16-662 Robot Autonomy
Assignment 2 – Motion Planning and Collision Avoidance

Heethesh Vhavle

Carnegie Mellon University

March 20, 2019

NOTE: The Python scripts are attached along with the zip file

Task 1 Command: `python2 cuboid_collision.py`
Task 3 Command: `python2 planner.py`

Folders may need to be created if plots are to be saved
Please use the updated URDF file attached and the VREP `lib2` library
Additional modules: `tf.transformation, numpy, matplotlib, scipy, sklearn`

Problem 1 – Solution

The results of collision checks are listed below.

TEST CASE	RESULTS
1	No
2	No
3	Yes
4	Yes
5	Yes
6	No
7	Yes
8	Yes

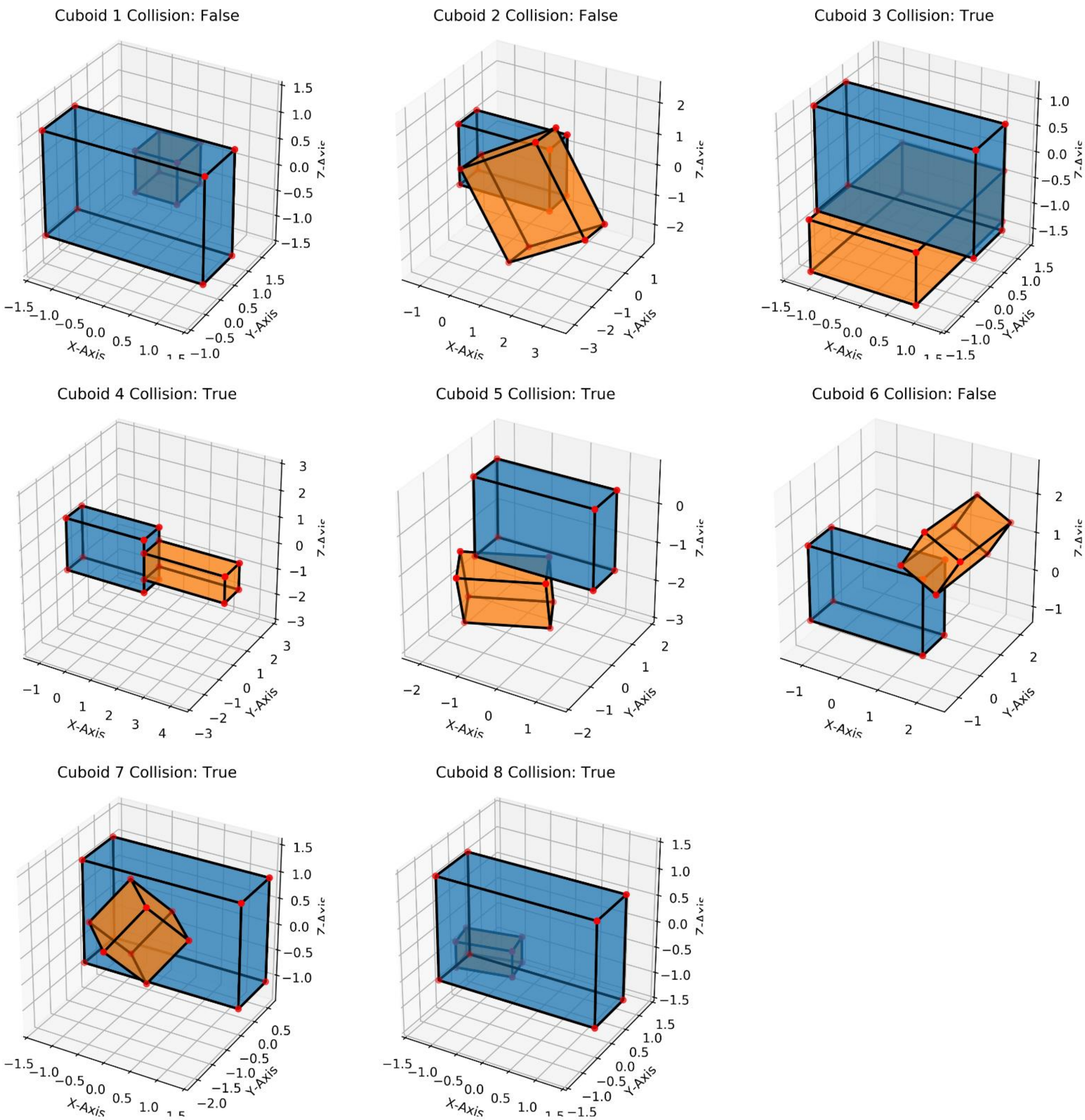

Figure 1: Results of obstacle cuboid (orange) collision check with the reference cuboid (blue).

Problem 2 – Solution

The bounding boxes were programmatically obtained from VREP. I initially faced few issues with getting the orientation of the bounding boxes. I then switched to querying quaternions from VREP. This is the reason I used an updated version of the VREP utils library (*lib2*) which I have attached. However, I later switched back to Euler angles and following the conventions specified in the VREP documentation, the bounding boxes were displayed correctly. I am still using the *lib2* module to maintain compatibility with quaternions.

Problem 3 – Solution

There was one change I had to make in the URDF file to update the *joint_1* position offset. I queried the *joint_1* position from VREP by setting all the joint angles to zero. I then updated the URDF file with the joint offset obtained from VREP. This helped me reuse my forward kinematics code from the previous assignment.

