FINAL PROJECT
ROBOTICS

UNIVERSITY OF PISA

DEPARTMENT OF INFORMATION ENGINEERING

Robot manipulator

Authors:
Leonardo Lai

Email:
l.lai@santannapisa.it

Date: November 5, 2020

Contents

1 Introduction

This project simulates a robotic arm performing a simple pick-and-place task.
The robot can be instructed to move to a specific position in the space, pick an object there and re-position it elsewhere with precision.
The demo shows how the robot is able to pick a set of bricks (cube) and rearrange them according to a pre-configured pattern.
For simplicity, this project does not consider the design of the gripper (end-effector) and its mechanical interaction with the objects.

1.1 Architecture overview

The robotic arm is simulated in *MATLAB* with the *Robotics Toolbox* (by Peter Corke). The low-level controller, enabling the robot to move from one position to another, is implemented directly in MATLAB. The high-level controller, a.k.a. *planner*, determines the initial and final positions of the objects, and commands the robot to move accordingly. It is implemented as a state machine in *Python* leveraging *ROS (Robotic Operating System)*. Note that the planner can potentially run on a different machine with respect to the low-level controller. In MATLAB, the *ROS Toolbox* acts as a glue layer between ROS and the other simulation scripts.

2 Manipulator

The robot is a crane-like arm with six revolute joints, the last three forming a spherical joint to achieve 6-DOF. Figure 1 shows its geometrical structure.

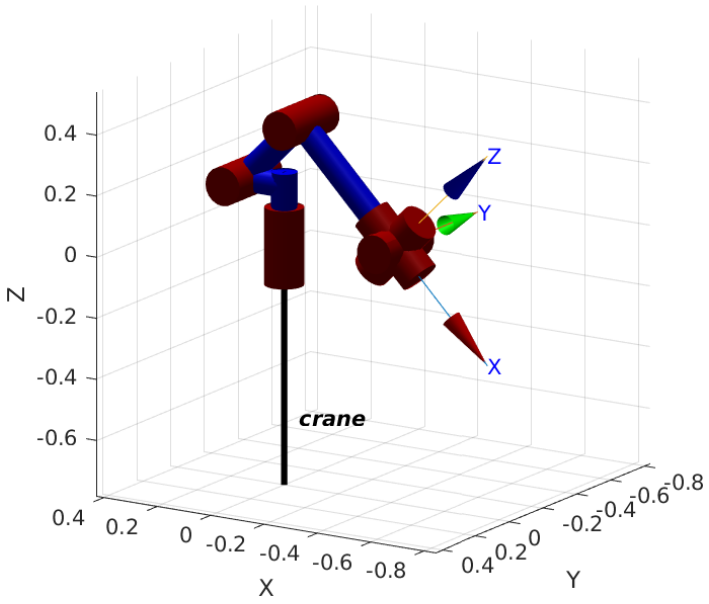


Figure 1: Robotic arm

2.1 Joints

According to the standard DH convention (Denavit-Hartenberg), the joints can be represented as in the following table. Angles are in radians, lengths in meters.

Joint	θ	**d**	**a**	α
1	$q1$	0.2	-0.15	$\pi/2$
2	$q2$	0	0.3	0
3	$q3$	0	0.5	0
4	$q4$	0	0	$-\pi/2$
5	$q5$	0	0	$\pi/2$
6	$q6$	0	0	0

Table 1: DH table

The second and third joints are constrained in a limited range, as shown in Fig. 2.

Joint	$\theta_{\mathbf{min}}$	$\theta_{\mathbf{max}}$
1	-	-
2	$\pi/6$	$\pi/2$
3	$-2\pi/3$	$-\pi/8$
4	-	-
5	-	-
6	-	-

Table 2: Joint limits

The singularity at joint 3 is avoided because $\theta_3 = 0$ is out of the feasible range.

2.2 Links

With a rough approximation, links have been designed as solid steel ($\rho = 8\,\mathrm{kg}/m^2$) cylindrical beams of 6 cm diameter. Masses and inertia were estimated accordingly.

Link	**Length** (m)	**Mass** (kg)	**Inertia** (kgm^2)
1	0.25	8	[0.03, 0.03, 0.006]
2	0.3	9	[0.06, 0.06, 0.006]
3	0.5	15	[0.16, 0.16, 0.006]
4	0	0.3	[0.001, 0.001, 0.001]
5	0	0.3	[0.001, 0.001, 0.001]
6	0	0.3	[0.001, 0.001, 0.001]

Table 3: Link parameters

3 Controller

The joint configuration at time t is indicated by $q(t) \in \mathcal{C}$. The reference configuration, i.e. the desired one, is indicated by q^*. The error is defined using the generalized coordinates as: $e = q^* - q$.
The controller is time-discrete with a frequency $f = 50\,\text{Hz}$ (i.e. $\delta t = 20\,\text{ms}$), which is empirically much smaller than the rise time (approximately $1\,\text{s}$).

