Appendix A

Making it Work

Getting Started First of all, complete a proper configuration of ROS and MoveIt! (respective websites available in the Bibliography).
Now, download my collection of files from :

```
https://github.com/AlessandroSantoni/SHERPA-Arm-MoveIt-Controller
```

and "catkin_make" them in the "src" folder where you installed MoveIt! too.
We're now ready to make some tests.

Homing Procedure We will simply ask the arm to plan to the "home" pose, which has been defined in the *.srdf file*. This would make easier the planning into more complex configurations, since at the very beginning the arm stands in a *singular* position.
You can refer to the source file `rover_arm_homing.cpp`.
Execute in the command line:

```
roscore &
roslaunch rover_arm_moveit_applications rover_arm_homing.launch
```

Note : If the planner seems not to display the motion, tick the checkbox "loop animation".

Joint Space Goal With this script, a setpoint in terms of positions is given to some joints.
Refer to the source file `rover_arm_joint_space_goal.cpp` and adjust the position values vector to the desired ones, if needed.
On a terminal, run:

```
roscore &
roslaunch rover_arm_moveit_applications rover_arm_joint_space_go-
al.launch
```

Simple Obstacle Avoidance Hereafter, a simple application of obstacle detection and avoidance is reported. This time, an *end-effector pose setpoint* is given; the trajectory will be computed at first in an obstacles-free environment and then, keeping the same desired pose for the gripper, obstacles will be introduced.
This is a "preliminary" script because, later on, we are going to implement obstacle detection through the camera sensor.
Refer to source code in `rover_arm_obstacle_avoidance.cpp`.
To test this application, execute :

```
roscore &
roslaunch rover_arm_moveit_applications rover_arm_obstacle_avoid-
ance.launch
```

Obstacles Detection with Camera Sensor In order to launch this functionality, we will need to set up a proper "collision scene" and give an end-effector pose that will make the test meaningful.
In my script,the setpoint is given with respect to my "testing-room settings", and it's a point behind an object (as shown in Figure 5.2). You should try to do the same.
The script `rover_arm_obstacle_on_camera.cpp` is similar to the one above, the only difference lies in the launching of `demo_tf_modified_camera.launch` rather than the regular RViz `demo.launch`. In fact, the first one, will make calls to the camera drivers, sensors configuration and OctoMap Update algorithm.
Launch this file:

```
roscore &
roslaunch rover_arm_moveit_applications rover_arm_obstacle_on_ca-
mera.launch
```

Target Recognition As explained in the appropriate chapter, we can also retrieve the reference frame of a desired object, by including its distinctive image pattern in the described path.
In verifying this application, we will practically put all things together. The recognition of an object will be made on top of the collision scene updating and the motion planning capabilities.
Here I list the commands to run the intended nodes:

```
roscore &
roslaunch rover_arm_moveit_config demo_tf_modified_camera.launch
```

After playing a little bit around with this "demo", in a second terminal execute:

```
roslaunch find_object_2d find_quadrotors_online.launch
```

Notes :

- I launch the "online" version of this last script, because I assume the camera drivers already running (those services are called by "demo_tf_modified.launch").
- To save your target image in the specified folder, follow the citation I made of the ROS package "find_object_2d".
- The launch file "find_quadrotors_online.launch" can be obviously called in the previous applications too; I launched the demo just as a test.
- Remember to add the "TF" panel in RViz to display the reference frame of the recognized object.