ΡΟΜΠΟΤΙΚΑ ΣΥΣΤΗΜΑΤΑ Ι

2η ΑΣΚΗΣΗ 2023-2024

Ονοματεπώνυμο: Γεώργιος Παπουτσάς

ΑΜ: 1083738

Έτος: 4ο

Contents

[Modeling and Simulating the Franka Panda manipulator: 3](#_Toc167362542)

[Loading the robot: 4](#_Toc167362543)

[Stepping into the world: 5](#_Toc167362544)

[Visualization of the manipulator 6](#_Toc167362545)

[Task-Space Controller: 7](#_Toc167362546)

[Testing the Simulator and Controller: 8](#_Toc167362547)

GitHub repository: <https://github.com/Papiqulos/Ergasia_2_RSI>

# Modeling and Simulating the Franka Panda manipulator:

A computer screen shot of a program code

Description automatically generatedΔημιουργήθηκε μια κλάση Franka για όλες τις ακόλουθες λειτουργιές που απαιτούνται

## Loading the robot:

Φορτώνουμε όλα τα απαραίτητα δεδομένα του βραχίονα στην **Frank.load\_franka()**:

def load\_franka(self)->tuple[RobotWrapper, pin.Model, pin.Data, pin.GeometryModel, pin.GeometryData]:

        """

        Load the Franka Panda Panda robot model

        Returns:

        - robot: robot wrapper

        - model: pinocchio model

        - data: pinocchio data

        """

        # Load the URDF model

        current\_path = os.path.abspath('') # where the folder `robot` is located at

        robot\_path = os.path.join(current\_path, "robot")

        # Read URDF model

        robot = RobotWrapper.BuildFromURDF(os.path.join(robot\_path, "franka.urdf"), package\_dirs = robot\_path)

        # Extract pinocchio model and data

        model = robot.model

        data = robot.data

        return robot, model, data

## Stepping into the world:

H **Franka.step\_world()** κάνει ένα βήμα dt για το εκάστοτε σήμα ελέγχου ροπής:

def step\_world(self, current\_q:np.ndarray, current\_u:np.ndarray, control\_t:np.ndarray, dt:float)->tuple[np.ndarray, np.ndarray]:

        """

        Step the world for a given time step dt

        Args:

        - model: pinocchio model

        - data: pinocchio data

        - current\_q: current position of each joint (angles of each joint)

        - current\_u: current velocity of each joint

        - control\_t: control torque to be applied at each joint

        - dt: time step

        Returns:

        - new\_q: new position of each joint

        - current\_t: new velocity of each joint

        """

        # Get joint limits

        joint\_limit\_up = self.model.upperPositionLimit

        joint\_limit\_low = self.model.lowerPositionLimit

        velocity\_limit = self.model.velocityLimit

        # Integrate the dynamics and get the acceleration

        aq = pin.aba(self.model, self.data, current\_q, current\_u, control\_t)

        # Integrate the acceleration to get the new velocity

        current\_u += aq \* dt

        # Integrate the velocity to get the new position

        new\_q = pin.integrate(self.model, current\_q, current\_u \* dt)

        # Clip the position if it exceeds the joint limits

        new\_q = np.clip(new\_q, joint\_limit\_low, joint\_limit\_up)

        # Clip the velocity if it exceeds the velocity limits

        current\_u = np.clip(current\_u, -velocity\_limit, velocity\_limit)

        return new\_q, current\_u

## Visualization of the manipulator

Για το visualization γίνεται χρήση της **Franka.visualize()**:

def visualize(self, qs:np.ndarray):

        """

        Visualize the robot

        Args:

        - robot: robot wrapper

        - qs: list of states of the robot at each time step or a single state

        """

        # Visualize the robot

        if len(qs) == self.model.nq:

            self.robot.display(qs)

        else:

            for q in qs:

                self.robot.display(q)

                time.sleep(0.01)

# Task-Space Controller:

A screen shot of a computer program

Description automatically generatedA black background with white text

Description automatically generatedA screenshot of a computer program

Description automatically generatedΗ συνάρτηση που δημιουργήθηκε για τον ελεγκτή είναι η εξής:

Το error signal υπολογίζεται ως εξής:

Με το τωρινό pose profile

Ο ελεγκτής είναι ένας απλός PID ελεγκτής και λειτουργεί ως εξής:

* A screen shot of a computer code

  Description automatically generatedΥπολογίζουμε το τωρινό πίνακα μετασχηματισμού (pose profile)
* Υπολογίζουμε ξεχωριστά το σφάλμα κατεύθυνσης και μετατόπισης και δημιουργούμε το σήμα του σφάλματος

A computer screen with text

Description automatically generated

* Εφαρμόζουμε το σήμα στον PID ελεγκτή

A computer screen with white text and blue text

Description automatically generated

* Υπολογίζουμε το αντίστοιχο σήμα ροπής

A computer screen shot of a computer code

Description automatically generated

* A screen shot of a computer program

  Description automatically generatedΕπαναλαμβάνουμε την διαδικασία για τον null space controller με 2ον task μια διαμόρφωση στο κέντρο των ορίων του βραχίονα

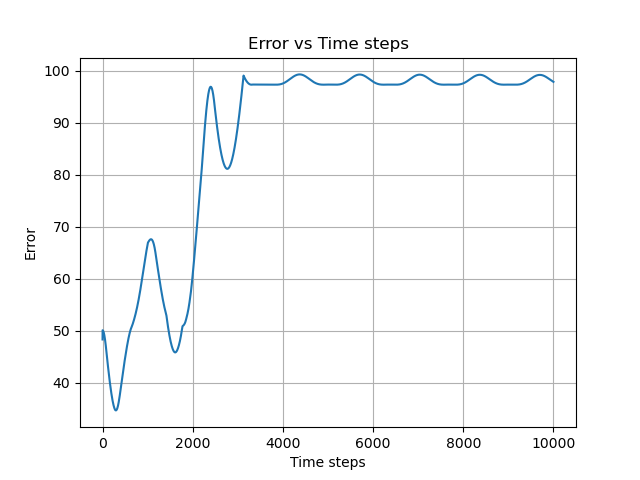
# Testing the Simulator and Controller:

A screenshot of a computer program

Description automatically generated

A computer screen shot of numbers and digits

Description automatically generated

Ο ελεγκτής και για κοντινά και για πιο απομακρυσμένα configurations παρουσιάζει μεγάλα σφάλματα, με τη νόρμα τους να κυμαίνεται συνήθως πάνω από 80. Πιο συγκεκριμένα φαίνεται και στις παρακάτω γραφικές παραστάσεις:

Εικόνα 1:Σφάλμα πρώτης διαμόρφωσης

A graph with blue lines

Description automatically generatedA graph with blue lines

Description automatically generated

Εικόνα 2:Σφάλμα τρίτης διαμόρφωσης

Εικόνα 3:Σφάλμα δεύτερης διαμόρφωσης

A graph with blue lines

Description automatically generated

Εικόνα 4:Σφάλμα τέταρτης διαμόρφωσης