A brief overview of my implementation of the PRM Planner is explained below.

1. Compute the link pose and joint pose with respect to world frame for the initial arm configuration.
2. Compute link to joint transforms for the initial arm configuration (remains the same for all configurations).
3. Sample *N* random arm joint configurations and check if they are in collision with any of the obstacles.
4. All the arm and gripper links are checked for collisions with the obstacle cuboids and self-collision check is performed for all non-adjacent links. All link vertices are checked if they are above ground plane as well.
5. Build a KD Tree with the sampled arm joint configurations.
6. Query the KD Tree for *K* nearest neighbors and use a local planner to build the roadmap.
7. Local planner interpolates 5 points between two given configurations and runs forward kinematics to check if any of the intermediate states are in collision. The check is the same as explained before.
8. The joint and gripper targets (initial and final values) are connected to *K* nearest neighbors in the roadmap.
9. The roadmap defined earlier was stored as a directed graph with Euclidean distance as weights. The roadmap is then converted to an undirected graph to interconnect all the nodes with the initial and final values.
10. The global planner plans a shortest using Dijkstra's algorithm.
11. The planned trajectory is executed using the ArmController designed in the previous assignment.

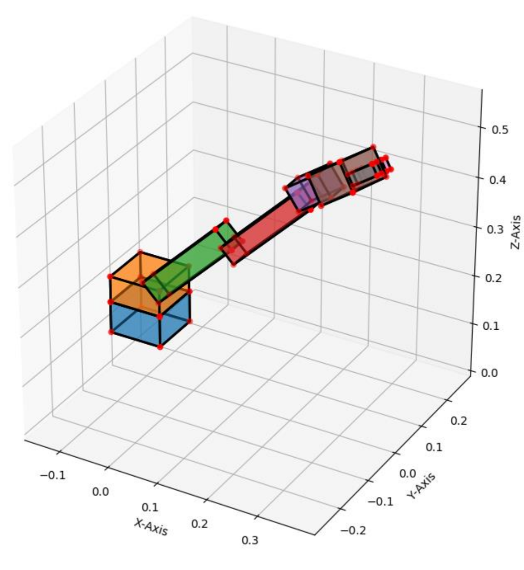

Figure 2: Visualization of arm link cuboids queried from VREP.

