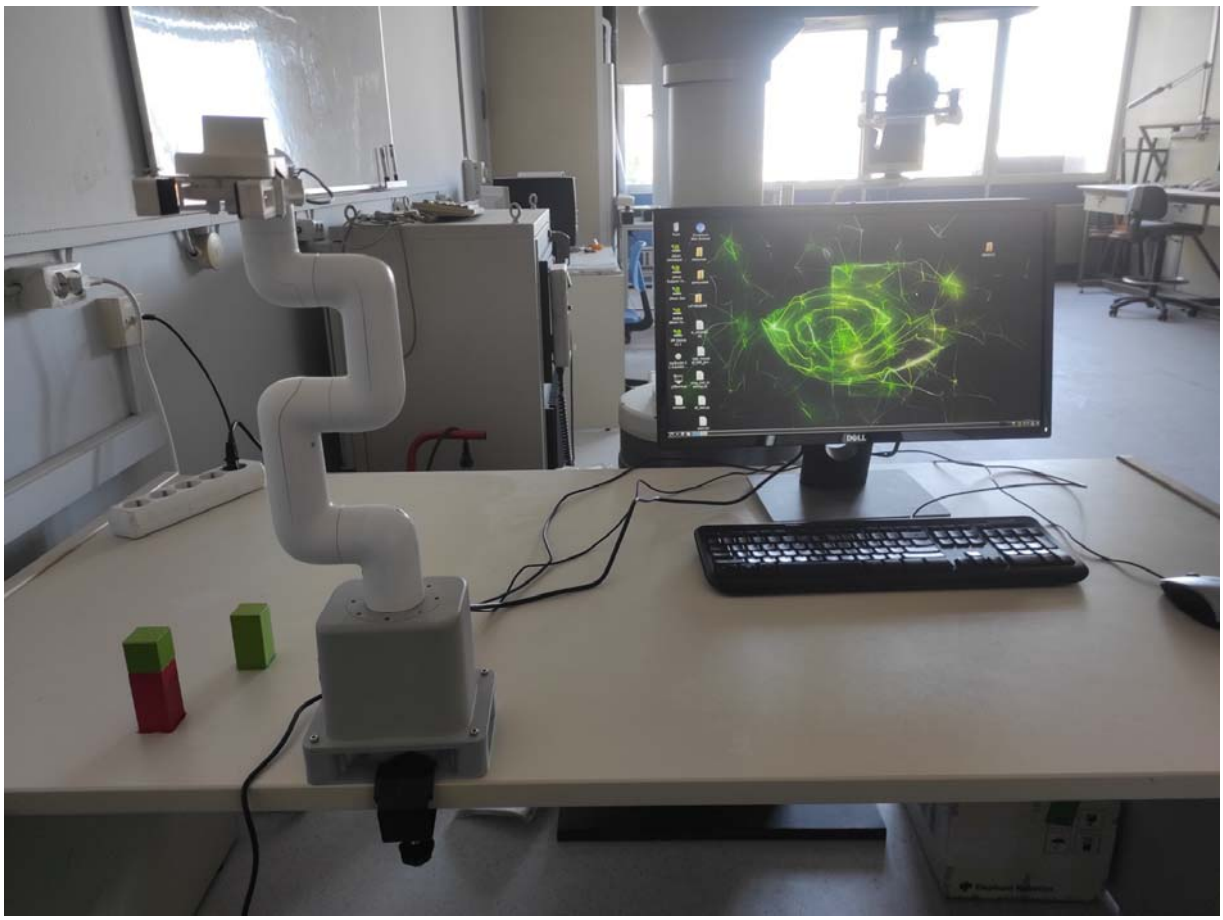
Στα ακόλουθο σχήμα εικονίζεται το κινηματικό διάγραμμα με τις κινηματικές παραμέτρους του βραχίονα καθώς και τα ενδιάμεσα πλαίσια αναφοράς (coordinate frames) τοποθετημένα συμβατικά κατά Denavit-Hartenberg (D-H).



Ο πίνακας των παραμέτρων D-H του βραχίονα είναι ο ακόλουθος:

i	θ_i (rad)	d_i (mm)	a_i (mm)	α_i (rad)
1	q_1	131.22	0	π/2
2	$q_2 + \pi/2$	0	110.4	0
3	q_3	0	96	0
4	$q_4 - \pi/2$	63.4	0	$-\pi/2$
5	$q_5 - \pi/2$	75.05	0	$-\pi/2$
6	q_6	45.6	0	0

4. Εργαστηριακή Άσκηση – Μέρος 1ο (pick-and-place)



Στόχος: Το ζητούμενο του πρώτου μέρους της εργαστηριακής άσκησης είναι ο προγραμματισμός του ρομποτικού βραχίονα ώστε να μετακινήσει με ασφάλεια έναν μικρό κύβο από ένα σημείο Α σε ένα σημείο Β, δηλαδή να εκτελέσει μια **εργασία λήψης-και-εναπόθεσης (pick-and-place)**. Δεδομένου ότι οι καρτεσιανές συντεταγμένες των σημείων λήψης και εναπόθεσης του κύβου θα είναι γνωστές, κύριος στόχος της άσκησης είναι ο προγραμματισμός ενδιάμεσων θέσεων ώστε να εκτελεσθεί με ασφάλεια η κίνηση αλλά και η κατάλληλη προσαρμογή της κατάστασης της αρπάγης (ανοιχτή ή κλειστή).

