A. 3D Simulation

The appendices describe how to run the demos in simulation and on the real robot, as well as the important packages, topics and nodes running within each. If you have further questions feel free to contact the author.

A.1. Packages

Table A.1 gives an overview of the self-compiled and modified packages within the visuo-tactile exploration demo. They are all located in `/home/catkin_ws/src` of the Docker container. Additional packages from the ROS repository were used but are not listed if they were not modified.

Package	Original Source	Description
franka_ros	`https://github.com/frankaemika/franka_ros`	Packages integrating Panda with ROS. We especially used franka_description as the basis for the simulated robot, extending it by the simulated camera and bumper sensors.
octomap_mapping	`https://github.com/OctoMap/octomap_mapping`, [32]	Manages the OctoMap; modified to be used with RGB-D data.
realtime_urdf_filter	`https://github.com/blodow/realtime_urdf_filter`	Creates the filter mask to remove self-observations of the robot. Extended to work with newer ROS, and to process the incoming depth image and segmentation mask into a RGB point-cloud (includes synchronization, dropping moving frames, remapping segmentation results, rescaling).
enhanced_simulation	Own, `https://github.com/kingjin94/enhanced_simulation`	Main package, including building the Docker image for the demo, and the main launch and exploration scripts.

panda_moveit_config	`https://github.com/erdalpekel/panda_moveit_config`, `https://github.com/ros-planning/panda_moveit_config`	Enables use of MoveIt to plan and execute trajectories on the Panda robot.
filter_octomap	Own, `https://github.com/kingjin94/filter_octomap`	Contains nodes that find tables and cans in the OctoMap, calculate the map's entropy and fill it with prior knowledge before starting. Contains message definitions for tables and cans.
mask_rcnn_ros	`https://github.com/akio/mask_rcnn_ros`, `https://github.com/matterport/Mask_RCNN`, [1], [31]	Node to segment RGB images into semantic labels.
panda_simulation	`https://github.com/erdalpekel/panda_simulation`	Package containing the Gazebo setup to simulate the Panda robot; especially the gains for the robot controller in `config/panda_control.yaml`

Table A.1.: Overview of modified packages.

A.2. Topics and Actions

Table A.2 lists the most important topics used in the demo.

Topic	Message Type	Description
/panda/bumper/colliding	enhanced_sim/CollisionState	Returns whether the robot is colliding with the environment and with which links
/panda/bumper/*	gazebo_msgs/ContactsState	More detailed collision state per robot link, including position and wrenches
/panda/ft/tool	geometry_msgs/WrenchStamped	Wrench in between the probing tool and the fingers.
/panda/depth_camera/depth_image	sensor_msgs/Image	Unfiltered depth image from the hand-held camera.

/panda/depth_camera/depth_image/camera_info	sensor_msgs/CameraInfo	Camera parameters for the depth image.
/panda/depth_camera/image	sensor_msgs/Image	Unfiltered color image from the hand-held camera.
/urdf_filtered_mask	sensor_msgs/Image	Filter mask indicating where the robot sees itself.
/panda/depth_camera/depth_image/filtered	sensor_msgs/Image	Depth image from the hand-held camera with the robot removed by setting those pixels to NaN.
/panda/depth_camera/depth_image/filtered/points	sensor_msgs/PointCloud2	3D point cloud from the depth image by the in-hand camera; points on the robot itself are removed.
/octomap_full	octomap_msgs/Octomap	OctoMap integrating all previous observations with full probability information.
/octomap_new	octomap_msgs/Octomap	OctoMap integrating all previous observations with binary occupancy information only.
/octomap_new/entropy	std_msgs/Float64	Total entropy density of the OctoMap published on /octomap_full offset by the initially unknown volume.
/octomap_new/table	filter_octomap/table	Best candidate for a table in the OctoMap, including min / max in x, y, z, a normal estimate and table score.
/octomap_new/table_touched	filter_octomap/table	Tactilely refined table parameters; with new maximum z and surface normal.
/octomap_new/cans	filter_octomap/cans	List and count of all cans found above the table in the OctoMap. Each can has a radius, height, centroid and score.