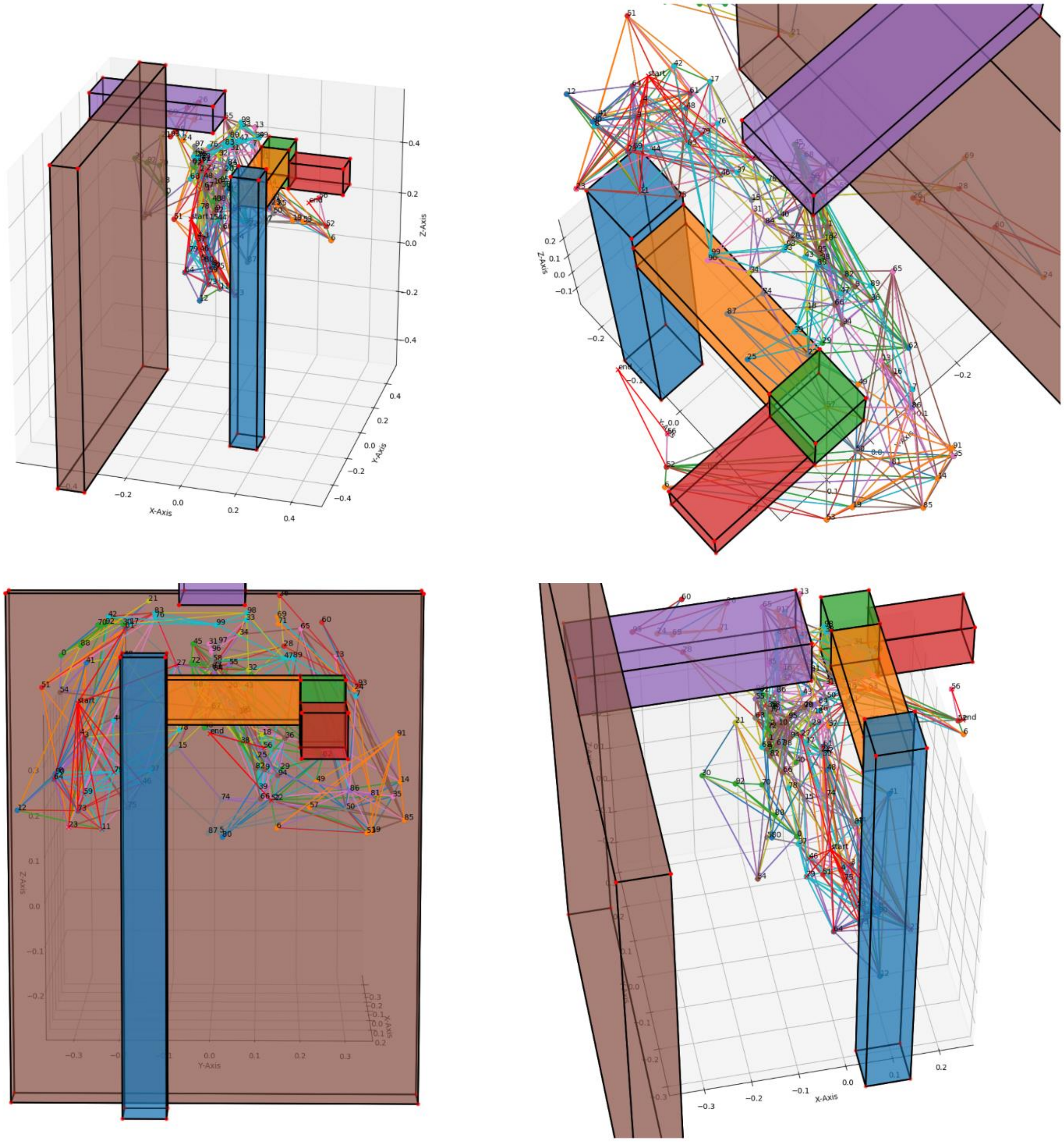

Figure 3: Visualization of the generated roadmap in Cartesian space for *N=100* and *K=10*.
(Some paths appear to be in collision with the cuboids due to display issues of cuboid transparency in Matplotlib)