Προγραμματισμός της κίνησης: Για τον προγραμματισμό του ρομποτικού βραχίονα έχει γραφεί ένα πρόγραμμα σε γλώσσα Python (περιγράφεται παρακάτω) το οποίο αλληλεπιδρά με ένα αρχείο της μορφής ".txt" στο οποίο θα χρειασθεί να προστεθούν κατάλληλες εντολές με τη σωστή σειρά ώστε να εκτελέσει ο βραχίονας τη ζητούμενη λειτουργία. Συγκεκριμένα, οι εντολές που διαβάζονται από το αρχείο και χρειάζονται για τον προγραμματισμό της εργασίας αυτής είναι αποκλειστικά δύο. Η πρώτη εντολή αφορά την κίνηση του βραχίονα σε επιθυμητές καρτεσιανές συντεταγμένες και η δεύτερη την αλλαγή της κατάστασης της αρπάγης και έχουν την εξής μορφή:

1. **set_coords: X, Y, Z**

Για τις ενδιαμεσες θέσεις, όπου X,Y, Z είναι καρτεσιανές συντεταγμένες σε χιλιοστά.

Παράδειγμα εντολής: set_coords: 120.0, 120.0, 170.0

2. **set_gripper_state: S**

Όπου S είναι η κατάσταση της αρπάγης.

Για S=0 η αρπάγη ανοίγει και για S=1 η αρπάγη κλείνει.

Παράδειγμα εντολής: set_gripper_state: 0

3. **set_coords_pick_pos: X, Y, Z**

Για την θέση αρπαγής, όπου X,Y, Z είναι καρτεσιανές συντεταγμένες σε χιλιοστά.

Παράδειγμα εντολής: set_coords: 120.0, 120.0, 170.0

4. **set_coords_place_pos: X, Y, Z**

Για την θέση εναπόθεσης, όπου X,Y, Z είναι καρτεσιανές συντεταγμένες σε χιλιοστά.

Παράδειγμα εντολής: set_coords: 120.0, 120.0, 170.0

Όταν θα γράφονται οι παραπάνω εντολές είναι απαραίτητο να έχουν ακριβώς την μορφή των παραπάνω παραδειγμάτων ώστε να διαβάζονται σωστά από το πρόγραμμα εκτέλεσης της ρομποτικής εργασίας.

Πρόγραμμα Python: Για τις ανάγκες της εργαστηριακής άσκησης έχει αναπτυχθεί ένα πρόγραμμα σε γλώσσα Python το οποίο αλληλεπιδρά με το προαναφερθέν αρχείο “.txt” και μεταφέρει τις εντολές στο ρομποτικό βραχίονα.

Ο σχετικός κώδικας είναι ο εξής:

```python
import time
from pymycobot.mycobot import MyCobot

# Initialize the current line variable
current_line = 12

# Open and read file
try:
    with open('first_task.txt', 'r') as f:
        ft_list = f.readlines()[current_line:]
except FileNotFoundError:
    print("Error: The file was not found")
except IOError:
    print("Error: An I/O error occurred while trying to read the file")

# Initialize a MyCobot instance with the appropriate port and baud rate
mc = MyCobot('/dev/ttyTHS1', 1000000)

# Home position
mc.sync_send_angles([0, 0, 0, 0, 0, 45], 50)  # Instead of send_angles() due to asynchronous return

# Set the gripper state and speed
mc.set_gripper_value(90, 70)

time.sleep(3)

```

```
for line in ft_list:
    current_line += 1

    if line.find("set_coords:") != -1:
        try:
            coords_str = line.split(": ")[1]
            x, y, z = map(float, coords_str.split(","))
            mc.sync_send_coords([x, y, z, -172.0, -2.5, 134.0], 30, 0)
            # mc.send_coords([x, y, z, -172.0, -2.5, 134.0], 30, 0)
            # time.sleep(5)
        except (ValueError, IndexError) as e:
            print("Line ", current_line, "seems to be invalid: ", e)
            continue
    elif line.find("set_coords_pick_pos:") != -1:
        try:
            coords_str = line.split(": ")[1]
            x, y, z = map(float, coords_str.split(","))
            mc.sync_send_coords([x, y, z, -170.96, -1.73, 133.26], 30, 0)
            # mc.send_coords([x, y, z, -170.96, -1.73, 133.26], 30, 0)
            # time.sleep(5)
        except (ValueError, IndexError) as e:
            print("Line ", current_line, "seems to be invalid: ", e)
            continue
    elif line.find("set_coords_place_pos: ") != -1:
        try:
            coords_str = line.split(":")[1]
            x, y, z = map(float, coords_str.split(","))
            mc.sync_send_coords([x, y, z, -173.22, -2.5, 136.73], 30, 0)
            # mc.send_coords([x, y, z, -173.22, -2.5, 136.73], 30, 0)
            # time.sleep(5)
        except (ValueError, IndexError) as e:
            print("Line ", current_line, "seems to be invalid: ", e)
            continue
```

```
60      elif line.find("set_gripper_state:") != -1:
61          try:
62              index = line.find(":")
63              state = int(line[index + 1])
64              if state == 0:
65                  mc.set_gripper_value(90, 70)
66              elif state == 1:
67                  mc.set_gripper_value(10, 70)
68              else:
69                  print(f"Error: invalid state({state}). Valid state is 0 or 1 ")
70          except (ValueError, IndexError) as e:
71              print("Line ", current_line, "seems to be invalid: ", e)
72              continue
73      else:
74          print("Cannot detect a valid command")
75
76  # Return to home position
77  mc.sync_send_angles([0, 0, 0, 0, 0, 45], 50)
78
```

Αρχικά γίνονται import οι κατάλληλες βιβλιοθήκες ώστε να καταστεί εφικτή η επικοινωνία και αλληλεπίδραση με το ρομπότ, και στη συνέχεια διαβάζεται σε μία λίστα το περιεχόμενο του αρχείου το οποίο περιέχει τις εντολές που προαναφέρθησαν, ξεκινώντας από τη γραμμή που βρίσκεται ακριβώς κάτω από τα βοηθητικά σχόλια που θα υπάρχουν στην αρχή του αρχείου. Στη συνέχεια αρχικοποιείται το κατάλληλο instance μέσω του οποίου επικοινωνούμε με τον βραχίονα, δίνοντας ως input την κατάλληλη θύρα και το αντίστοιχο baudrate.