/octomap_new/cans_touched	filter_octomap/cans	Tactilely refined can parameters for can with highest score; with new radius, height and centroid.
/planning_scene	moveit_msgs/PlanningScene	Planning scene for moveit. Especially, contains /octomap_new to constrain the robot movement to known free space.
/joint_states	sensor_msgs/JointState	State of the robot's joints, including configuration, velocity and effort.
/tf(_static)	tf2_msgs/TFMessage	Transformations between frames of the kinematic chain.
/panda_arm_controller/follow_joint_trajectory	control_msgs/FollowJointTrajectory (Action)	Commands the robot to follow a joint trajectory; e.g. $(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t))$.

Table A.2.: Overview of published topics and actions.

A.3. Nodes and Executables

The following lists all ROS nodes and standalone executables, marked by an [e], for the demo. They are ordered by the package they are in and we provide a description for each:

- franka_ros
- octomap_mapping
  - octomap_server – Integrates RGB point clouds from `/panda/depth_camera/depth_image/filtered/points` into a common map based on octrees. Publishes this map with color and occupancy probabilities on `/octomap_full`.
- realtime_urdf_filter
  - cloudify – Takes in the filtered depth image (`panda/depth_camera/depth_image/filtered/image`), segmentation image (`panda/depth_camera/image/filtered/seg`) and camera information (`panda/depth_camera/depth_image/filtered/camera_info`) and produces a RGB point cloud on `panda/depth_camera/depth_image/filtered/points`. It removes the points marked with NaN coming from nanifier
  - nanifier – Node responsible for synchronization, replacing filtered depth points by NaN and rescaling the depth and segmentation images. It takes in the filtered depth image on `/panda/depth_camera/depth_image/filtered`

and replaces the pixels set to the robot value, here 50.0, with NaN. It takes the results from Mask R-CNN on /panda/depth_camera/image/seg_res and remaps the segmentation masks, such that only tables and cans are marked in green and red, respectively. To be able to rescale the images one has to alter the camera information on /panda/depth_camera/depth_image/camera_info, too. Furthermore, this node drops images where the transform is not well known due to ongoing movement, measured as the magnitude of the velocity on /joint_states. The four topics are synced and the results are published to /panda/depth_camera/depth_image/filtered/image (modified depth image), /panda/depth_camera/image/filtered/seg (modified segmentation image) and /panda/depth_camera/depth_image/filtered/camera_info (modified camera info).

  – realtime_urdf_filter – This node takes in the raw depth image from the camera and its pose, and masks all pixels in the depth image that see the robot itself by replacing them with a fixed value, here 50.0. It subscribes to the depth image on /panda/depth_camera/depth_image, camera pose from tf called /panda/panda_camera and the robot's URDF description on the parameter server under robot_description. It publishes the filtered image on /panda/depth_camera/depth_image/filtered and the filter mask on /urdf_filtered_mask. Details are provided in section 5.3.4.

- enhanced_simulation
  – CollidingNode – Summarizes the collision state of the robot by looking at all bumper topics (/panda/bumper/*) and publishing an overview on /panda/bumper/colliding.
  – touch_table / touch_table2 [e] – Node doing the tactile exploration of the found table as described in section 5.3.2. It subscribes to /octomap_new/table to get the initial guess for the table's position and sends the refined parameters on /octomap_new/table_touched.
  – touch_can – Node doing the tactile exploration of the found cans as described in section 5.3.2. It subscribes to /octomap_new/cans to get the initial guess for the cans' positions and sends the refined parameters of the best can on /octomap_new/cans_touched.
  – random_explorer2 – Node responsible for the high-level coordination of the exploration as stated in section 5.3.2. It first starts exploring visually and monitors the mapping progress on /octomap_new/entropy and the table score on /octomap_new/table. If both break a threshold tactile exploration of first the table (node touch_table) and then the cans (node touch_can) is started.
- panda_moveit_config
- filter_octomap
  – find_cans_in_octomap – Finds all cans on-top of the table from /octomap_new/table that are in the OctoMap from /octomap_new. Publishes its results on /octomap_new/cans. Employs the algorithm described in section 5.3.4.

