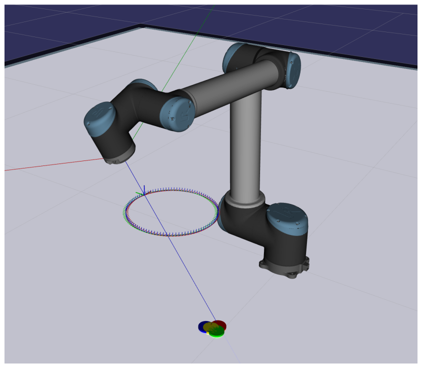
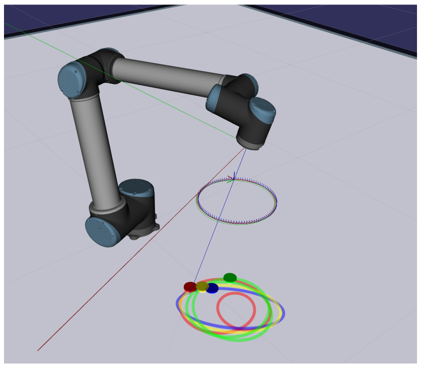
Multi-sensory Based Robot Dynamic Manipulation – Final Project Report

Marco Oliva

18.03.2021

1 Approach

1.1 Overview

The following sections describe the approaches that have been used to complete the tasks. Three control laws are applied: a joint space controller for the initial motion from the singular position into the start position, a Cartesian space controller to follow the circular trajectory, to control the orientation towards the targets and to move along trajectories back to the circular path after being diverted due to nearby obstacles. An impedance controller is used to avoid collisions with obstacles.

1.2 State Machine

At each time step, the controller classifies the current situation into a distinct state and performs actions specific for this state. The possible states and the corresponding actions are:

1. *leaveSingularity*: The initial state. Here, the robot moves into a non-singular start position using a simple joint space control law.
2. *followTrajectory*: The robot follows the circular trajectory using a Cartesian space control law.
3. *avoidObstacles*: The robot computes repulsive forces originating from the obstacles, acting on each joint. Based on these forces, an impedance control law produces torques that move the robot away from the obstacles to avoid collisions. This state is automatically enabled when the distance of at least one obstacle to at least one joint is smaller than a specified threshold.
4. *returnToTrajectory*: The robot calculates a trajectory that moves it from the current position to the desired position on the circular trajectory. When the trajectory is completed, the state *followTrajectory* is activated.

Except for the initial joint space trajectory, these states only affect the first three degrees-of-freedom of the robot. At any time, the orientation is controlled such that the end-effector points towards the currently active target, utilizing the remaining three degrees-of-freedom.

1.3 Cartesian Control Law

The Cartesian space controller uses an adaptive PID control law in error-space formulation with dynamic model compensation. Thus, the Cartesian velocity references $\dot{x}_r \in \mathrm{R}^6$ are defined as

$$\dot{x}_r = \dot{x}_d - K_p \Delta x - K_i \int \Delta x \; d\Delta x \tag{1}$$

and transformed into joint space with

$$\dot{q}_r = J^\dagger \dot{x}_r \tag{2}$$

where $J^\dagger$ is computed depending on the current manipulability index

$$w = \sqrt{\det(JJ^T)} \; . \tag{3}$$

If $w \geq 0.1$, then $J^\dagger = J^{-1}$ is the inverse of the geometric Jacobian, otherwise a damped pseudo-inverse is used, given by

$$J^\dagger = J^T (JJ^T + \psi \mathrm{I})^{-1} \tag{4}$$

with $\psi = 0.05$. To compensate for the unknown dynamical parameters, an adaptive control law is applied that estimates the dynamics. The compensation term is defined as $Y_r \hat{\theta}$, where $Y_r = Y_r(q, \dot{q}, \dot{q}_r, \ddot{q}_r)$ is the robot regressor matrix and $\hat{\theta}$ is the current estimate of the dynamical parameters. At each time step, this estimate is updated as

$$\hat{\theta}_{t+1} = \hat{\theta}_t - Y_r^T S_q \delta t \tag{5}$$

where $S_q = \dot{q} - \dot{q}_r$ and δt is the time in seconds passed since the last update. The final control law is then given by

$$\tau = -K_d S_q + Y_r \hat{\theta} \ . \tag{6}$$

1.4 Calculation of Desired Orientations

For orientations, a quaternion representation is chosen. The desired orientation Q_d is computed by first calculating the target orientation Q_t that points the end-effector to the target. Then, depending on the angular distance $\Delta\alpha$ of the current and target orientation, the desired orientation is interpolated using *slerp*. For this, a configurable *slerpAngularErrorThreshold* is defined as a parameter. If the orientation deviation $\Delta\alpha$ is below this threshold, the target orientation is directly used as desired orientation without interpolation. Otherwise, the Q_d is interpolated over the interpolation parameter s whose trajectory is generated using a 5th-order polynomial. The duration d of the trajectory of s is computed depending on the angular error with

$$d = k_s \Delta\alpha \tag{7}$$

where k_s is a configurable parameter called *slerpDurationErrorGain*. Using this approach, the controller allows for more time when the sudden change in target orientation is larger, e.g. when the active target switches, while quickly rotating to new targets with a small angular distance.