Εν συνεχεία μέσω των εντολών “sync_send_angles([])” και “set_gripper_state()” μετακινείται ο βραχίονας στην αρχική του θέση και ανοίγει η αρπάγη. Όπως φαίνεται και στα σχόλια μπορεί να χρησιμοποιηθεί και η συνάρτηση send_angles(). Η διαφορά μεταξύ των δυο εντολών είναι ότι η send_angles() επιστρέφει άμεσα, χωρίς να περιμένει να ολοκληρωθεί η κίνηση, πράγμα που σημαίνει ότι αν δοθεί νέα εντολή πριν ολοκληρωθεί η κίνηση, η πρώτη κίνηση δεν θα ολοκληρωθεί. Αυτό λύνεται μέσω της εντολής time.sleep(), αλλά αυτή η λύση μπορεί να μην είναι πάντα ακριβής, καθώς ο χρόνος αναμονής ίσως δεν είναι επαρκής ή προκαλεί καθυστέρηση χωρίς λόγο στην εκτέλεση του προγράμματος. Με την αντίστοιχη λογική χρησιμοποιείται η εντολή sleep() και στις εντολές του gripper(). Μια σημαντική παρατήρηση είναι ότι τελικά στον κώδικα, αντί για την εντολή set_gripper_state() χρησιμοποιείται η εντολή set_gripper_value() η οποία καθορίζει το ποσοστό ανοίγματος του gripper. Η επιλογή αυτή έγινε, διότι παρατηρήθηκε πως μερικές φορές η εντολή set_gripper_state() αποτύγχανε.

Ύστερα, μέσω της λίστας που προαναφέρθηκε διαβάζεται γραμμή-γραμμή το αρχείο στο οποίο έχουν εισαχθεί οι εντολές για την εκτέλεση της ζητούμενης ρομποτικής εργασίας. Αν στη συγκεκριμένη γραμμή υπάρχει κάποια εντολή τότε ο κώδικας αποκωδικοποιεί κατάλληλα την εντολή και, εάν δεν υπάρχει κάποιο λάθος στο format της εντολής το οποίο κάνει handle ο κώδικας μέσω κατάλληλων exceptions, τότε η εντολή εκτελείται από τον βραχίονα μέσω των

εντολών “send_coords()”, send_coords_pick_pos(), send_coords_place_pos() ή “set_gripper_state()”, ανάλογα με την εντολή που διαβάσθηκε. Όταν ολοκληρωθεί η αποκωδικοποίηση όλου του αρχείου, τότε αποστέλλεται ξανά ο βραχίονας στην αρχική του θέση.

5. Εργαστηριακή Άσκηση – Μέρος 2ο (ευθεία κινηματική)

Στόχος: Το ζητούμενο του δεύτερου μέρους της εργαστηριακής άσκησης είναι η επαλήθευση της ευθείας κινηματικής εξίσωσης του βραχίονα για τους 3 πρώτους κύριους άξονες του μηχανισμού. Όπως και στην προηγούμενη άσκηση, ο ρομποτικός βραχίονας προγραμματίζεται μέσω ενός ειδικά διαμορφωμένου αρχείου .txt το οποίο διαβάζεται από κατάλληλο πρόγραμμα σε γλώσσα python το οποίο χειρίζεται την επικοινωνία με το ρομπότ. Στη συγκεκριμένη άσκηση, ζητούμενο είναι ο προγραμματισμός του ρομποτικού βραχίονα ώστε να μετακινηθεί σε δύο (τουλάχιστον) διατάξεις (γωνιακές μετατοπίσεις αρθρώσεων) της επιλογής σας, δίνοντάς του ως είσοδο τις επιθυμητές γωνίες των τριών πρώτων αρθρώσεων. Σημειώνεται ότι οι επιθυμητές γωνίες των υπόλοιπων (τελευταίων) τριών αρθρώσεων θα είναι μηδενικές. Με την εκτέλεση κάθε κίνησης σε καθεμία από τις διατάξεις αυτές της επιλογής σας, το πρόγραμμα επιστρέφει τις καρτεσιανές συντεταγμένες του εργαλείου (end-effector), όπως υπολογίζονται εσωτερικά από κατάλληλη εντολή του ρομπότ. Οι φοιτητές καλούνται να καταγράψουν τις γωνίες των αρθρώσεων που αποστέλλονται προς εκτέλεση καθώς και την αντίστοιχη καρτεσιανή θέση του εργαλείου όπως αυτή επιστρέφεται από το πρόγραμμα. Ο στόχος της άσκησης είναι η επακόλουθη επαλήθευση της ευθείας κινηματικής εξίσωσης, η οποία πρέπει να προσδιορισθεί από τους φοιτητές αναλυτικά βάσει του πίνακα παραμέτρων D-H που δίνεται, και η σύγκριση των υπολογισμών με τις τιμές που επιστρέφει το πρόγραμμα ως προς την τελική θέση του εργαλείου σε κάθε κίνηση.

Προγραμματισμός της κίνησης: Σε αυτήν την άσκηση η μόνη εντολή που θα χρειαστεί να εισαχθεί στο αντίστοιχο αρχείο μορφής “.txt” είναι η εξής:

set_angles: A, B, C όπου A,B,C οι γωνίες των τριών πρώτων αρθρώσεων με τη σειρά (σε μοίρες). Παράδειγμα εντολής: set_angles: 40.0, 50.3, 20.1

Όταν γράφεται η παραπάνω εντολή είναι απαραίτητο να έχει ακριβώς την ίδια μορφή με το παραπάνω παράδειγμα ώστε να διαβάζεται σωστά από το πρόγραμμα εκτέλεσης της ρομποτικής κίνησης.