The system is non-linear, as the inertia and forces strongly depend on the current configuration of the robot. In general, the torque can be written as:

$$\tau = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q)$$

where M is the inertia matrix, C is the Coriolis vector and G the gravity vector. Nonetheless, having a complete description of the system, the dynamics can be computed and embedded in the controller to cancel the nonlinear terms, so to exploit a linear controller like a PID to follow a trajectory. The *computed-torque* controller outputs a torque equal to:

$$\tau = M(q)[\Phi] + C(q, \dot{q})\dot{q} + G(q)$$

where Φ is the output of a linear controller. In this project, I use a PID controller:

$$\Phi = K_p e + K_v \dot{e} + K_i \int e$$

, although the integrative term coefficients are left to zero (no steady state error since the input is step-like and the gravity is already compensated).
The controller parameters, reported in Table 4, are obtained by trial and error.

Joint	Kp	Kv
1	15	0.15
2	15	0.15
3	8	0.1
4	10	0.1
5	10	0.1
6	10	0.1

Table 4: Controller parameters

Saturation was applied downstream to the PID to mitigate heavy peaks in the output torque, that could be problematic in practice for the joint motors (not modeled in this project). Also, the angle values for joint 1 are dynamically adjusted (modulo 2π) to ensure that the robot always spins in the direction (clockwise/counterclockwise) that is closest to the target.
The desired position is considered 'reached' when the (Euclidean) distance from it is below a threshold (in practice, $1\,\text{cm}$ for the placing task).
When the arm is carrying an object, the payload mass is set to $1\,\text{kg}$.

4 State Machine

From a functional point of view, the task of choosing, picking and placing a set of objects can be represented and decomposed by means of a state machine, whose states map to elementary actions for the low-level controller (e.g. move from one point to another). See Fig. 3 to have an idea of the states and transitions (outcomes).

4.1 States

Four different states are defined:

- SELECT: the SM schedules the next object to pick, if any.
- PICK: the robot moves towards the object to pick.
- PLACE: the robot, holding the item, moves towards the destination to lay it.
- REST: the robot returns to its resting position.

PICK and PLACE are grouped in the PICKANDPLACE superstate, thus making the state machine hierarchical. The reason for such aggregation is to handle potential failures in a uniform fashion.

In practice, SELECT picks the next object from a static predefined list and returns two vectors: the first is the initial pose, where the object spawns, the second is the desired pose, where the robot should move it.
The former information is needed by PICK, the latter is used by PLACE. Both PICK and PLACE are ROS action clients. Actions in ROS are a communication mechanism, that allows nodes to ask other nodes to asynchronously perform some function, and later retrieve the result by means of a callback. Actions specify goals (things to do) and can either succeed, abort or be interrupted (preempted). Finally, the REST state (which is also a simple action server) tells the robot to return to its default pose.

4.2 Behavior

The SM starts in the SELECT state; after choosing a block, it transitions into PICK and remains in this state until the robot end effector reaches the target pose. Afterwards, the SM moves to the PLACE state and, when the object is successfully placed, return back to SELECT. When there are no more blocks to place, the state machine moves to REST and commands the manipulator to return to the default position.

The state machine is implemented in ROS using *smach*.
Overall, the ROS architecture is shown in Fig. 2. The state machine (action clients) communicates with the action server (ideally running close to the robot), which in turn messages with MATLAB scripts via topics. Note that MATLAB Ros Toolbox does not implement action servers, so an extra node is needed acting as a bridge.

