Machine Learning 2025/26

Homework 1: Robot Control

Master in Artificial Intelligence and Robotics

SAPIENZA
UNIVERSITÀ DI ROMA

Professor Luca Iocchi

Homeworks

- Each homework will assign up to 2 points that will be added to the final score of the exam in any session within this academic year.
- Homework points will remain valid independently of acceptance/failure in exam sessions.
- Homeworks are not mandatory.
- It is not possible to deliver homeworks outside the deadline given during the course.

Homework 1 : Robot Control

Deadline: **8/12/2025 23:59 CET** (STRICT DEADLINE!!!)

Assignment through Classroom

Deliver through the assignment:

1) A report (PDF file)

2) A ZIP file with the code you used in the project.

3) [OPTIONAL] Videos of robot control

Assignment through Classroom

Report

- PDF file of about 10 pages excluding code, with your name and matricola code
- implemented solutions (for each robot)
  - how data have been generated/preprocessed
  - which methods/algorithms have been used
  - which configurations of the methods have been tried
  - performance metrics (e.g., plot results over training steps)
- hyper-parameter search
- results for different hyper-parameters
- computational training time
- conclusions and future work

Note: GenAI must be properly acknowledged!

Assignment through Classroom

Submit the files (PDF report, ZIPped code) through this assignment, make sure to turn the assignment in.

NOTES:

1) do ***NOT*** put the PDF report into the ZIP file!

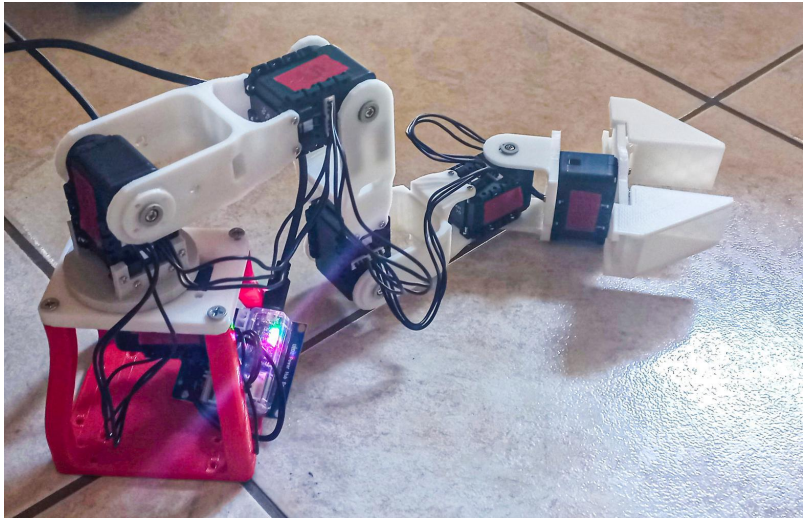
2) no other submission mode will be considered (e.g. do ***NOT*** send submissions by email).

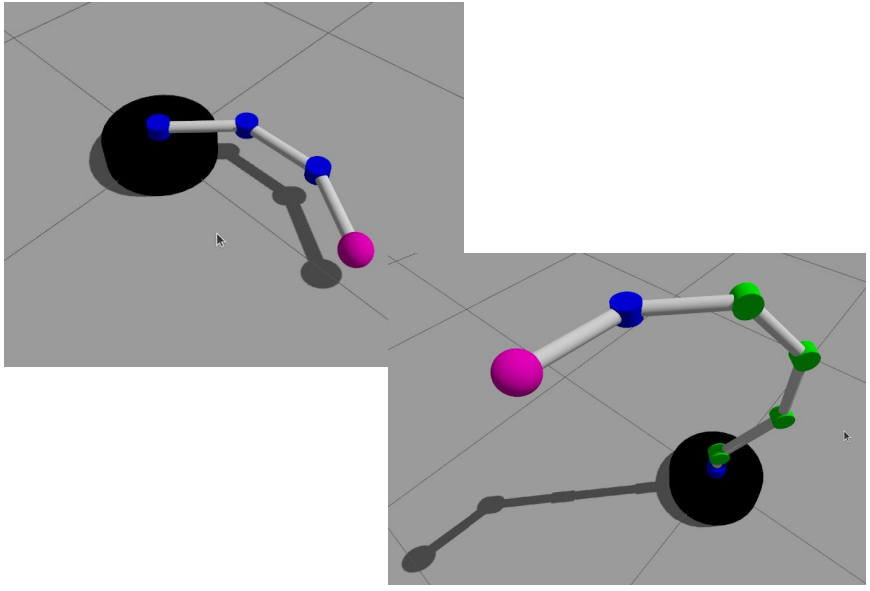
This assignment must be **individual** (i.e., one submission for each student) and **original** (i.e., not equal or too similar to other works either from other students in this class or from other sources).

Evaluation will be based on the appropriateness and correctness of the described solution, regardless of the numeric results (as long as they are reasonable).