Πρόγραμμα Python: Για τις ανάγκες της εργαστηριακής άσκησης αναπτύχθηκε ένα πρόγραμμα σε γλώσσα Python το οποίο αλληλεπιδρά με το προαναφερθέν αρχείο “.txt” και μεταφέρει τις εντολές στο ρομποτικό βραχίονα.

Ο σχετικός κώδικας είναι ο εξής:

```python
import time
import numpy as np
from pymycobot.mycobot import MyCobot

# Mycobot280 robot DH parameters
# (theta, d, a, alpha offset # mm and radians)
# pi/2 = 1.5708
mycobot_dh_parameters = [
    [0.0, 131.22, 0.0, 1.5708, 0.0],
    [0.0, 0.0, 110.4, 0.0, 1.5708],
    [0.0, 0.0, 96.0, 0.0, 0.0],
    [0.0, 63.4, 0.0, -1.5708, -1.5708],
    [0.0, 75.05, 0.0, -1.5708, -1.5708],
    [0.0, 45.6, 0.0, 0.0, 0.0]
]

# Define transformation (theta, d, a, alpha)
def transform(theta, d, a, alpha):
    return np.array([[np.cos(theta), -np.sin(theta)*np.cos(alpha), np.sin(theta)*np.sin(alpha), a*np.cos(theta)],
                     [np.sin(theta), np.cos(theta)*np.cos(alpha), -np.cos(theta)*np.sin(alpha), a*np.sin(theta)],
                     [0, np.sin(alpha), np.cos(alpha), d],
                     [0, 0, 0, 1]])

# Define forward kinematics (dh_params, joint_angles)
def forward_kinematics(dh_params, joint_angles):
    # Function to compute overall transformation from the base to the end-effector
    T = np.eye(4)
    for i in range(len(joint_angles)):
        theta, d, a, alpha, offset = dh_params[i]  # Unpack DH parameters
        theta += joint_angles[i]
        T = T @ transform(theta + offset, d, a, alpha)  # Compute transformation
    return T  # Return the end-effector position (x, y, z)
```

```python
# Log messages (user_angles_data, and end_effector_position_user_data, joints_data_encoder, cartesian_data_encoder, end_effector_position_encoder_data)
def log_messages(user_angles_data, end_effector_position_user_data, joints_data_encoder, cartesian_data_encoder, end_effector_position_encoder_data):
    print(f"User input -> Joints angles from user: J1:{user_angles_data[0]}, J2:{user_angles_data[1]}, J3:{user_angles_data[2]}, J4:{user_angles_data[3]}, J5:{user_angles_data[4]}, J6:{user_angles_data[5]}\n")
    print(f"End effector position calculation using user input -> X:{end_effector_position_user_data[0][3]}, Y:{end_effector_position_user_data[1][3]}, Z:{end_effector_position_user_data[2][3]}\n")
    print(f"Joints angles from encoders -> J1:{joints_data_encoder[0]}, J2:{joints_data_encoder[1]}, J3:{joints_data_encoder[2]}, J4:{joints_data_encoder[3]}, J5:{joints_data_encoder[4]}, J6:{joints_data_encoder[5]}\n")
    print(f"Cartesian coordinates from encoders -> X:{cartesian_data_encoder[0]}, Y:{cartesian_data_encoder[1]}, Z:{cartesian_data_encoder[2]}\n")
    print(f"End effector position calculation using encoders input -> X:{end_effector_position_encoder_data[0][3]}, Y:{end_effector_position_encoder_data[1][3]}, Z:{end_effector_position_encoder_data[2][3]}\n")
    print("\n")
    time.sleep(1)
```

```python

# Initialize the current line variable
current_line = 5

# Read file
try:
    with open('second_task.txt', 'r') as f:
        ft_list = f.readlines()[current_line:]
except FileNotFoundError:
    print("Error: The file was not found")
except IOError:
    print("Error: An I/O error occurred while trying to read the file")

# Initialize myCobot instance with the appropriate port and baud rate
mc = MyCobot('/dev/ttyTHS1', 1000000)

# Send angles (joint1, joint2, joint3, joint4, joint5, joint6 : speed:range 0-100)
mc.send_angles([0, 0, 0, 45, 30], 30) # Instead of send_angles() due to asynchronous return
mc.set_gripper_state(0, 70)  # Set the gripper state and speed

time.sleep(2)
```

```python
time.sleep(2)

for line in ft_list:
    current_line += 1
    j_angles = []

    if line.find("set_angles:") == -1:
        continue
    if line.find("set_angles:") != -1:
        try:
            coords = line.split(":")[-1].split(",")
            j1, j2, j3 = map(float, coords.strip().split(", "))
            j_angles = [j1, j2, j3, 0.0, 0.0, 45.0]
            mc.sync_send_angles(j_angles, 30)
        except (ValueError, IndexError) as e:
            print(f"Line {current_line} seems to be invalid: {e}")
            continue

    time.sleep(2)

    end_effector_position_user_data = forward_kinematics(mycobot_dh_parameters, np.radians(j_angles))

    current_angles_encoder = mc.get_angles()
    current_coords_encoder = mc.get_coords()

    end_effector_position_encoder_data = forward_kinematics(mycobot_dh_parameters, np.radians(current_angles_encoder))

log_messages(j_angles, end_effector_position_user_data, current_angles_encoder, current_coords_encoder, end_effector_position_encoder_data)

# Return to home position
mc.sync_send_angles([0, 0, 0, 0, 0, 45], 30)

