Learning for Adaptive and Reactive Robot Control
Instructions for Practical 3

Professor: Aude Billard
Assistants: Harshit Khurana,
Lukas Huber and Yang Liu
Contacts:
aude.billard@epfl.ch, harshit.khurana@epfl.ch,
lukas.huber@epfl.ch, yang.liuu@epfl.ch

Spring Semester 2022

Introduction

This practical takes place in the LASA robot room ME A3 455 and REHAssist room MED 3 1015, using a Franka Emika Panda robot with 7 degrees of freedom. In this practical you will teach a task to a Panda robot using kinesthetic demonstrations and the Dynamical Systems formulation taught throughout the class. You will then control the robot using the leDS.

The software structure for the practical is as follows: we are using ROS2, with a node in MATLAB sending cartesian velocity commands computed from a dynamical system, and a C++ node converting these commands into joint torques for the robot. You will work mainly with the MATLAB part of the application, but you can also use the terminal to access ROS topics if needed.

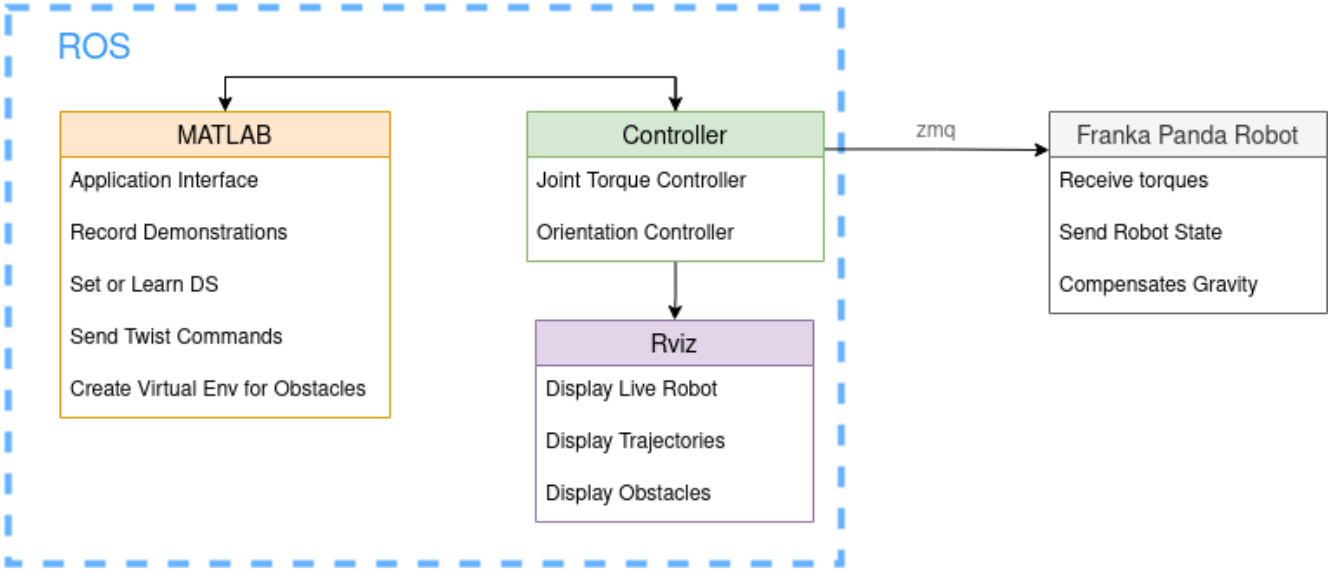

Figure 1: Software Structure

The robot is controlled by sending joint torques commands. Our controller tracks the desired twist (linear and angular velocity) by feedforward linearization and a correction on twist error. We also control the null space of the robot to stabilize the posture around the default configuration $\mathbf{q}_0$:

$$\boldsymbol{\tau} = \boldsymbol{\tau}_f + \mathbf{G}(\mathbf{q}) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{D}\dot{\tilde{\mathbf{q}}} + \boldsymbol{\tau}_0 \qquad \text{with} \quad \boldsymbol{\tau}_0 = k_0 * \mathbf{N}^{\#} * (\mathbf{q} - \mathbf{q}_0) \tag{1}$$

with the feedforward terms being $\boldsymbol{\tau}_f$ the estimated joint torques frictions, $\mathbf{G}(\mathbf{q})$ the gravity torques, and $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$ the Coriolis torques.

