TÉCNICO LISBOA

Robotics

Winter 2021

Departamento de Engenharia Electrotécnica e de Computadores

1st lab assignment - v1.2 - Last updated 19/12/2021

Introduction to Serial Manipulators

(To be handed – by e-mail – by 30 December 2021, 23:59:59)

João S. Sequeira

1 Syllabus

The aim of this lab assignment is to demonstrate the importance of kinematic models in the execution of a typical task by the serial manipulator with 6 rotational degrees-of-freedom (dof) in figure 1.

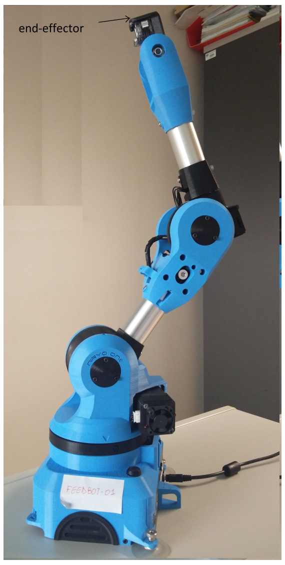

Figure 1: The 6-dof serial manipulator

From a mathematical perspective, this serial manipulator maps the set of 6 joint angles, $\theta_1, \ldots, \theta_6$, into the gripper's 3 position coordinates, x, y, z, and 3 orientation coordinates, α, β, γ. This map is called **direct kinematics** or **forward kinematics**.

The inverse of the **direct kinematics** is called **inverse kinematics** or **backwards kinematics**. This map returns sets of joint angles that correspond to a given position and orientation input.

All the geometric information, necessary for the development of kinematics models is contained in the physical dimensions shown in figure 2.

The explicit computation of both the direct and inverse kinematics is a topic that will be covered in the theory classes. However, it must be emphasized that it relies, exclusively, in the careful definition of reference frames (i.e., coordinate systems where to measure the angles, i.e., the dof of the manipulator) and on the computation of homogeneous transformations between consecutive reference frames. Therefore, this is a topic that has already been discussed in previous Linear Algebra courses.

The basic software available for the Niryo One simplifies, has available function "get_pose" that implements the direct kinematics.

2 Tilling task

The work to be developed in the lab consists in having the robot picking a set of rectangular pieces of light material (plastic) from a pre-defined source area and place them in a different pre-defined goal area (much like covering an area with tiles).

The goal is to lay down the pieces such that the target area is completely covered. The order by which

Degrees of Freedom	6
Weight	3,3 kg
Reach	440 mm

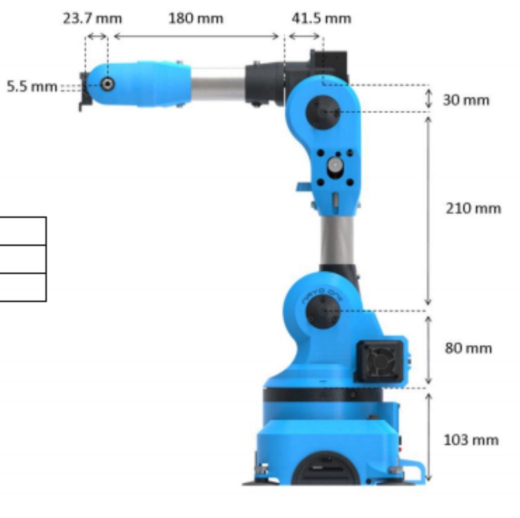

Figure 2: Niryo One physical dimensions. Source: Niryo One Mechanical Specifications

the pieces are placed is irrelevant, i.e., each group can define its own strategy to pick and place.

At the source area, the pieces will be stacked on top of one another. The stacking order will be fixed and known a priori.

The pieces are covered in Velcro so that they can be grabbed by the robot gripper, which has also a Velcro strip.

Once the piece is placed in the target area, a lab assistant will hold it in place before the arm is moved back (and the Velcro from the gripper disconnects from that in the piece). This avoids having to install (and deal with) a complex gripper.