```

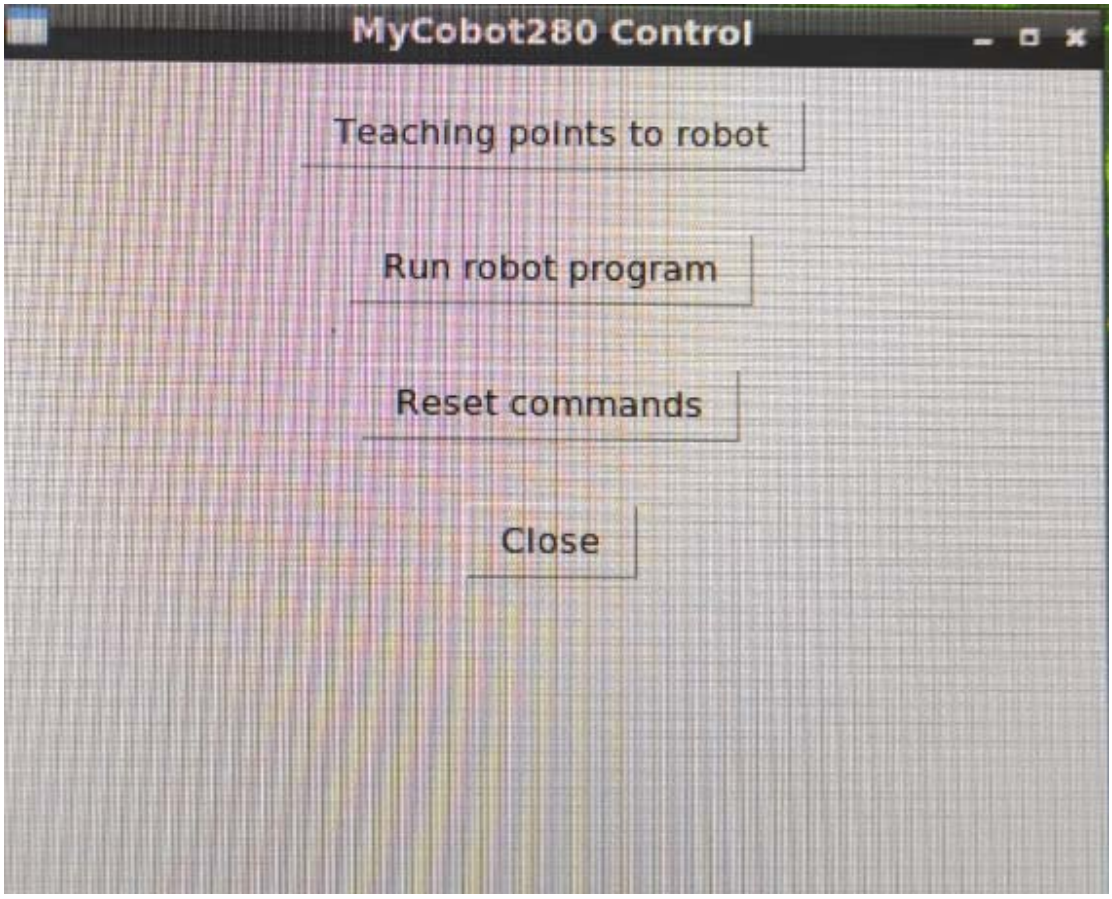
Η δομή του κώδικα παραμένει ίδια με την προηγούμενη άσκηση με τη διαφορά ότι αυτή τη φορά ο βραχίονας εκτελεί κινήσεις μόνο μέσω την εντολής “send_angles([])”. Ακόμη έχει προστεθεί ο υπολογισμός της ευθείας κινηματικής μέσω των Denavit – Hartenberg παραμέτρων. Μετά από κάθε κίνηση, τυπώνονται στο terminal οι γωνίες των αρθρώσεων μέσω της εντολής get_angles() και οι καρτεσιανές συντεταγμένες του end-effector μέσω της εντολής get_coords(). Επιπλέον, τυπώνεται η τελική θέση του end-effector, υπολογισμένη μέσω της ευθείας κινηματικής, τόσο για τις γωνίες που εισήγαγε ο χρήστης όσο και για τις πραγματικές γωνίες από τους encoders. Αυτό επιτρέπει τη σύγκριση των θεωρητικών και των πραγματικών τιμών, ώστε να αξιολογηθεί η ακρίβεια των κινήσεων του βραχίονα.

6. Εργαστηριακή Άσκηση – Μέρος 3ο (pick and place - kinesthetic teaching)

Στόχος: Στο πλαίσιο της συγκεκριμένης άσκησης, η μέθοδος kinesthetic teaching περιλαμβάνει αλληλεπίδραση με τον χρήστη. Σε ορισμένες εφαρμογές, η μέθοδος kinesthetic teaching μπορεί να χρησιμοποιηθεί για να εκπαιδευτεί ένας ρομποτικός βραχίονας μέσω της φυσικής αλληλεπίδρασης με έναν άνθρωπο. Ο χρήστης μπορεί να καθοδηγεί τις κινήσεις του βραχίονα χειροκίνητα, και το ρομπότ "μαθαίνει" την κίνηση αυτή για να την επαναλάβει αργότερα αυτόνομα.

Σε αυτή την άσκηση θα προσπαθήσουμε να υλοποιήσουμε το ίδιο σενάριο του πρώτου μέρους, με μια σημαντική διαφορά. Οι συντεταγμένες της θέσης λήψης και εναπόθεσης θα είναι άγνωστες εκ των προτέρων. Σκοπός είναι να εκπαιδευτεί το ρομπότ μέσω της αλληλεπίδρασης του με τον άνθρωπο και να μετακινείται στον χώρος εργασίας του με ασφάλεια και αποτελεσματικότητα.

