B. Changes for the Real Robot

B.1. Packages

The following packages were added in order to work with the real robot and the Realsense camera:

Package	Source	Description
real_robot_explorer	Own, `https://github.com/kingjin94/real_robot_explorer`	Contains the launch scripts for the camera calibration and demo on the real robot. Does all the rerouting to run the explorer from the simulation stack on the real robot.
tuw_marker_detection	`https://github.com/tuw-robotics/tuw_marker_detection`	Contains tuw_checkerboard which we use to track the pose of a checkerboard for the calibration of the in-hand camera.
easy_handeye	`https://github.com/IFL-CAMP/easy_handeye`	Package for the extrinsic camera calibration.
realsense-ros	`https://github.com/IntelRealSense/realsense-ros`	Publishes the images provided by the Intel RealSense camera via librealsense to ROS.
librealsense (Version 2.29)	`https://github.com/IntelRealSense/librealsense`	Driver and library to interact with the Intel RealSense camera. The driver parts have to be installed on the host, outside of Docker. They needed to be compiled due to a non-supported Linux kernel version on the host.

panda_moveit_control_only	`https://github.com/kingjin94/panda_moveit_control_only`, based on franka_ros and panda_moveit_config	Contains the controller parts of the MoveIt stack to be run on a separate PC interacting with the Panda Controller

Table B.1.: Overview of additional packages for the real robot.

B.2. Topics

The following topics were renamed due to new publishers, are new or were replaced:

Simulation Topic	**Real Robot Topic**	**Description**
/panda_arm_controller/follow_joint_trajectory	/position_joint_trajectory_controller/follow_joint_trajectory	New action interface to sent joint trajectories to.
/panda/depth_camera/depth_image	/camera/aligned_depth_to_color/image_raw	Raw depth image from the Realsense camera aligned with the color image.
/panda/depth_camera/depth_image/camera_info	/camera/aligned_depth_to_color/camera_info	Parameters of the depth image from the Realsense camera.
/panda/depth_camera/image	/camera/color/image_raw	Raw color image taken by the Realsense camera.
-	/franka_state_controller/F_ext	Libfranka's estimation of the wrench acting on the end-effector.
/panda/bumper/panda_probe_ball	/panda/bumper/panda_probe_ball	Information about collisions with the tip of the probing tool. Only the contact position is set.

Table B.2.: Overview of additional or changed topics for the real robot.

B.3. Nodes

The following Nodes were added or changed:

- CameraTransformPanda – Publishes the transform between the color image's frame in ROS convention (x to the right of the image, y to the bottom) and Panda's hand frame. It is set with the calibration data from section B.5.
- CameraTransformRealSense – Publishes the transform between Panda's hand frame and the Realsense internal frame camera_link. It is **not** calibrated!

- CollisionObserverNode – Applies the theory from section 6.1.1 to map observed forces on the end-effector (`/franka_state_controller/F_ext`) to a contact position. This is published on `/panda/bumper/panda_probe_ball` as a `gazebo_msgs/ContactsState` message to keep compatibility with the simulation code; only its `contact_position` field is valid though.
- camera/realsense2_camera & camera/realsense2_camera_manager – Nodes responsible for interactions with the Realsense Camera. They especially publish the images and can be used to change the cameras parameters with rqt-reconfigure.
- controller_spawner, franka_control, franka_gripper, joint_state_desired_publisher, joint_state_publisher, robot_state_publisher, state_controller_spawner – These are responsible for interaction with the real robot, generating movement commands from the planned trajectories and publishing the state of the robot. All of them are running on the real-time capable PC B that directly interacts with the robot.

B.4. Network Configuration

As stated in subsection 6.1.3 the real demo is running on multiple computers. **PC A** is the same used for the simulation. **PC B** is the PC communicating with the Panda Controller, telling it where to move to and publishing the robot state on ROS. The third computer is the Panda Controller delivered with Panda.

PC A and PC B are connected to the same IP network; make sure they can ping each other. PC B and the Panda Controller interact via a direct Ethernet link; follow Franka Emika's instructions to set up Desk, libfranka and franka_ros[1].