1.5 Trajectory Generation

The custom implemented class *CartesianTrajectory* offers functions for 5th-order polynomial trajectories, as well as a circular trajectory. The circular trajectory is constantly giving the current desired tool pose, as defined in the task. The 5th-order polynomial is used to generate a trajectory that moves the end-effector (tool) towards that circular path, either after the initial joint space motion or after the robot has left the desired path in order to avoid obstacles.

To generate a trajectory back to the circular path, there is an important problem to consider. The trajectory needs to lead the robot to $p_c^*(t_f)$, the target position on the circle at the time step when the trajectory is completed. However, in the meantime the target position has moved along the circular path resulting in a deviation of the final position $p(t_f)$ and the actual target position $p_c^*(t_f)$. Using a fixed duration would fix this problem easily, however this would lead to very fast motions when the robot is far away, as well as very slow motions when the end-effector is close to the circle. For an improved solution, the following two-step procedure is applied. First, a naive estimate of a good duration Δt is calculated by using as final position the current desired position of the circular motion $p(t_f) = p_c(t_0)$. Then, the distance to that position is used to define a duration Δt, given by

$$\Delta t = k_t \|p - p_c(t_0)\|_2 \tag{8}$$

where k_t is a configurable parameter called *trajectoryDurationErrorGain* that influences the velocity of the motion along the trajectory. However, this estimated duration is suboptimal since the desired position on the circular path will have changed until the robot reaches the currently considered target position: $p_c^*(t_f) \neq p_c(t_0)$. For this reason, the naive duration estimate is only used as an approximation of a *good* duration towards to circular path. Instead of using Δt to generate a trajectory to $p_c(t_0)$, the controller computes the actual target position on the circular path $p_c^*(t + \Delta t)$. Then, an improved duration Δt^* to $p_c^*(t + \Delta t)$ is computed equivalently to (8). Consequently, a polynomial that interpolates the current position, velocity and acceleration of the end-effector to the target position, velocity and acceleration on the circular path at time $t + \Delta t^*$ is generated that smoothly converges the end-effector back to the circular motion.

1.6 Obstacle Avoidance

For obstacle avoidance, a simple impedance control law is applied, given by

$$\tau = \sum_{i}^{6} J_{l,i}^T k_{f,i} \sum_{j}^{n} f_{rep}(i,j) \tag{9}$$

where $J_{l,i} \in \mathrm{R}^{3\times 6}$ is the linear part of the Jacobian w.r.t. to joint i, $k_{f,i} \in \mathrm{R}^6$ is an element-wise gain that influences how the repulsive forces affect joint i, $f_{rep}(i,j) \in \mathrm{R}^3$ is the repulsive force acting on joint i, originating from obstacle j, an n is the number of objects that are within a distance r_o to joint i. The distance threshold is a configurable parameter. The repulsive potential $f_{rep}(i,j)$ is calculated by

$$f_{rep}(i,j) = \left(1 - \frac{d(i,j)}{r_o}\right)^2 \frac{d(i,j)}{\|d(i,j)\|_2} \tag{10}$$

where $d(i,j)$ is the distance from joint i to obstacle j and r_o is the radius of influence of the obstacle's repulsive potential.

To enable constant orientation control towards to active target even during collision avoidance, the impedance control law is combined with the Cartesian controller. As such, a Cartesian control law computes torques for orientation control and dynamic model compensation, while the impedance control law contributes torques only to the first three joints for position control. This is done by error weighting, where the position errors of the Cartesian controller are multiplied with zero.

2 Results and Possibilities of Improvement

With the above-mentioned approaches, the controller is able to fulfill all tasks effectively. The following plots display the motion and errors of the TCP when following the circular trajectory and controlling the orientation towards a fixed non-moving, as well as the red moving target. The performance with respect to obstacles is presented in the included video.