Για την ομαλή υλοποίηση της άσκησης δημιουργήθηκε ένα πρόγραμμα σε python. Με την χρήση της βιβλιοθήκης tkinter, αναπτύχθηκε ένα αρκετά απλό και φιλικό ως προς τον χρήστη GUI:



Μέσω αυτού του γραφικού περιβάλλοντος (UI), δίνεται η δυνατότητα στον χρήστη να έχει τον πλήρη έλεγχο του ρομπότ. Συγκεκριμένα, μπορεί να το μετακινεί με τα χέρια του σε οποιοδήποτε σημείο εντός του χώρου εργασίας του ρομπότ.

Βασικές λειτουργίες του UI:

- Teaching mode: Όταν ο χρήστης πατήσει το κουμπί “teaching points to robot”, μπορεί να αποθηκεύσει τη θέση στην οποία βρίσκεται το ρομπότ. Έτσι, ο χρήστης μπορεί να καταχωρίσει διαδοχικές θέσεις, τις οποίες το ρομπότ μπορεί να εκτελεί αυτόνομα με ακρίβεια σε μεταγενέστερο χρόνο.
- Έλεγχος του gripper: Εκτός από τις θέσεις, ο χρήστης έχει τη δυνατότητα να αποθηκεύει εντολές για το gripper, όπως το άνοιγμα και το κλείσιμό του

Αυτές οι επιλογές εμφανίζονται στο UI μέσω διαφόρων κουμπιών που επιτρέπουν τον έλεγχο του ρομπότ και την καταγραφή των σημείων και των εντολών.

Επεξήγηση του Κώδικα:

Ο σχετικός κώδικας βρίσκεται σε δύο αρχεία: το κύριο αρχείο Python(**teaching_points.py)** και το αρχείο **`utils.py`**, όπου έχουν υλοποιηθεί οι βοηθητικές συναρτήσεις. Η χρήση σχολίων και των κατάλληλων ονομάτων στις παραμέτρους των συναρτήσεων έχει ως στόχο να κάνει τον κώδικα εύκολο στην κατανόηση και συντήρηση.

***utils.py**

```python
import time
from pymycobot.mycobot import MyCobot

PORT = "/dev/ttyTHS1"
BAUD_RATE = 1000000
DEFAULT_ROBOT_SPEED = 50
DEFAULT_GRIPPER_SPEED = 70
DEFAULT_GRIPPER_TYPE = 1  # adaptive
DEFAULT_GRIPPER_VALUE_OPEN = 90
DEFAULT_GRIPPER_VALUE_CLOSE = 10
OPEN_GRIPPER = 1
CLOSE_GRIPPER = 0
DEFAULT_MOVEMENT_MODE = 0
HOME_POSITION_JOINT_ANGLES = [0, 0, 0, 0, 0, 45]
INVALID_VALUE = -280  # mycobot280
DEFAULT_TIMEOUT = 7

def is_angle_close(desired_angle, current_angle, threshold):
    return abs(desired_angle - current_angle) < threshold

def is_coordinate_close(desired_value, encoder_value, threshold):
    return abs(desired_value - encoder_value) < threshold

def are_all_angles_close(desired_angles, current_angles, threshold):
    return all([is_angle_close(desired_angles[i], current_angles[i], threshold) for i in range(len(desired_angles))])

def are_all_coordinates_close(desired_coordinates, current_coordinates, threshold):
    return all([is_coordinate_close(desired_coordinates[i], current_coordinates[i], threshold) for i in range(len(desired_coordinates))])