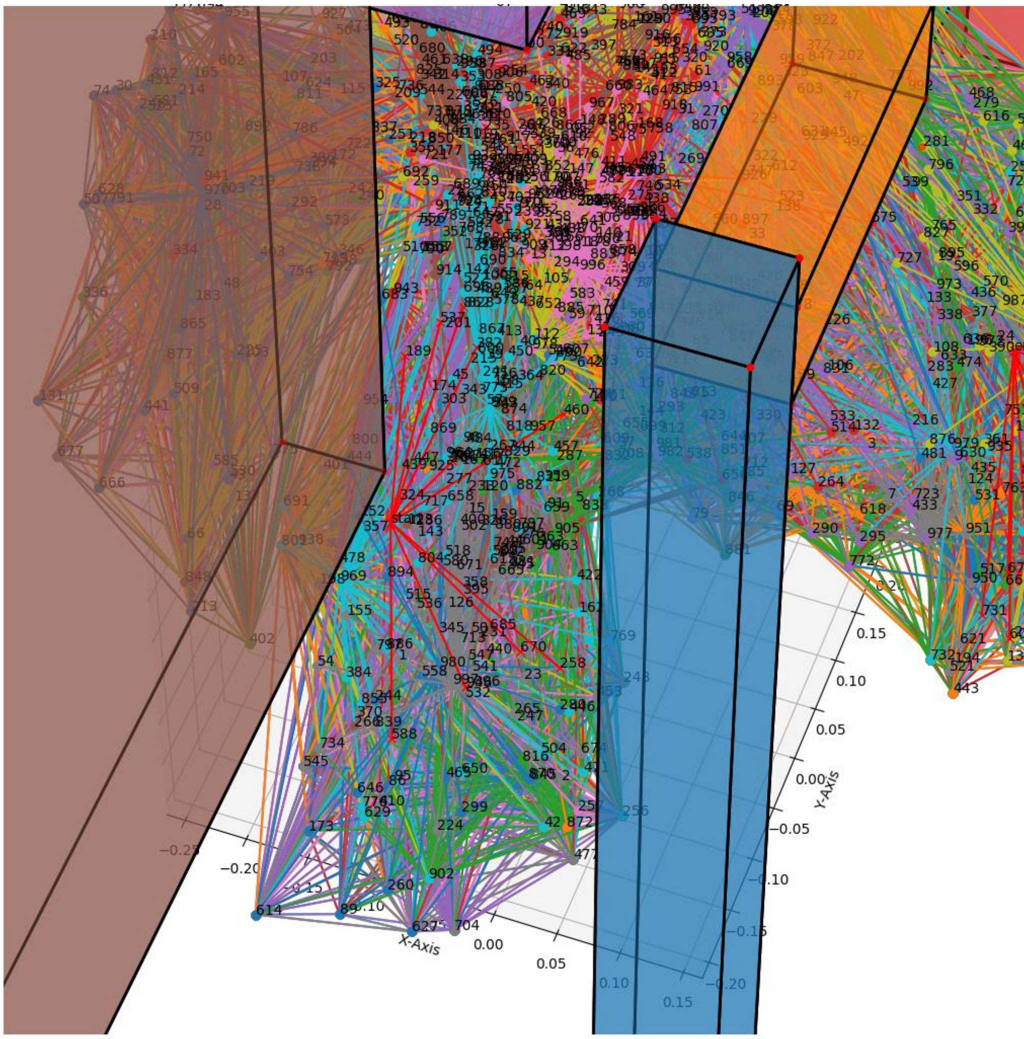

Figure 4: Visualization of the generated roadmap in Cartesian space for *N=1000* and *K=50*.
(Some paths appear to be in collision with the cuboids due to display issues of cuboid transparency in Matplotlib)