- find_table_in_octomap – Finds the best table in the OctoMap from `/octomap_new`. Employs the algorithm described in section 5.3.4.
- planningSceneUpdate_entropyCalc – Subscribes to `/octomap_new` to calculate the map's entropy with Equation 5.1. Publishes it on `/octomap_new/entropy`. Additionally, updates MoveIt's planning scene by publishing the binary representation of the OctoMap to `/planning_scene`.
- octomap_generate_empty_start [e] – Used before starting the demo to initialize an empty OctoMap with the assumed prior knowledge of free space. Takes as a single argument the desired resolution in meters.

• mask_rcnn_ros
  - mask_rcnn – Segments the RGB image into different semantic classes, as described in section 5.3.4. Subscribes to the RGB image on `/panda/depth_camera/image` and publishes the segmentation results on `/panda/depth_camera/image/seg_res`. A debug view is published on `/panda/depth_camera/image/seg`.

• tf
  - CameraTransform – Publishes the static offset between the world frame and camera frame of a statically mounted camera in the robots workspace.
  - CameraTransformPanda – Publishes the static offset between the pose of the camera block in the robots hand and the camera coordinate system.

A.4. Robot Parameters and Description

The simulated robot is based on the Panda by Franka Emika. We use the franka_description inside franka_ros [1] as a basis for our simulated robot. On top of this Erdal Pekel implemented a simulation in gazbeo for the panda robot [2] which we modified for our needs.

As the kinematic parameters of the panda robot are not available publicly we took the same as guessed by Erdal Pekel in [60]. They are based on assuming a homogeneous robot density and giving each link its weight according to its mesh's volume. The values are given in Table A.3. The inertia tensors were chosen to be diagonal with an amplitude of $0.3\frac{kg}{m^2}$. We would like to note that the chosen inertia values seem not realistic, especially the smaller parts have a to high inertia. More realistic values could probably be drawn from [28], but the reduced inertias and joint friction would probably require more sophisticated control.

The control of the simulated robot is handled via *libgazebo_ros_control* from the ROS package gazebo_ros_control. We mostly kept Pekel's configuration, which defined *effort_controllers/JointTrajectoryController* for each the arm and the hand, which are PID controllers. Their gains are defined in `panda_simulation/config/panda_control.yaml` We

[1] `https://github.com/frankaemika/franka_ros`
[2] `https://github.com/erdalpekel`

Link	Mesh	Volume $[m^3]$	Mass $[kg]$	Inertia $[\frac{kg}{m^2}]$
0	collision	0.002996	3.06357	0.3
1	visual	0.002293	2.34471	0.3
2	visual	0.002312	2.36414	0.3
3	collision	0.002328	2.38050	0.3
4	collision	0.002374	2.42754	0.3
5	collision	0.003419	3.49611	0.3
6	collision	0.001435	1.46736	0.3
7	collision	0.000446	0.45606	0.3
hand	collision	0.000709	0.67893	0.3
Finger 1 / 2	visual	0.000011	0.01053	0.3

Table A.3.: Volume, Masses and Intertias of the robot links from [60] and own calculations of inertias.

Joint	P	D	I	I-clamp
panda_joint1	18000	100	0.0	10000
panda_joint2	50000	200	0.02	10000
panda_joint3	24000	100	0.01	1
panda_joint4	24000	140	0.01	10000
panda_joint5	18000	140	0.01	1
panda_joint6	10000	300	0.01	1
panda_joint7	3000	350	0.0	1

Table A.4.: PID gains for the robot arm.

added gravity compensation by importing the plugin *gravity_compensation* into the robot's Gazebo configuration (`franka_ros/franka_description/robots/panda.transmission.xacro`) and tuned the PID gains to the values given in Table A.4.