```

```python
def send_coordinates_synchronous(mycobot, coords, speed, mode, timeout):
    mycobot.send_coords(coords, speed, mode)
    if mycobot.is_moving() == -1:
        print("error, the robot seems that it cannot execute the command")
        return False
    if mycobot.is_moving() == 0:
        current_coordinates = mycobot.get_coords()
        if are_all_angles_close(current_coordinates[3:6], coords[3:6], 3) and are_all_coordinates_close(current_coordinates[:3], coords[:3], 6):
            print("The robot is not moving, probably already in the desired position")
            return True
        else:
            print("Command failed to be executed. It will be sent again")
            time.sleep(3) #Give some time to recover from a possible fault situation
            mycobot.send_coords(coords, speed, mode)

    if mycobot.is_moving() == 1:
        start_time = time.time()
        while (time.time() - start_time < timeout):
            current_coordinates = mycobot.get_coords()
            if are_all_angles_close(current_coordinates[3:6], coords[3:6], 3) and are_all_coordinates_close(current_coordinates[:3], coords[:3], 6):
                print(f"Successfully reached the position, desired position: {coords}, current position: {current_coordinates}")
                return True
            time.sleep(0.5)  # I could use pass. Small sleep to prevent excessive CPU usage
        print(f"Failed to reach the position, desired position: {coords}, current position: {current_coordinates}")
        return False
```

```python

def send_angles_synchronous(mycobot, angles, speed, mode, timeout):
    mycobot.send_angles(angles, speed)
    if mycobot.is_moving() == 1:
        print("error, the robot seems that it cannot execute the command")
        return False
    if mycobot.is_moving() == 0:
        current_joint_angles = mycobot.get_angles()
        if are_all_angles_close(angles, current_joint_angles, 3):
            print("The robot is not moving, probably already in the desired position")
            return True
        else:
            print("Command failed to be executed. It will be sent again")
            time.sleep(3) #Give some time to recover from a possible fault situation
            mycobot.send_angles(angles, speed)

    if mycobot.is_moving() == 1:
        start_time = time.time()
        while (time.time() - start_time < timeout):
            current_joint_angles = mycobot.get_angles()
            if are_all_angles_close(angles, current_joint_angles, 2):
                print(f"Successfully reached the position, desired position: {angles}, current position: {current_joint_angles}")
                return True
            time.sleep(0.5)
        print(f"Failed to reach the position, desired position: {angles}, current position: {current_joint_angles}")
        return False
```

```python

def get_current_cartesian_pose(mycobot):
    current_cartesian_pose = mycobot.get_coords()
    print("Cartesian pose is: ", current_cartesian_pose)
    return current_cartesian_pose

def get_current_joints_pose(mycobot):
    current_joints_pose = mycobot.get_angles()
    print("Joints pose is: ", current_joints_pose)
    return current_joints_pose

def move_to_home_position(mycobot):
    mycobot.sync_send_angles(HOME_POSITION_JOINT_ANGLES, DEFAULT_ROBOT_SPEED)

def close_gripper(mycobot):
    #mycobot.set_gripper_state(CLOSE_GRIPPER, DEFAULT_GRIPPER_SPEED)
    mycobot.set_gripper_value(DEFAULT_GRIPPER_VALUE_CLOSE, DEFAULT_GRIPPER_SPEED)
    data = mycobot.is_gripper_moving()
    if data == -1:
        print("Error of closing the gripper")
    elif data == 0:
        print("Gripper is not moving, probably is already close")
    elif data == 1:
        print("Gripper is moving to the desired position")
        while(mycobot.is_gripper_moving()):
            #waiting to finish the command execution
            time.sleep(0.2) #Small sleep to prevent excessive CPU usage
    else:
        print("Unknown error in gripper status")
    time.sleep(1.5)  #Safety factor. In maual they use 3 seconds but we constantly check the gripper if is moving
```

```python
def open_gripper(mycobot):
    mycobot.set_gripper_value(DEFAULT_GRIPPER_VALUE_OPEN, DEFAULT_GRIPPER_SPEED)
    data = mycobot.is_gripper_moving()
    if data == -1:
        print("Error of opening the gripper")
    elif data == 0:
        print("Gripper is not moving, probably is already in the desired position")
    elif data == 1:
        print("Gripper is moving to the desired position")
        while(mycobot.is_gripper_moving()):
            #waiting to finish the command execution
            time.sleep(0.2) #Small sleep to prevent excessive CPU usage
    else:
        print("Unknown error in gripper status")
    time.sleep(1.5) #Safety factor(overkill). In maual they use 3 seconds but we constantly check the gripper if is moving
```

```python

def execute_robot_path(mycobot, poses, gripper_states, robot_speed, movement_mode, timeout, joints_angles_control):
    if len(poses) != len(gripper_states):
        print("Number of poses does not match number of gripper states")
        move_to_home_position(mycobot)
        return False
    for i in range(len(gripper_states)):
        if gripper_states[i] == CLOSE_GRIPPER:
            close_gripper(mycobot)
        elif gripper_states[i] == OPEN_GRIPPER:
            open_gripper(mycobot)
        elif gripper_states[i] == INVALID_VALUE:
            if len(poses[i]) == 6:
                if joints_angles_control == 1:
                    result = send_angles_synchronous(mycobot, poses[i], robot_speed, movement_mode, timeout)
                else:
                    result = send_coordinates_synchronous(mycobot, poses[i], robot_speed, movement_mode, timeout)
            else:
                print("Invalid pose")
        else:
            print("Invalid gripper state")
        time.sleep(1.5) #Let some time pass due to oscillations of braking

    move_to_home_position(mycobot)
```

***teaching_points.py:**

```python
import tkinter as tk
from tkinter import simpledialog
import utils
from utils import *

class RobotArmApp:
    def __init__(self, master, port, baudrate):
        self.master = master
        self.master.title("MyCobot280 Control")
        self.master.geometry("400x300")
        self.master.minsize(400, 300)

        self.port = port
        self.baudrate = baudrate
        self.my_cobot = MyCobot(self.port, self.baudrate)

        self.poses = []
        self.gripper_states = []
        self.joints_angles_control = True
        self.recording_window = None  # Track the recording window

        # Create buttons for GUI control
        self.start_button = tk.Button(master, text = "Teaching points to robot", command = self.teach_robot_path)
        self.start_button.pack(pady = 10)

        self.run_button = tk.Button(master, text = "Run robot program", command = self.run_path)
        self.run_button.pack(pady = 10)

        self.reset_button = tk.Button(master, text = "Reset commands", command = self.reset_commands)
        self.reset_button.pack(pady = 10)

        self.close_button = tk.Button(master, text = "Close", command = self.close_program)
        self.close_button.pack(pady = 10)
```

```python
    def teach_robot_path(self):
        control_type_str = tk.simpledialog.askstring("Choose Control Type", "Enter 'Y' for Joint Control or 'N' for Cartesian Control")
        if control_type_str == 'N' or control_type_str == 'n':
            self.joints_angles_control = False
        else:
            self.joints_angles_control = True

        response = tk.simpledialog.askstring("WARNING", "Get ready, servos will be released. Please, hold the robot carefully. Should we continue? 'Y/N' ")
        if response == "Y" or response == "y":
            self.recording_window = tk.Toplevel(self.master)
            self.recording_window.title("Recording commands")
            self.recording_window.geometry("400x300")
            self.recording_window.minsize(400, 300)
            self.my_cobot.release_all_servos()

            # Create buttons to control the procedure of teaching points in robot
            save_pose_button = tk.Button(self.recording_window, text = "Save pose", command = self.save_pose)
            save_pose_button.pack(pady = 10)

            close_gripper_button = tk.Button(self.recording_window, text = "Catch object", command = self.close_gripper_command)
            close_gripper_button.pack(pady = 10)

            open_gripper_button = tk.Button(self.recording_window, text = "Release object", command = self.open_gripper_command)
            open_gripper_button.pack(pady = 10)

            stop_button = tk.Button(self.recording_window, text = "Stop recording", command = self.stop_recording)
            stop_button.pack(pady = 10)
```

```python

    def save_pose(self):
        joints_pose = get_current_joints_pose(self.my_cobot)
        cartesian_pose = get_current_cartesian_pose(self.my_cobot)
        if self.joints_angles_control:
            pose = joints_pose
        else:
            pose = cartesian_pose

        if pose and all(isinstance(value, (int, float)) for value in pose):
            self.poses.append(pose)
            self.gripper_states.append(INVALID_VALUE)  # In this way, I will check if I should execute command, poses and should have same length

    def open_gripper_command(self):
        self.gripper_states.append(OPEN_GRIPPER)
        self.poses.append(INVALID_VALUE)

    def close_gripper_command(self):
        self.gripper_states.append(CLOSE_GRIPPER)
        self.poses.append(INVALID_VALUE)

    def reset_commands(self):
        self.poses = []
        self.gripper_states = []

    def run_path(self):
        move_to_home_position(self.my_cobot)
        open_gripper(self.my_cobot)
        execute_robot_path(self.my_cobot, self.poses, self.gripper_states, DEFAULT_ROBOT_SPEED, DEFAULT_MOVEMENT_MODE, DEFAULT_TIMEOUT, self.joints_angles_control)

    def stop_recording(self):
        move_to_home_position(self.my_cobot)
        self.recording_window.destroy()