The work to be developed is, simply, a program that, for each piece in the starting area, (i) computes a path for the robot to move from its current position to the position of the piece, with the adequate orientation, (ii) moves along this path and picks the piece, (iii) computes a path between the current position and the point in the target area where the piece will be deployed and moves along this path, (iv) drops the piece and moves back to a suitable position.

For the purpose of the work, a path is an array of points and orientations.

The paths to be computed are completely free. In a real scenario, these paths may be obtained from some optimization problem or a priori knowledge about uncertainties in the kinematics. This is a point where Engineering creativity can be assessed (and bonus points obtained).

3 Connecting to the robot

The robot can be assessed remotely via a specific Wi-Fi network and SSH interface. The development of software uses a ROS+Python^{TM} environment.

The website of the robot (www.niryo.org) and GitHub resources (for example, github.com/NiryoRobotics/niryo_one_ros) provide multiple resources that can be used by the students to complete the lab assignment.

To connect to the robot Wi-Fi network use the usual (computer dependent) apps.

To login in the robot use the command line instruction

```
ssh -XC 10.10.10.10 -l niryo
```

With the password “robotics”

Students are not required to know about ROS. A library of Python^{TM} functions is available (see the Niryo One documentation). These allow the any Python^{TM} script to access the joint values and to move the joints to any admissible value – it is of critical importance to understand the limits of each joint prior developing any software to move the joints.

However, to operate the robot ROS has to be running and some initialization is necessary. This can be made with the following instructions (which can be launched in separate windows/terminals),

```
roscore
```

Under unspecified circunstances roscore mail fail to launch (you get a number of error messages). In that case do,

```
killall -9 roscore

killall -9 rosmaster
```

and relaunch roscore. Also launch the following node,

```
roslaunch niryo_one_bringup rpi_setup.launch
```

Once this script is running the robot is ready to run any software developed for this lab assignment.

In a new terminal, make

```
cd catkin_ws/src/js
```

and “look” at the Python scripts there. The script “test_py_api.pi” exemplifies the usage of functions to start and initialize the robot, and how to get the angles of the joints and how to move them.

Try running the script with

```
python test_py_api.py
```

and you’ll see the robot calibrating itself and making a small movement in the first three joints.

The Appendix illustrates a Python^{TM} script exemplifying the use of functions to get the current joint values and to move the joints by a relative value.

4 Development alternatives

The tilling task requires that a path for the end-effector, between the initial and final places of each tile is computed. The programming resources for the Niryo One are easily available online, so the students are encouraged to find the alternative that suits them the best (keeping in mind that the robots are shared resources and no changes to the original software installed can be made).

The students are required to develop a Python^{TM} script that either

1. Runs inside the robot and makes it execute the task, or
2. Runs outside the robot and generates a time indexed array of joint values corresponding to positions of the path the end-effector has to move through in order to execute the task.

In the first alternative, the students are required to connect to the robot and use the normal tools available in Ubuntu installed onboard the robot to develop their Python^{TM} programn (following the procedure indicated in section 3). No additional Python^{TM} packages should be installed onboard the robot (the original software of the robot is NOT to be modified in any form).

In the second case, the Python^{TM} program is to be developed away from the robot, using whatever tools are needed, e.g., in laptops proprietary of the students. The output must be an ASCII file, with a timed array of joint values (each set of joint values represents a point of a path the robot has to follow to execute the task, that has to be reached at the time specified). This file will then be downloaded to the robot, e.g., using the a USB pen and read by a specific Python^{TM} script (to be developed by a third party – not by the students) that will make the robot move, in sequence, through the positions in the file at the indicated times.

The two alternatives above are, by no means, the only possible approaches. For example, students with a Matlab Robotics toolbox may use the Matlab ROS bridge to connect to the robot and take profit of the Matlab environment (non Python)to develop

5 Expected outcome

- Python^{TM} scripts implementing the tilling task.
- Code shall be handed, by e-mail, ready for automated demonstration, no later than 30 December 2021, 23:59:59.
- A report, in pdf format, detailing all the steps and assumptions taken, and how to operate the software developed, must be handed 1 week later (max). The report must have no more than 8 A4 pages (strict).

