TUM Institute for Cognitive Systems (ICS)
Multi-Sensory Based Robot Dynamic Manipulation

Session 9: Final Project

Dr. Emmanuel Dean, Simon Armleder, Maximilian Bader

01.2021

1 Introduction

In this session, we will explain the tasks of the final project. The project will be conducted in a virtual scenario. We provide you with a simulator of the UR10 robot in ROS that behaves in the same way as the real robot, see Figure 1.1. Your goal is to develop controllers for this virtual robot.
This time, in contrast with the tutorials, your task is to use the full six degrees of freedom and control the Cartesian position and orientation of the robot's end-effector. The dynamic parameters of the robot are unknown and need to be estimated/handled by the controller.

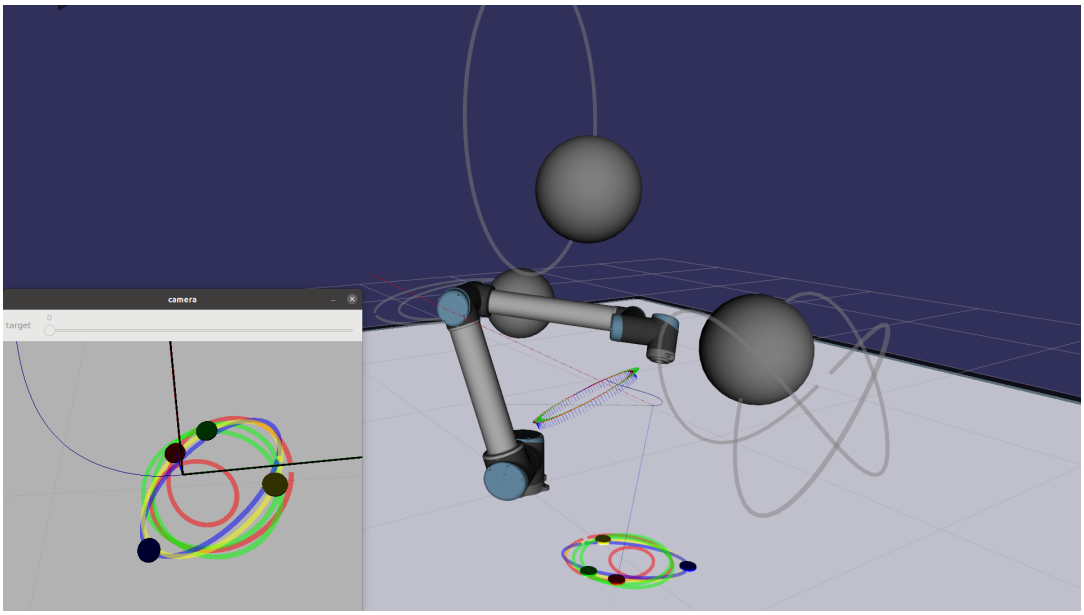

Figure 1.1: **Simulator:** UR10 robot, spherical Obstacles and targets on the ground (small circles of different color).

2 Tasks

2.1 Adaptive Motion Control

The first task is to obtain the full kinematic and dynamic models of the robot. This requires you to create the complete six DOF DH-table and port the robot model from matlab to c++ functions.
Think especially about which representation you want you to use for orientations and how this affects the mappings. You are free to choose from Euler angles, Quaternions, Axis-Angle, etc.

Once all model equations are in place, the next step is to test the model. For this, you will implement a Cartesian controller that moves the robot on a simple test trajectory (see Figure 2.1).

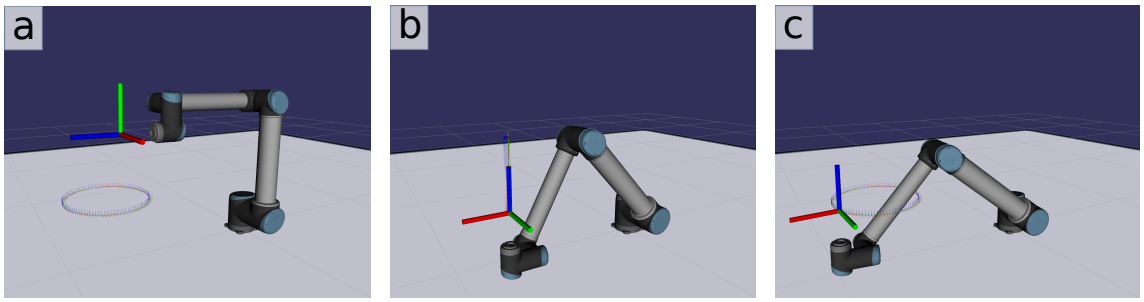

Figure 2.1: **Motion Control:** a) Robot moves from the initial position to a non-singular position. b) Robot moves to a target cartesian position, changing the orientation of the end-effector. c) Robot follows a desired Cartesian position, keeping the orientation constant.