Safety Protocol

- Always keep the emergency STOP button within reach and ready to use. Do not hesitate to use it for any reason. Once pressed, the lights of the robot turn white: Call an assistant to unlock the robot.
- Only touch and move the robot when the light is blue.
- If the light on the end effector starts flickering, immediately stop trying to to move the robot. Press the emergency STOP button and call an assistant.
- Whenever you are changing parameters, make sure to first press **"S"** to stop sending commands to the robot.

Application Interface

During this practical, you will be controlling everything using a MATLAB GUI which is created when launching the `practical3_main.m` script. With this, you can record demonstrations, learn a DS using those demonstrations and send commands from your DS to the robot, among other things. Important notes :

- To use the GUI correctly you must first select the interface window, called Figure 1. Make sure your last mouse click was inside this window before using the keyboard.
- Matlab will display information in the command window whenever a new action begins, use this to verify that your are using the GUI correctly.
- You only need to run the main script once, as all functions can be called directly from the command window and code can be edited without needing to close the app. Available functions and properties are listed at the end of the `practical3_main.m` script.

1 Part 1: Learning from Kinesthetic Demonstrations

TASK 1 Program a linear DS in Matlab to reach a position in task space. Compare your results with what you achieved in simulation during the semester.

1. Press "**S**" in the MATLAB window to be sure the robot position controller is stopped. Move the robot by hand in the workspace. Which degrees of freedom are controlled, and which ones are free to move ?
2. Move the robot end effector in the workspace to get a sense of its boundary. Move slowly the robot to the elbow joint limits. It should automatically stop and blink blue. Call an assistant to unlock it.

3. Move the end effector to a target position. In the terminal, you can type `ros2 run matlab_bridge print_robot_state`.
   This runs a small script to display the 3D end effector position in the terminal.
4. In MATLAB's command window, call the function:
   `myHub.myDS.setLinearDS(attractor)` with the desired attractor as a 3D position vector to program your linear DS.
5. Press the key "**P**" (Play) to start the position controller. The robot should now follow your DS to reach the specified target. Answer the following questions:
   - What steady-state error does the robot achieve with this controller? Where does this error come from?
   - How compliant is the robot when you perturb it? Can it still reach the target? What is the tradeoff between compliance and tracking precision?

Whenever you are changing parameters, make sure to first press **"S"** to stop sending commands to the robot.

TASK 2 Record pick-and-place trajectories so that the robot end effector reaches the inside of the basket from the top.
You can record a trajectory by pressing the key "**R**" (record) when starting, then "**S**" (stop) when you finish you demonstration.