A Python^{TM} script example

```
#!/usr/bin/bash

from niryo_one_python.api.niryo_one_api import *
import rospy
import time

rospy.init("Niryo One node init")

n = NiryoOne()
n.calibrate()
c = 0

try:
    print("test counting ".format(c))
    c = c + 1

    joints = n.get_joints()
    print(joints)

    joints = joints + [0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
    n.move_joints(joints)

except niryo_one_exception as e:
    print(e)
```

A Guidance notes

The goal of the lab is the development of a Python script to make the Niryo to move 4 plastic blocks from an initial position to a target area.

At the target area the blocks must be organized as if they were tiles, i.e., they must as close as possible, and covering completely the target area.

Given the frailty of the real robots, it is a sensible approach to work with the available simulator. This, however, requires groups to carefully select and model the effects caused by the uncertainties of the real robot and of the plastic blocks.

In the following, a set of guidance notes aim at highlighting the key points that should be taken into consideration.

- The work can be developed exclusively for the Niryo One robot and the specific task of moving the blocks, i.e., any specific feature of the Niryo and of the task that are used to simplify the work are admissible.
- The software developed for the simulator should use only functions that can also be used with the real robot.
- The approaching (and grabbing) of the plastic blocks must be made very carefully (otherwise, – in the real scenario – the whole pile of blocks would fall down) – this has been exemplified in the lab.

  This means that the gripper must move at a very slow speed when approaching a block. Similarly when approaching the ground at the target area.

  The software must show, clearly, that the approach speed for grabbing and placement tasks is small enough. This is valid for software developed for both the simulator AND the real robot.

  Ideally, these approach velocities should be an input parameter to the software.
- In the real robot, the grabbing force would result from the overlapping area of the two Velcro surfaces.

  In the simulation, it is up to each group to decide whether or not to also simulate this feature. The simple alternative is to assume a ON/OFF gripper, without any grabbing force control.
- The simulator does not represent the plastic blocks. Its purpose is, uniquely, to display the motion of the robot.

  In general, it should be possible to represent the plastic blocks (as the graphics are RViz/ROS based). This however would require some time and it is NOT the focus of the work. Therefore, the suggestion is to use alternative simple plots (e.g., use Python's matlibplot or similar, to show where the blocks are – showing at least the initial and final positions)
- The real robot exhibits uncertainties in the position of each of the joints (as in any real robot), due, for instance, to gear backlash and wear, and loose mechanical couplings. These uncertainties have a major influence in the behavior of the robot.

  Simulating these uncertainties is, is general, a complex process. However, an approximation can be constructed assuming that each joint value follows some distribution (e.g., Normal with mean at the nominal value of the joint and variance a user defined parameter) and studying how these joint uncertainties are mapped by the direct kinematics.
- The report show discuss, at least, the following points

- The direct kinematics for the Niryo One (including the chosen frames).
- A block diagram with the architecture corresponding to the work developed (refer to the lecture on architectures for inspiration).
- An explanation on how the uncertainties where tackled.
- How scalable is the work developed, e.g., could it be applied to a real tilling problem.

B Evaluation guidelines

- Groups can choose if they want their work to be evaluate through the simulator or through the real robot (in case it becomes available).

  The two ways will be graded similarly, i.e., accounting for the performance of the robot.

- It is strongly suggested that the outcomes of each group include a file with all the points/commands send to the robot (independently of that being the simulator or the real robot). This allows debugging in case of software installation/running problems.

  This file must be readable (ASCII) with a basic text editor. Each line must include a single command AND must include the time at which the command was sent to the robot.

  Note that this is the same file already referred as the simplest solution alternative for the lab. Given the changing conditions, the role of this file is now amplified.