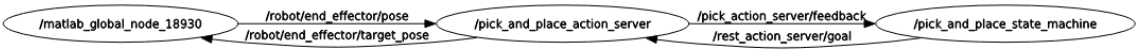


Figure 2: ROS architecture

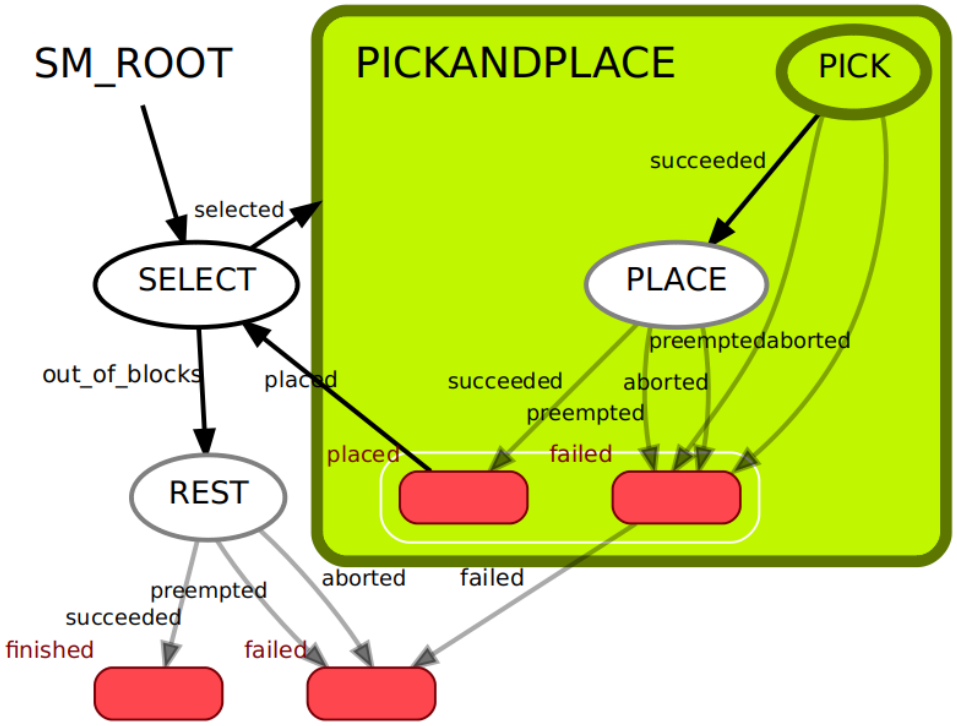


Figure 3: Pick and place task state machine

5 Results

After the simulation, the situation looks like in Fig. 4, with the robot resting in the default configuration and the objects arranged as requested. The robot completed the task in approximately 20 seconds. Figure 5 and 6 show the angular position and the torque applied to the three main joints, respectively. Thanks to the chosen controller parameters, all the three joints reach their desired angle approximately at the same time; in other words, no joint reacts significantly slower or faster than the others, which is good because the slowest would bottleneck the task.

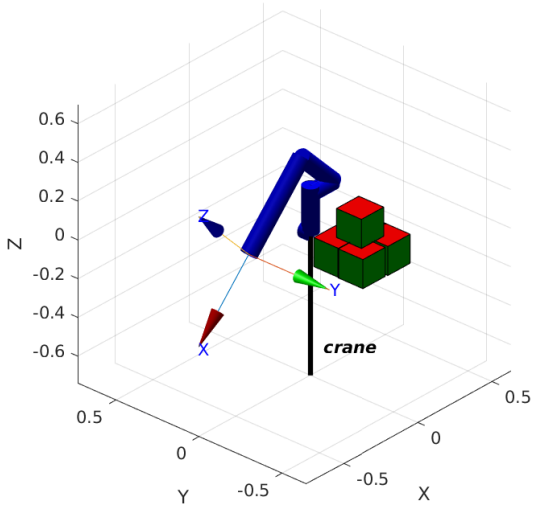
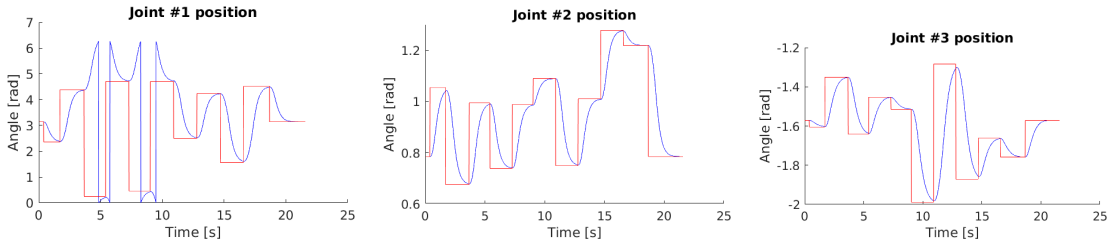
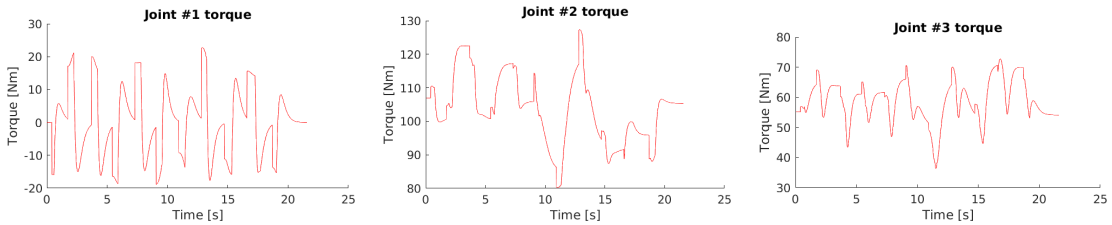


Figure 4: Environment after finishing the task



(a) Joint 1 (b) Joint 2 (c) Joint 3

Figure 5: Joint angles (blue: actual value, red: reference)



(a) Joint 1 (b) Joint 2 (c) Joint 3

Figure 6: Torque

6 Limitations

- The geometrical and physical properties of the robot are extremely simplified.
- Joint motor dynamics is not considered in the simulation.
- Frictions are not considered too.
- The controller relies on the accurate knowledge of the dynamics parameters, which is a non-trivial requirement in a realistic environment.
- Failure handling is only partially implemented (stub) in the state machine.