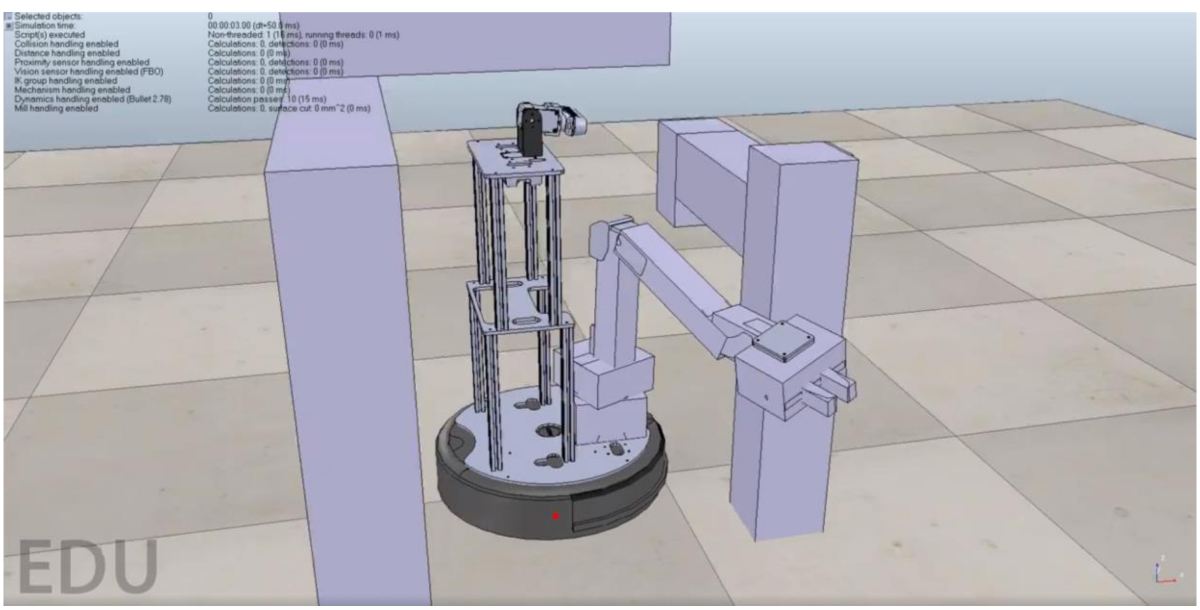

Figure 5: Link to video demonstration (https://youtu.be/nhoNoumHrlE).

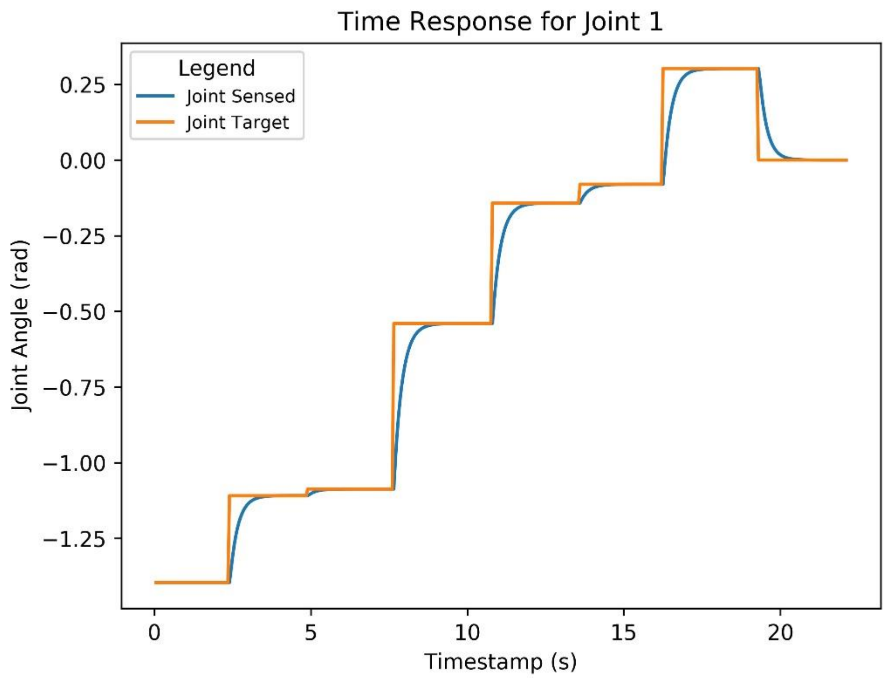

Figure 6: Time response of the closed-loop system with PID controller for Joint 1