```

```python

    def run_path(self):
        move_to_home_position(self.my_cobot)
        open_gripper(self.my_cobot)
        execute_robot_path(self.my_cobot, self.poses, self.gripper_states, DEFAULT_ROBOT_SPEED, DEFAULT_MOVEMENT_MODE, DEFAULT_TIMEOUT, self.joints_angles_control)

    def stop_recording(self):
        move_to_home_position(self.my_cobot)
        self.recording_window.destroy()

    def close_program(self):
        try:
            move_to_home_position(self.my_cobot)
            self.my_cobot = None
        except Exception as e:
            print(f"Error while shutting down: {e}")
        finally:
            self.master.destroy()

if __name__ == "__main__":
    root = tk.Tk()
    app = RobotArmApp(root, PORT, BAUD_RATE)
    root.mainloop()
```

Είναι χρήσιμο να σχολιασθεί στο σημείο αυτό ότι έχει χρησιμοποιηθεί μια custom μέθοδος για τον έλεγχο του κατά πόσο το ρομπότ έχει φτάσει στην επιθυμητή θέση. Αυτό υλοποιήθηκε για τους εξής λόγους:

1. Παρατηρήθηκαν κάποιες εσφαλμένες εκτελέσεις κατά τη διάρκεια των εκτελέσεων με την sync_send_angles or sync_send_coorsd()
2. Είναι μια αρκετά απλή υλοποίηση αλλά αποτελεσματική, βάση των πειραμάτων που διεξήχθησαν, που έχει ενδιαφέρον να δει ο χρήστης σε μελλοντικές περιπτώσεις όπου, δεν διασφαλίζεται η σύγχρονη εκτέλεση της κίνησης.
3. Δεν δίνεται η υλοποίηση της ενσωματωμένης μεθόδου που χρησιμοποιείται για τον έλεγχο της θέσης του ρομπότ.

7. Παραδοτέο

Να παραδοθεί γραπτή αναφορά, η οποία να περιγράφει τα θεωρητικά και πειραματικά βήματα, καθώς και τα συμπεράσματά σας κατά την εκτέλεση της εργαστηριακής αυτής άσκησης.

- Να περιγράψετε αναλυτικά το setup που χρησιμοποιήθηκε σε αυτή την εργαστηριακή άσκηση. Να αναφερθούν οι βασικές διαφορές μεταξύ robot και cobot, τα χαρακτηριστικά του mycobot280 και σε τι εφαρμογές θα μπορούσε να χρησιμοποιηθεί.
- 1^{ο} μέρος – Να περιγράψετε πώς καταφέρατε να μεταφέρετε τον κύβο με ασφάλεια, γνωρίζοντας τη θέση εναπόθεσης και λήψης και να περιγράψετε ένα σενάριο όπου δεν θα ήταν ασφαλής η κίνηση και γιατί. Κατά τη διάρκεια της εργαστηριακής άσκησης, θα καταγράψετε με το κινητό σας ένα βίντεο της τελικής διαδικασίας και θα το ανεβάσετε ξεχωριστά στην τελική αναφορά. Ακόμη, θα φωτογραφίσετε το txt αρχείο με τις εντολές και τις συντεταγμένες που χρησιμοποιήσατε.
- 2^{ο} μέρος – Να γράψετε αναλυτικά την ευθεία κινηματική χρησιμοποιώντας τη μέθοδο Denavit – Hartenberg και θα υπολογίσετε την τελική θέση του end – effector χρησιμοποιώντας ως γωνίες των αρθρώσεων αυτές των encoders. Κατά τη διάρκεια της άσκησης θα σας δοθούν οι τελικές θέσεις που υπολογίστηκαν στον κώδικα της python με τις γωνίες των αρθρώσεων να ισούνται με αυτές που έδωσε ο χρήστης αλλά και αυτές που επιστρέφουν οι encoder του cobot(οι πραγματικές δηλαδή).
- 3^{ο} μέρος – Να σχολιάσετε τις διαφορές μεταξύ αυτού και του πρώτου μέρους. Με ποιον τρόπο επιτυγχάνεται αυτή η διαδικασία; Ποια είναι τα πλεονεκτήματα αυτής της μεθόδου; Θα καταγράψετε τη διαδικασία σε βίντεο και θα την ανεβάσετε ξεχωριστά..
- Προαιρετικά (δεν θα βαθμολογηθεί): Μελλοντικές βελτιώσεις ή ιδέες σχετικά με την συγκεκριμένη άσκηση.