PC A will act as the Master of the ROS network. Its Master URI must be known to both PCs by setting the environment variable `ROS_MASTER_URI` to `http://<IP PC A>:11311`. This happens for the docker container on PC A if it is started with the **second** start command found in `enhanced_sim/install/commands_for_docker`, which is intended for the use with the real robot. On PC B it has to be set manually. Once this is done test their connection by running the listener / talker example from the ROS tutorials[2]. You might have to manually add the host name and IP address of PC A to the Docker image's `/etc/hosts` if the roscore within the Docker image can not find itself.

In addition, PC B needs the package `panda_moveit_control_only` installed in its catkin workspace. This contains the launch script to start the lower parts of the MoveIt controller stack that interact with libfranka (`panda_control.launch`). One can either start it locally at the same time as the launch script on PC A or start it via ssh from PC A. To do the second, one has to set up an ssh connection between both PCs with RSA keys, adapt `real_robot_explorer/scripts/startControllerOnNUC.bash` on PC A and `panda_moveit_control_only/launch/remoteEnv.bash` on PC B, and uncomment the remote_controller node in `real_robot_explorer/launch/setupMoveIt.launch`.

Once these basic ROS interactions between the two PCs work, one can start running the calibration and main demo.

[1] `https://frankaemika.github.io/docs/overview.html`
[2] `http://wiki.ros.org/ROS/Tutorials/MultipleMachines`

B.5. Camera Calibration

To calibrate the extrinsic orientation of the camera make sure that it is connected, can be accessed by the realsense-viewer from librealsense and that the robot arm is running in guide mode. Also make sure that the network configuration allows ROS to communicate between PC A and B. The camera calibration can then be run with the following command: `roslaunch real_robot_explorer calibrate.launch` on PC A, which should open the windows shown in Figure B.1. The top one shows the output of the checkerboard tracker laid over the output of the camera. The one below is the interface for the calibration routine.

First test whether the checkerboard can be reliably tracked in the first window. Make sure that the coordinate system does not disappear when moving the robot's hand or the checkerboard, while keeping the board in the image frame. You may have to adapt its parameters with rqt's dynamic reconfigure under checkerboardfinder. In its current configuration it searches for a 9 by 6 checkerboard with a square size of $26mm$. For more details visit the tuw_marker_tracker website listed in section B.1. Once the checkerboard is found you can start the calibration in the second window. During the calibration you should not move the checkerboard. Only move the arm to different poses from which the checkerboard can be seen and press `Take Sample` at each of these. Record multiple poses trying to maximize the angle between each while reducing the translation of the camera [81].

Afterwards, hit `Compute` which will return the pose of the camera relative to the hand as a translation vector and a unit quaternion. These have to be put into `real_robot_explorer/launch/setupExplorer.launch` replacing the first seven arguments of the node CameraTransformPanda. Put in the translation components first than the rotation. The new calibration will then be used during exploration.

B.6. Main Demo

The main demo can then be run by first launching `real_robot_explorer/launch/setupExplorer.launch`. This will prepare the perception and control for later exploration. Wait for RViz to come up with a similar interface to the simulated case (Figure A.1). Once the whole perception pipeline is running, as seen by all the images at the bottom being filled, exploration can be started with the script `real_robot_explorer/launch/randomExplorer.launch`. Make sure to keep the enable button nearby in case the robot makes dangerous movements! The visual exploration may not end by itself, due to difficult to determine thresholds for a finished map. Once the OctoMap in RViz shows the geometry of the room the robot is in one may want to stop the exploration script, and adapt the thresholds on the entropy and table score in the exploration script (`enhanced_sim/src/randomExplorer2.py`); restart it afterwards to see the tactile refinement part.

If safe-guards of the robot's controller are triggered one can restart the controller with the command `rostopic pub -1 /franka_control/error_recovery/goal_franka_control/ErrorRecoveryActionGoal "{}"`. Furthermore, one may want to reduce the robot model's joint velocities by editing `franka_ros/franka_description/robots/panda_arm.xacro` in the lines noted with the comment `reduce for real robot`.