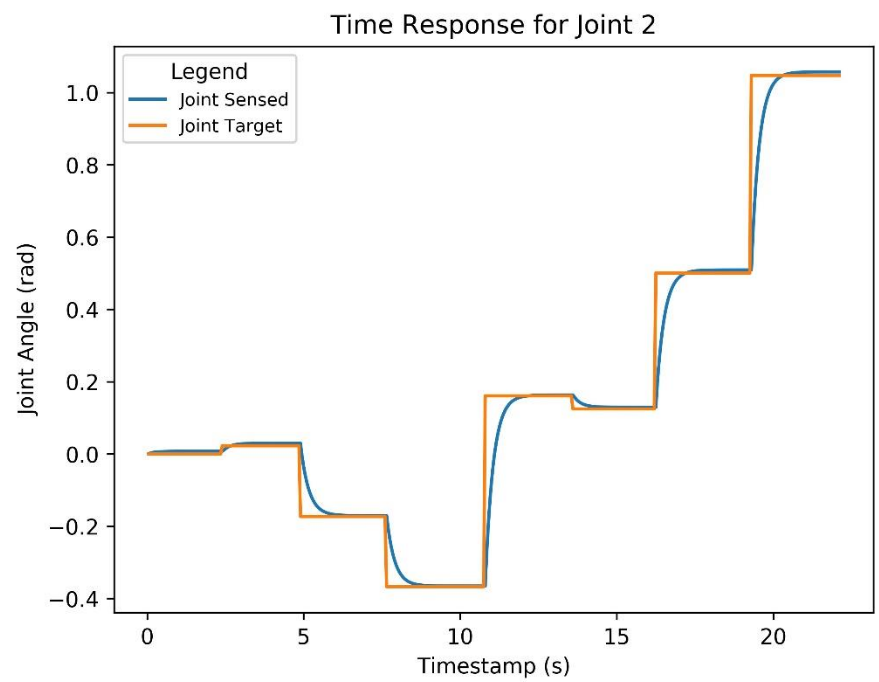

Figure 7: Time response of the closed-loop system with PID controller for Joint 2

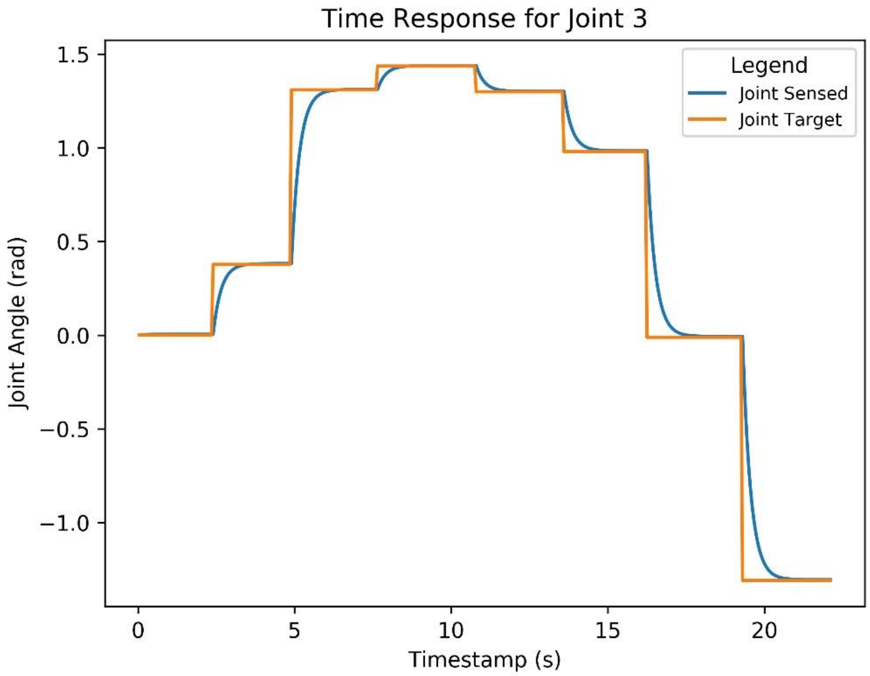

Figure 8: Time response of the closed-loop system with PID controller for Joint 3