There are, however, certain aspects that I would have liked to improve with more time available. The biggest challenge in this project for me was dealing with the joint velocity limits of the last three joints and this challenge is visible in the results. Tuning the joint gains was a constant compromise between accuracy and avoiding joint velocity limits. In the final submission, I've configured the gains to be a bit higher compared to the values I used during working on the tasks in order to reduce the orientation errors when following the moving target while moving the end-effector along the circular path. However, this comes at the cost of a higher probability of exceeding the limits, especially when changing the active target. Further, this problem caused the need for a slower orientation interpolation which leads to sub-optimal trajectories when changing targets due to the fact that the longer trajectory enables the new target to move further away from the planned target orientation. One strategy that mitigates this issue to some extent is applied now, where the proportional and the derivative gains of the last three joints are lowered during orientation interpolation. Given more time, I would have liked to figure this out in more detail.

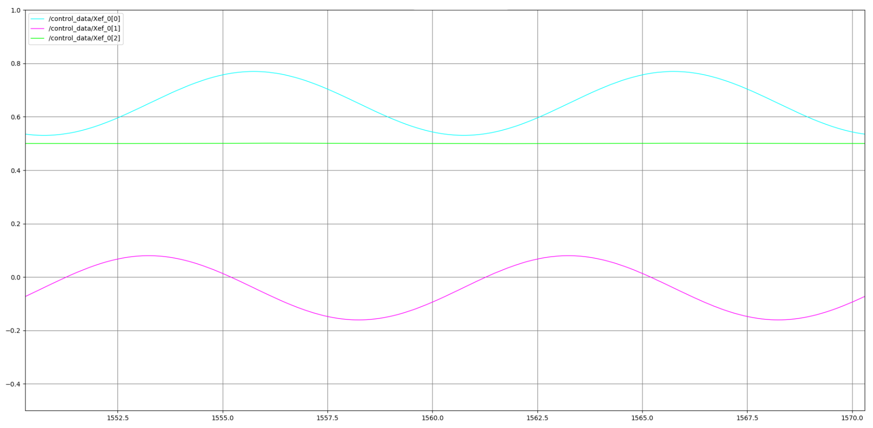


Figure 1: The tool trajectory in x,y,z-coordinates when following the circular trajectory and controlling the orientation towards a fixed non-moving target.

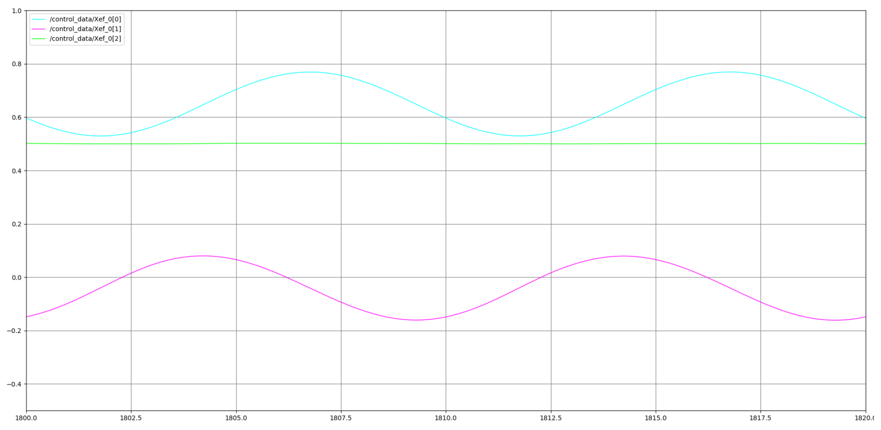


Figure 2: The tool trajectory in x,y,z-coordinates when following the circular trajectory and controlling the orientation towards the moving red target.

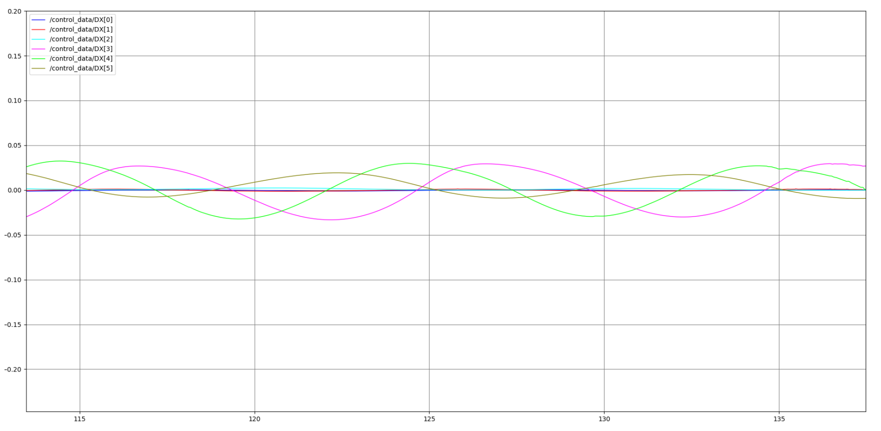


Figure 3: The position errors (m) [0-2] and orientation errors (rad) [3-5] when following the circular trajectory and controlling the orientation towards a fixed non-moving target.