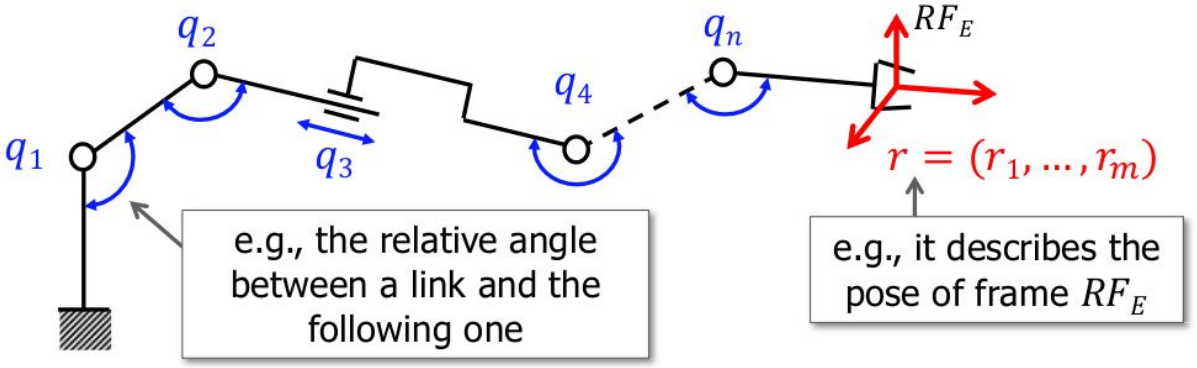
Control of robotic arms

No knowledge of the model => use ML approach





Forward kinematics



Compute pose of EE from joint angles q

Robotics 1 - Prof. Alessandro De Luca

Homework 2024 (NOT VALID FOR 2025 !!!)

Dataset:

- joint angles
- EE position and orientation

Learn forward kinematics (FK) (regression)

- joint angles -> EE position (orientation)

Compute Jacobian matrix J of the learned FK function

Compare learned J with analytical J (using robot model details)
at least for 2 DOF robot arm

HW 2024 - Learning forward kinematics

Train a model for learning forward kinematics (FK) for each collected dataset

- define a model (e.g., feedforward NN) - possibly different for each robot
- choose a loss function
- choose a solver
- fit the model and test performance on validation data
  - random train-test split on the same log
  - train on logs with some seeds, test on logs with other seeds
- do some hyper-parameter search (at least 2 hyper-parameters and 4 combinations)

HW 2024 - Inverse kinematics (optional)

Solve inverse kinematics problem with the learned FK and J

current joints, target pos (orientation) -> target joints

Use the learned Jacobian matrix to implement an inverse kinematics algorithm

- Newton-Raphson
- Levenberg-Marquardt

HW 2024 - Robot Control (optional)

Implement a controller to reach a specific joint configuration

current joints, target joints -> control actions

Use the learned IK to implement robot control function

- PID controller

Integrate with inverse kinematics to reach a specific target position.

HW 2024 - Deep Reinforcement learning (maybe later)

Use DRL to learn policies to reach a specific target position

current joints, target joints -> control actions

Homework 2025

Learn a function for position control in joint space of a robotic arm **without its model**

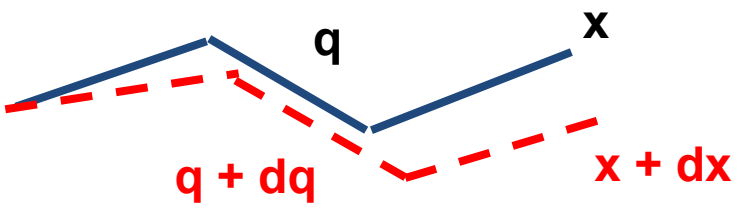
Target function: f (**x, q, dx**) = **dq**

Input:

- current EE position **x**
- current joint angles **q**
- desired EE displacement **dx**

Output:

- joint angles displacement **dq**

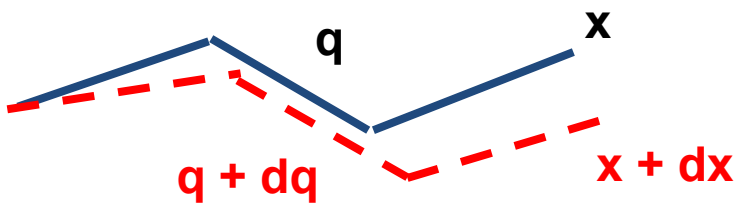


Homework 2025

Dataset acquisition (without model) - just using sensors to measure **x** and **q**

Repeat until enough data:

- **q** <- random value in robot workspace
- position_control (**q**)
- read **x**
- For some steps:
  - **dq** <- small random value
  - position_control (**q** + **dq**)
  - read **x'** , **q'**
  - **dx** = **x'** - **x**, **dq** = **q'** - **q**,
  - collect < **x**, **q**, **dx**, **dq** >