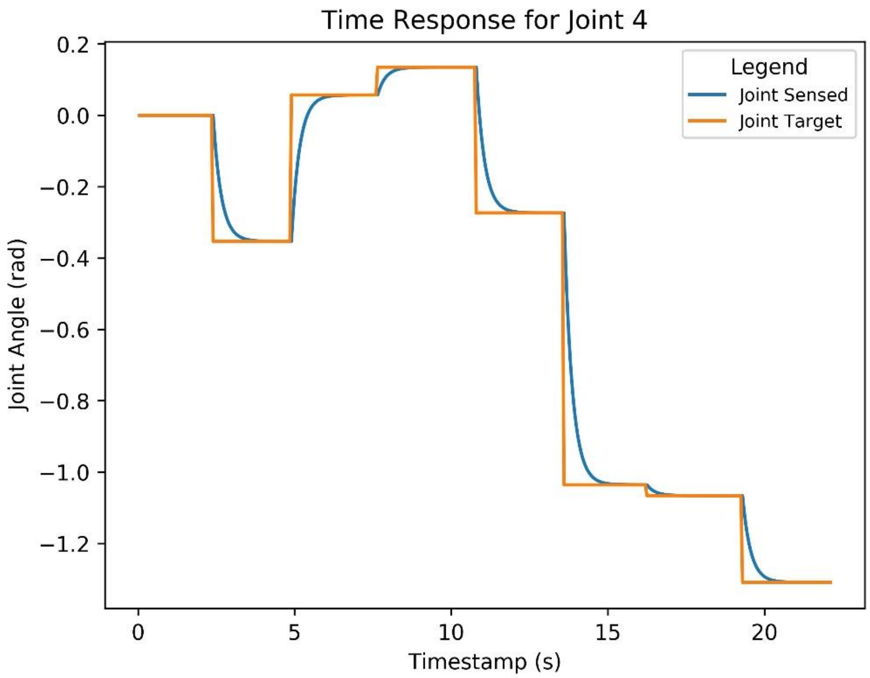

Figure 9: Time response of the closed-loop system with PID controller for Joint 4

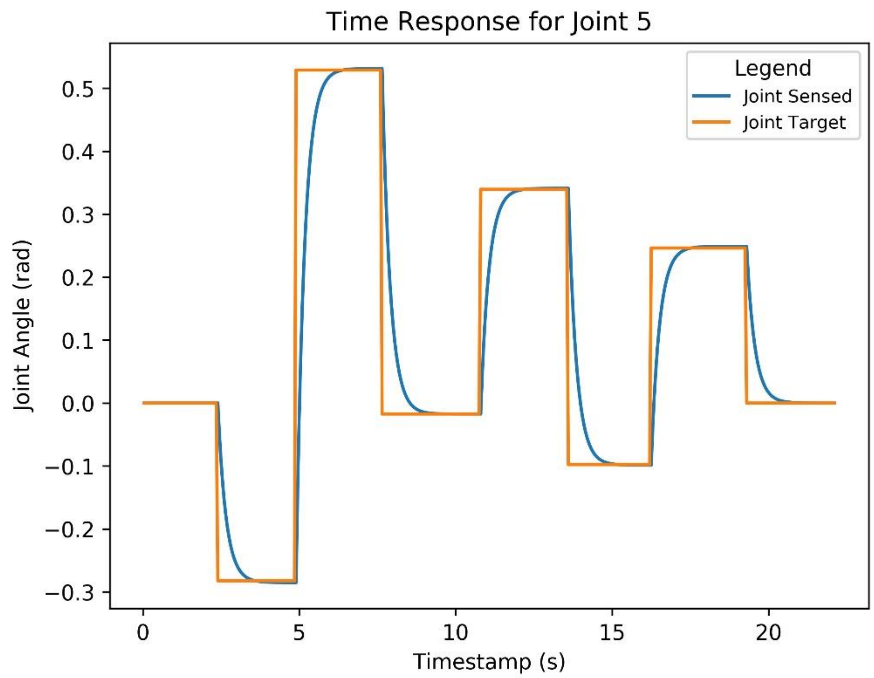

Figure 10: Time response of the closed-loop system with PID controller for Joint 5