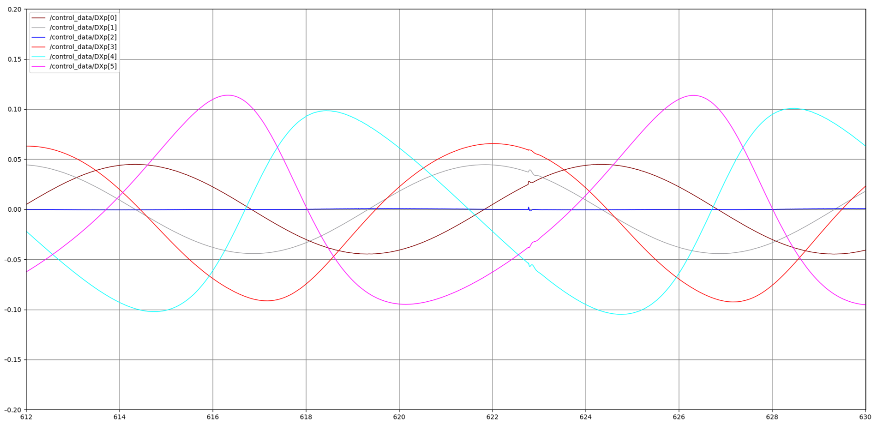


Figure 4: The linear [0-2] and angular velocity [3-5] errors when following the circular trajectory and controlling the orientation towards a fixed non-moving target.

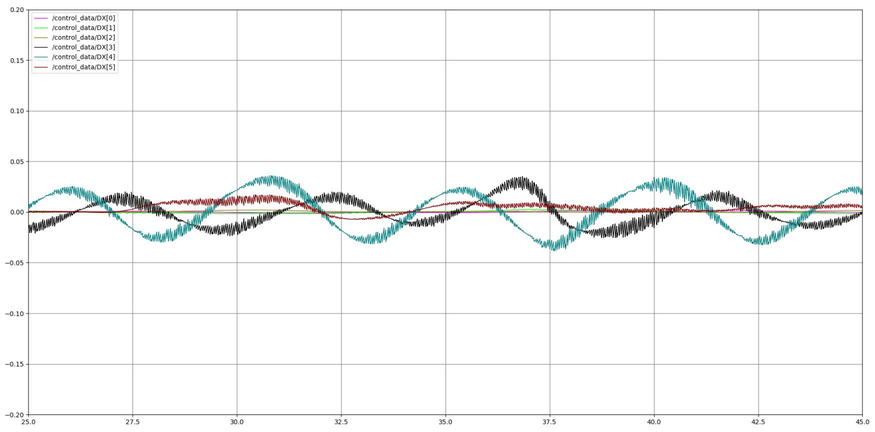


Figure 5: The position errors (m) [0-2] and orientation errors (rad) [3-5] when following the circular trajectory and controlling the orientation towards the moving red target.

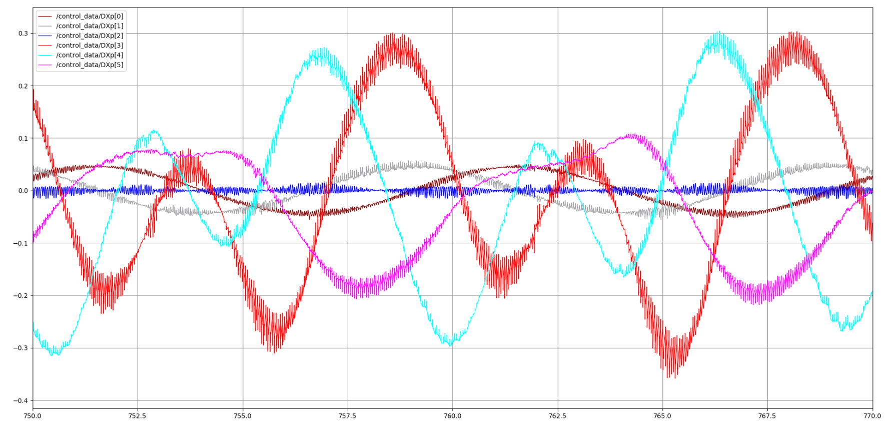


Figure 6: The linear [0-2] and angular velocity [3-5] errors when following the circular trajectory and controlling the orientation towards the moving red target.