Exercise 2: Reactive control and obstacle avoidance

Try out the example controllers in the Franka_ros package, by launching the controllers with the launch files.
Note: in simulation it will most likely not work, or crash after some time. Do it with the real robot.

- /catkin_ws/src/franka_ros/example_controllers

Make sure the robot is in its initial position before a controller is launched.

- move_to_start.launch

For example:
The Cartesian velocity controller runs the controller on the Franka Hardware, but the generation of the velocity command is computed in the node itself:
(line 88 in /franka_example_controllers/src/cartesian_velocity_example_controller.cpp).

Task 1: Reactive control

Create a new node that implements a control architecture. You can choose a reactive, deliberative or hybrid architecture. Behavior/actions can be e.g. Cartesian velocity/position, joint velocity/position, gripper action, etc. and the switching between them can be simulated (when reaching certain position or via keyboard input) or with a real sensor.

Tips:

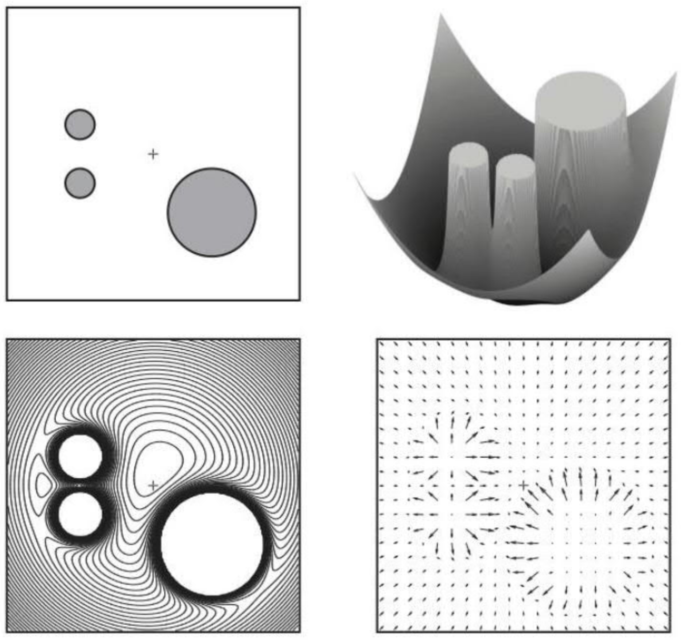
- Copy + adapt the example_controller node to subscribe to a command topic
- Put the architecture in a new node that you run separately, which publishes the command message
- Switching between behavior/actions can be done via e.g. if-loops, switch cases or more advanced with the ROS actionlib (http://wiki.ros.org/actionlib)
- Input for switching can also be simulated with ROS message:
  rqt_ez_listener or rostopic pub
- https://answers.ros.org/question/273480/publish-and-subscribe-array-of-vector-as-message/
- https://gist.github.com/alexsleat/1372845
- http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers
- http://wiki.ros.org/rospy/Overview/Publishers%20and%20Subscribers

Be creative!

Task 2: Obstacle avoidance

Developments for obstacle avoidance are quite similar to the previous task. Set up a node that does all the processing, listens to (or requests) the current state of the robot end-effector, and publishes the commands to control the robot.

1. Implement classic artificial potential field for point end-effector, point obstacle
   Hint: Do the 2D case first, similar to the figure below. Move the robot in xy-plane, from its initial position to a desired position, with an attractor function (e.g. velocity command). This attractor function (potential field) should be a map for the workspace of the robot. Then add simulated obstacle(s) in this map.


Potential field (xy-plane) with attractor point (center +) and three obstacles

2. Implement a code to keep the end-effector inside a given bounding box
   hint: make this part of the potential field. Demonstration of this can be done by changing the desired goal to be outside the bounding box. The robot motion should then not go outside the bounding box.
3. Avoid hitting the table while moving from A to B
   hint: the table is now also an obstacle. Treat this as the 3D case, where the motion cannot go below the table. Try to map only the table on which Panda is mounted, not an infinite plane.

Demonstrate these with multiple points inside and outside the robot's defined workspace.

Bonus 1: implement 3 for whole robot body
hint: see nullspace motion in cartesian_impedance_example_controller.cpp
Hint: see elbow_example_controller.cpp
Bonus 2: implement 1 for a moving obstacle

Evaluation: 3.10.2018 @ 2PM in K2341D