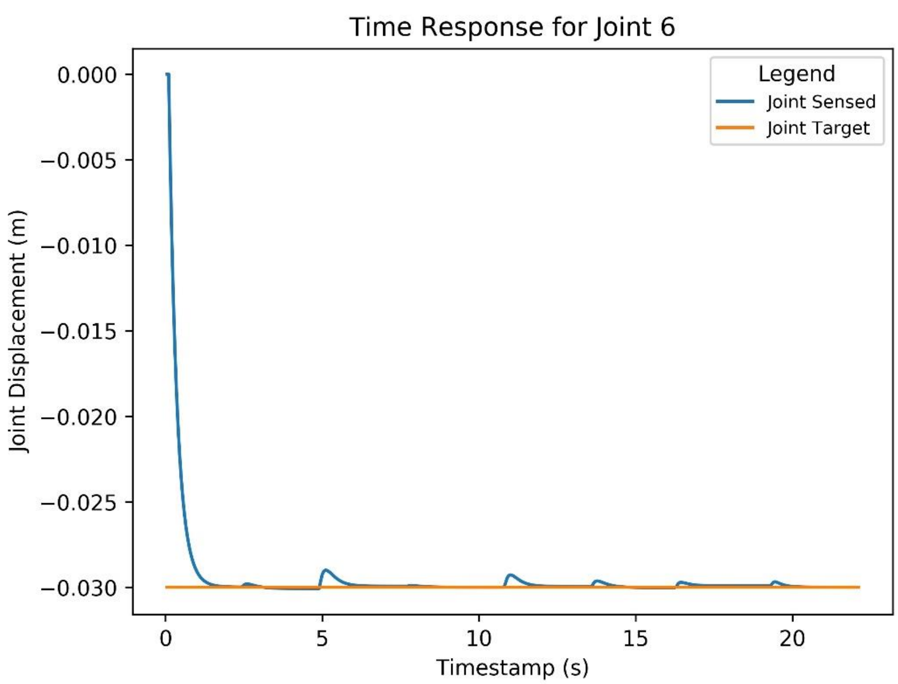

Figure 11: Time response of the closed-loop system with PID controller for Joint 6

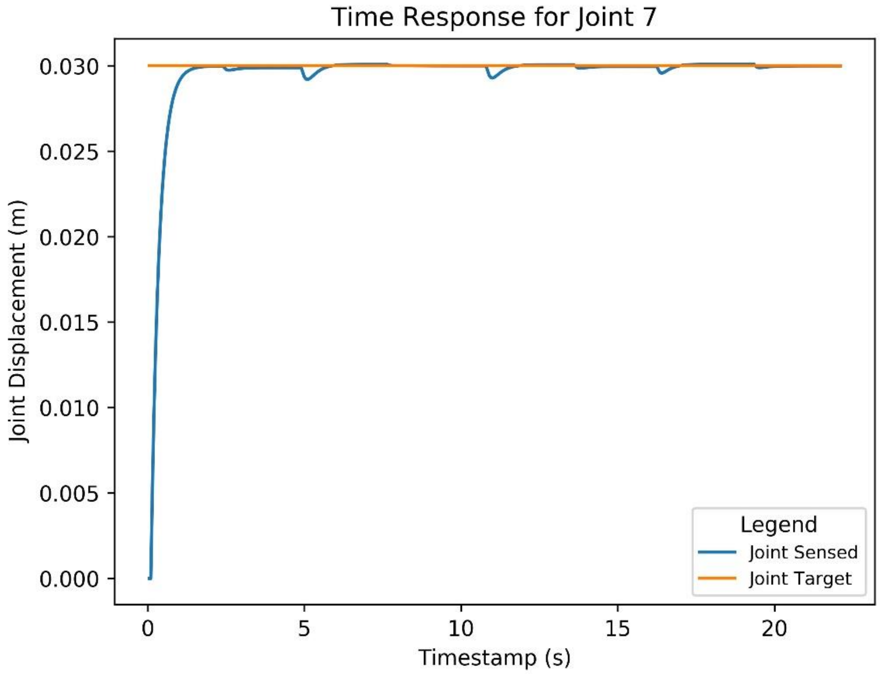

Figure 12: Time response of the closed-loop system with PID controller for Joint 7