This test should evaluate if the Cartesian controller can follow desired trajectories for the position of the end-effector and for its orientation. An example of such test is:

- First, move out of the initial singularity, using a joint space controller.
- Then, lower the end-effector position and rotate such that its *z-axis* points upwards.
- Finally, move on a circular trajectory, keep the end-effector *x-axis* pointing forward.

This test is an important step to verify whether your robot models and regressor are implemented correctly. The concrete goals are:

- Obtain and verify the 6 Dof robot models (5P)
- Obtain and verify the 6 Dof robot regressor (10P)
- Implement and test a motion controller in Cartesian space, for position, orientation, and pose controls. (15P)

2.2 Gazing + Motion Control + Obstacle Collision Avoidance

Now that the robot model and motion control work, it's time to tackle the main task.

The scenario is depicted in Figure 1.1: The robot is equipped with a camera that is mounted on its end-effector. On the ground there are multiple moving targets. A Gaze controller makes sure

that one of the visual targets remains in the camera's field of view. This controller adjusts the orientation of the end-effector such that the camera always points towards the current target. At the same time, the end-effector should move along a circular trajectory. Additionally, there are moving obstacles (modeled as flying spheres) in the environment that the robot has to avoid.

This leads to three tasks that need to be handled:

Gazing
The highest priority task is to keep the end-effector's orientation pointing to the direction of the current visual target. This is a three DOF orientation task that has to be fulfilled all the time.

Traking and obstacle collision avoidance
The second and third tasks are the trajectory tracking and obstacle collision avoidance (OCA). Since the robot has six DOF and three are already used by the previous orientation task, there are only three DOFs left. This means that we cannot fulfill tracking and OCA simultaneously. Instead, the trajectory tracking should be suspended if an obstacle gets too close to the robot. The robot then avoids a collision and continuous with the tracking once the obstacle has moved to a save distance.

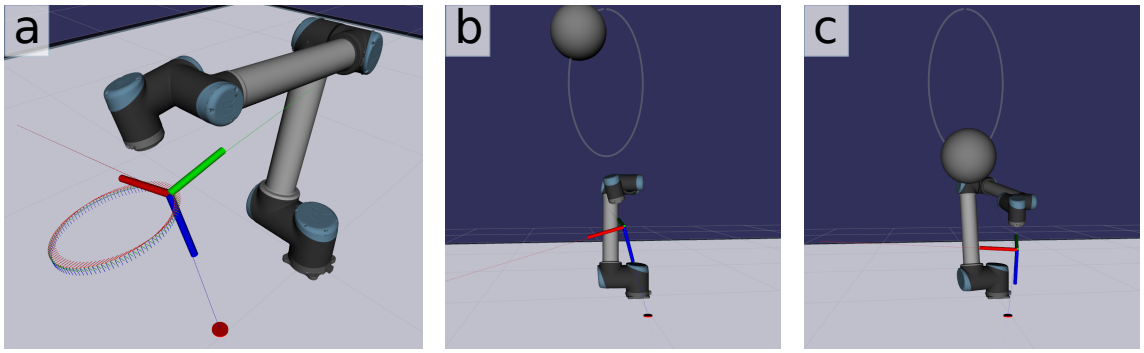

Figure 2.2: **Robot Tasks:** First: Gazing + Tracking. Second, Third: Gazing + Obstacle collision avoidance with single obstacle.

These tasks requires you to think about the following problems:

- How to obtain the rotation matrix that points the end-effector towards the desired target?
- How to deactivate the the trajectory tracking when an obstacle gets close to the robot?
- How to avoid collisions with multiple obstacles?
- How to smoothly converge back to the trajectory after the obstacle is at a save distance?
- How to make sure that highest priority tasks is executed all the time?