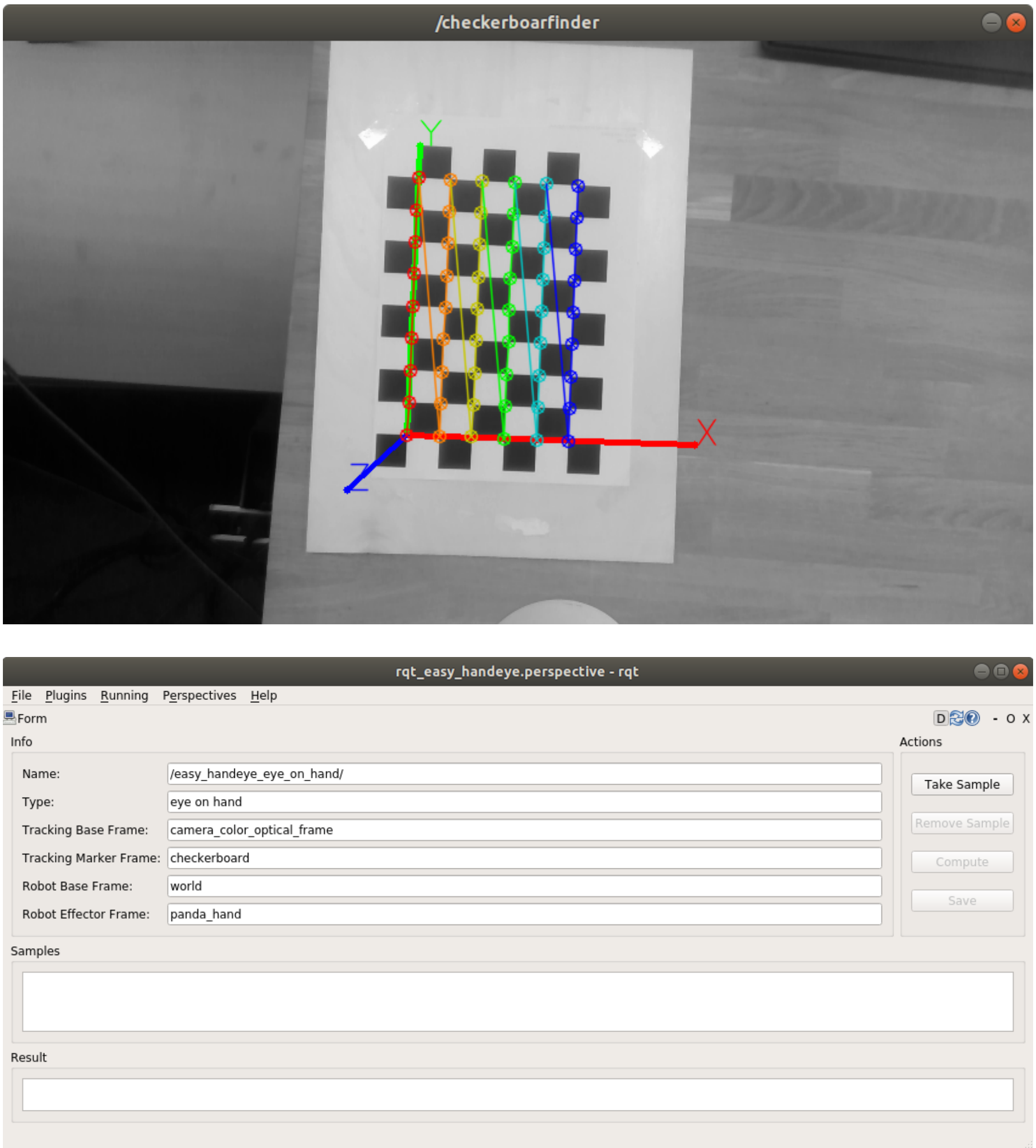

Figure B.1.: The two main windows for the calibration process. On the top the window for the checkerboard tracker, on the bottom the one for the hand-eye calibration.