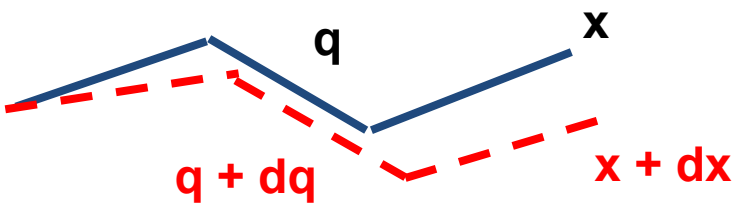


Homework 2025

Problem 1:
x in Re^2 and **q** in Re^3
Input space: Re^7, Output space: Re^3

Problem 2:
x in Re^3 and **q** in Re^4
Input space: Re^10, Output space: Re^4

Problem 3:
x in Re^3 and **q** in Re^6
Input space: Re^12, Output space: Re^6



Homework 2025 datasets (version 1)

Reacher3
x in Re^2 and **q** in Re^3
Input space: Re^7, Output space: Re^3

reacher3_train_1
reacher3_test_1

Reacher4:
x in Re^3 and **q** in Re^4
Input space: Re^10, Output space: Re^4

reacher4_train_1
reacher4_test_1

Reacher6:
x in Re^3 and **q** in Re^6
Input space: Re^12, Output space: Re^6

Homework 2025 datasets (version 2)

Reacher3
x in Re^2 and **q** in Re^3
Input space: Re^7, Output space: Re^3

reacher3_train_2
reacher3_test_2

Reacher4:
x in Re^3 and **q** in Re^4
Input space: Re^10, Output space: Re^4

reacher4_train_2
reacher4_test_2

Reacher6:
x in Re^3 and **q** in Re^6
Input space: Re^12, Output space: Re^6

Dataset file format

CSV file (separator ,)

- EE position
- joint angles
- EE displacement
- joint angles displacement

Read data

```
import pandas as pd

df = pd.read_csv('...csv',sep=' ,',header=0)

X = df[['..', '..', ...]].values # features
Y = df[['..', '..', ...]].values # target
```

Simulator environment

https://github.com/iocchi/marr_gz

Config file to launch the arm robot

```
robot:
  base: reacher
  dof: 3
  arms_control_interface: position
```

Reacher with 3 dof is a planar robot, with 4,5,6 dof is a 3D robot

Simulator environment - Data acquisition

Subscriber and callback to read joint angles **q**

```
# Subscriber
self.joint_sub = self.create_subscription(
  JointState, '/joint_states',
  self.joint_state_callback, 10 )
# Callback
def joint_state_callback(self, msg):
  arm_indices = [i for i, name in enumerate(msg.name)
                        if 'arm_joint' in name]
  self.q = [ msg.position[arm_indices[i]]
                        for i in range(self.njoints) ]
```

Simulator environment - Data acquisition

Link transform to read EE position **X**

```
# tf listener
self.tf_buffer = Buffer()
self.tf_listener = TransformListener(self.tf_buffer, self)

# lookup transform
t = self.tf_buffer.lookup_transform(
      'base_link', 'tip_link', rclpy.time.Time(),
      timeout=rclpy.duration.Duration(seconds=0.5) )
self.x = [ t.transform.translation.x,
           t.transform.translation.y,
           t.transform.translation.z ]
```

Simulator environment - Position control

position_control (**q**)

```
# Publisher
self.arm_cmd_pub = self.create_publisher(
  Float64MultiArray,
  '/arm_position_controller/commands', 10 )

# set joint angles
msg = Float64MultiArray()
msg.data = q_target
self.arm_cmd_pub.publish(msg)
```

Simulator environment

Use the learned function to control the robot to reach a target **t**

- read **q**, **x**
- **dt = t - x**
- while **dt** > threshold
  - **dx** = normalize (**dt**) // dt outside the range of values in dataset
  - **dq** = learned_function (**x**, **q**, **dx**)
  - position_control (**q + dq**)
  - read **q**, **x**
  - **dt = t - x**

Hints

Save trained model for next steps

```
# Save the learned model
model.save('trained_model.h5')
```

Load saved model

```
# Load saved model
model = tf.keras.models.load_model('trained_model.h5')
```

Hints

Joint limits

```
joint1          -pi, +pi
joint2,3,...    -1.8, 1.8
```

Small displacements

```
| dq_i | < 10 deg

sum_i |dq_i| > 10 deg
```

Hints

Velocity of the joints to determine when the arm reached its target position

```
def joint_state_callback(self, msg):
  arm_indices = [i for i, name in enumerate(msg.name)
                        if 'arm_joint' in name]
  self.q = [ msg.position[arm_indices[i]]
                         for i in range(self.njoints) ]
  self.q_dot = [ msg.velocity[arm_indices[i]]
                         for i in range(self.njoints) ]
```

Hints

Problems:

- small absolute values of **dx** and **dq** => MSE and MAE are low
- dataset is small

Possible Solutions:

- Consider normalization/scaling
- Use R^2 score
- Consider impact of random seed on performance (compute mean and standard deviation of multiple runs with different random seeds)