Its best to solve these problems in small steps and only increase the complexity if the previous step works (see e.g. Figure 2.2). The suggested steps to solve this task are defined as follows:

1. Move the robot over the visual targets.

2. Implement the Gaze controller towards a fixed target and no end-effector trajectory. (5P)
3. Implement the Gaze controller towards a moving target and no end-effector trajectory. (5P)
4. Implement the Gaze controller towards a moving target and a circular end-effector trajectory. (10P)
5. Implement the obstacle collision avoidance for a fixed end-effector pose and a single obstacle. (5P)
6. Implement the obstacle collision avoidance for a circular end-effector trajectory and a single obstacle. (5P)
7. Implement the obstacle collision avoidance for a circular end-effector trajectory and multiple (at least two) obstacles. (10P)
8. Finally, combine the Gaze controller with the end-effector trajectory and obstacle collision avoidance. (15P)

3 Software

3.1 Installation

The simulator requires Ubuntu 18.04 with ROS-Melodic. First you need to install the following packages:

- `sudo dpkg -i libtum-ics-tools-*`
- `sudo dpkg -i tum-ics-lib*`
- `sudo dpkg -i libtumtools-*`
- `sudo dpkg -i ros-melodic-tum-ics-*`
- `sudo apt-get install ros-melodic-map-server`

Then, you create a new workspace and copy the packages of the template into the src folder. Finally you build with:

- `catkin_make`

3.2 Running

An overview of the active nodes is shown in Figure 3.1. You only have to modify the controller node and subscribe to the right topics, to get feedback about targets and obstacles.

To run the code you need to launch the simulator first

- `roslaunch tum_ics_ur10_bringup bringUR10-simulator.launch`

This will brinup the UR10, a `camera node` and an `object_server node`. The `object_server node` will provide you with the services:

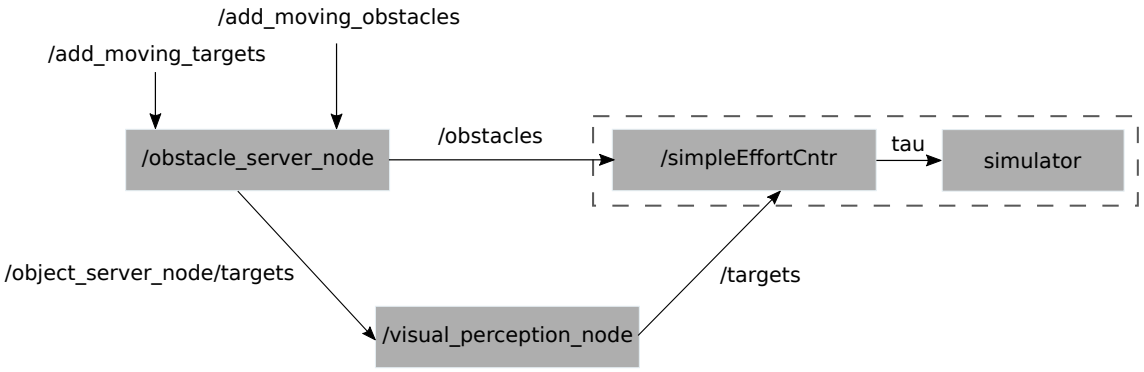

Figure 3.1: **Nodes:** The controller receives feedback about the pose of the visual target from the `/visual_perception_node`. Poses of obstacles are published by the `/obstacle_server_node`.

- `/add_moving_obstacles`
- `/add_moving_targets`

Which add moving obstacles or moving targets to the scene. Also, the two new topics:

- `/target`
- `/obstacles`

will appear, which provide the current pose of targets and obstacles. You need to use the information in this topics as feedback signal for your controllers.

Next, you can run the default controller, which simply executes a joint spline.

- `roslaunch tum_ics_ur10_controller_tutorial testSimpleEffortCtrl.launch`

The gains and goal are loaded form the yaml file in:

- `/tum_ics_ur10_controller_tutorial/launch/configs/simpleEffortCtrl.yaml`

3.3 Delivery and Time Schedule

3.4 Time Schedule

- **09.02.21 - 25.02.21**:
  - Obtain and verify 6 DOF robot model and regressor
  - Run Test trajectories with Cartesian controller
  - Finish the last two homeworks
- **25.02.21 - 18.03.21**:
  - Work on the main task (Gaze control, Tracking and OCA)

3.5 Delivery

- **25.02.21**:
  - The last two homeworks
  - The 6 DOF kinematic model and regressor.
    Note this is an optional delivery. However, we will add a Bonus to your project grade (+0.5) if you deliver the model before this date.
- **18.03.21**:
  - Presentation (3-5 min, Approach, Block Diagram, Plots/Video) (10P)
  - Short Report describing the Approach/Results (5P)
  - Screen Recording of the final controller
  - Code

4 Notes

The following resources might be interesting:

- Eigen
- ROS Plotting Program
- ROS visualization marker
- Orientations
- Redundancy
- Robotics Lecture Notes