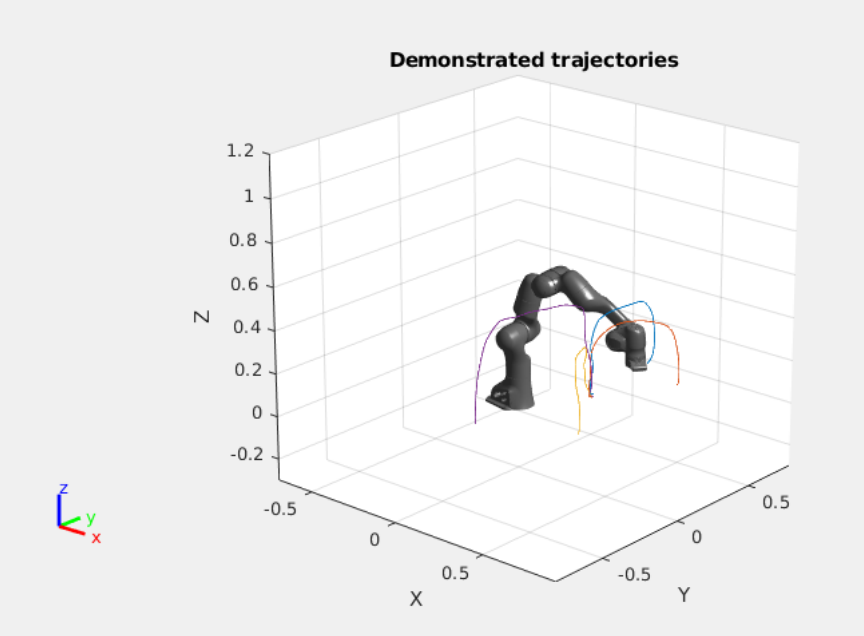

Figure 2: Examples of pick-and-place trajectories

1. Record two trajectories starting from close initial positions. When you're done, you can press the "**L**" key (Learn) to start learning with SEDS. If you need to modify SEDS parameters, you can do so in the function `learnSEDS()` of the file `DS.m`, and press again the "**L**" key to learn again until you're satisfied with the results.
   - Does SEDS fits well your demonstration?
   - Does it generalizes well on the whole workspace? What would happen if your start from another starting position?

2. Press the "**P**" key to start following your DS and observe the result. In Rviz, you can see in red the open loop trajectory from your starting point and in green the robot's actual path. Do not hesitate to press the emergency stop button before the robot hits anything.
3. Without deleting your previous dataset, record new trajectories starting from different initial conditions to span the whole workspace (see figure 2). Change the learning algorithm by typing `myHub.myDS.algoName = 'LPVDS'`, then start the learning again. Similarly, you can change the LPVDS parameters in the function `learnLPVDS()` in the file `DS.m`.
4. Press the "**P**" key to start following your DS. Answer the following questions:
   - How well can the robot reproduce your demonstrations? What is the main challenge?
   - How many trajectories are needed for generalization? What other factor affects generalization?
   - How does the quality of your DS affect the tracking performance? What could happen if the motion is not generalized over the whole workspace?

2 Part 2: Implementing Obstacle Avoidance

TASK 1 Learn a DS to follow the red tape path on the tablers and reach the target without hitting the obstacle.

1. Use the **"X"** key to clear your previous dataset. Record a few trajectories starting at similar position and following the red path. Don't approach the obstacle and the table too much to reduce the chances of collision.
2. Learn the motion using LPVDS and the Physically Consistent Non-Parametric Initialization (`est_options.type = 0`). The optimal learning parameters should already be set.
3. Start the controller to see how well it tracks your DS. Can the robot follow the path well and avoid all the obstacles at the same time?

TASK 2 Add obstacles and modulate your DS with obstacle avoidance methods

1. Add an ellipsoid in the workspace to represent your obstacle by using `myHub.addEllipsoid(position, axes)` with `axes` the 3D vector of the ellipsoid semi-axes, and `position` the 3D vector of it's center position. The [0,0,0] position is at the base of the robot. You can call `myHub.updateObstacle(obstacleNumber, newPosition, newDimensions, newRho)` if you need to change the obstacle's position, dimension, or sensitivity (rho value). You can use the terminal to match precisely the position in the application and in your workspace.

```
% Create a sphere of radius 10cm with default rho value of 1.
myHub.addEllipsoid([0; 0;0], [0.1, 0.1, 0.1]);

% Move the sphere (obstacle 1) up by 50cm
myHub.updateObstacle(1, [0; 0; 0.5], [0.1; 0.1; 0.1])
```

2. In case the modulation is not activated in your DS, you should call the function `myHub.myDS.activateModulation` in MATLAB's command window. Then, press **"P"** to command the robot with your modulated DS. Does it now avoid the obstacle? What about the table?
3. You can now add table avoidance to your DS by adding an horizontal plane in the workspace to represent the table. You can use the function `myHub.addPlane(position, normal)` where normal is the vector normal to the plane. Play the DS again. Can the robot better avoid the obstacles?
   The Rviz display is adapted so the position parameter is the center of the top surface of the object, to better view and represent an infinite plane.
4. Change the sensitivity `rho` of the obstacles to see their influence on the initial trajectory. What happens if the sensitivity is to large?

```
% Set the rho value of the first obstacle to 0.1
myDS.myWorld.listOfObstacles(1).rho = 0.1
```

References

[1] Aude Billard, Sina Mirrazavi, and Nadia Figueroa. *Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach*. MIT press, 2022.