Moreover, there are PD controllers responsible for enforcing the joint limits which turned out to be to stiff which led to oscillations in the simulation. Their gains are set in `franka_ros/franka_description/robots/panda_arm.xacro` in the *safety_controller* tag and we reduced it to 40% for joint 3 and 4, to 20% for joint 5, and 10% for joint 6 and 7.

A.5. Using the Simulation

The simulation is provided as a Docker image which can be build with the help of the main repository: `https://github.com/kingjin94/enhanced_simulation`. To run the image and simulation one requires a Linux PC with atleast a Quad Core, 16 GB of RAM, an NVidia GPU, as well as Nvidia-docker[3] installed.

The image can be build by the provided shell script `install/build_all.sh`. The build script needs your private ssh key to access the private repositories generated for this project. Instructions on how to work with docker containers, especially how to start the simulation container, are provided in `install/commands_for_docker`.

[3] `https://github.com/NVIDIA/nvidia-docker`

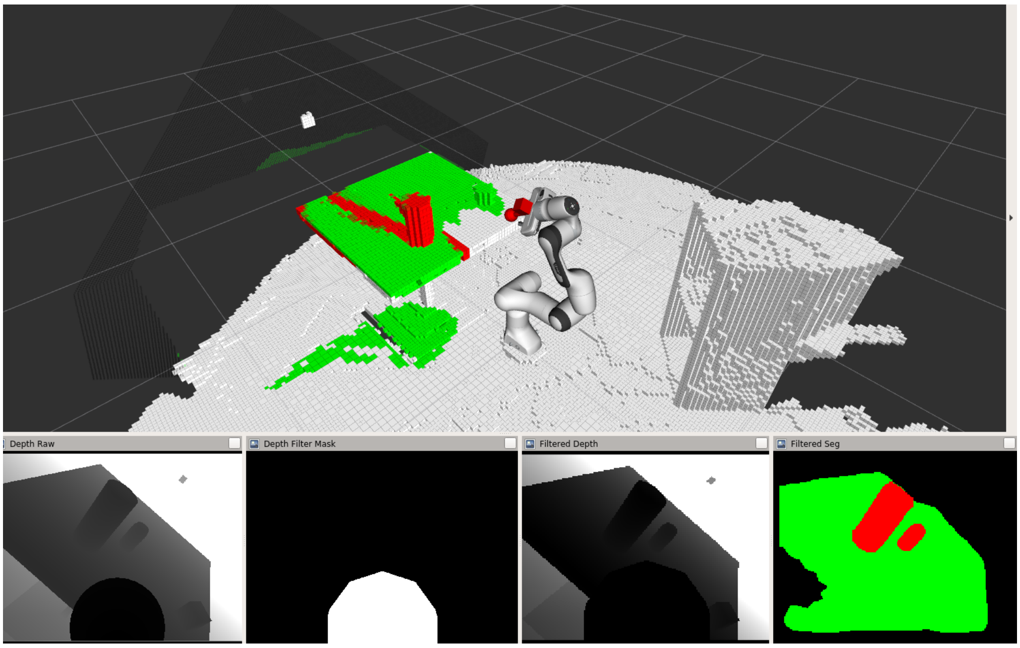

Figure A.1.: Expected RViz debug view after the end of visual exploration.

The inside of the container is a full fledged ROS melodic[4] install with the Gazebo simulator. The catkin workspace with all the self-written code is in `/home/catkin_ws`. The simulation can be started by running `roslaunch enhanced_sim explorer.launch`. This should open the simulation in a Gazebo window, as well as a debug view in RViz, which is shown in Figure A.1. One might have to execute `source devel_isolated/setup.bash` if the launch script is not found.

The first start may take a while because Gazebo and Mask R-CNN have to download additional content which should be done once the Gazebo window opens. This delay may cause the robot controllers to shutdown requiring a restart of the whole simulation. If the error `Could not open file /home/catkin_ws/src/filter_octomap/maps/prefilled0.02.ot` appears go into the `filter_octomap` package and run `makeMaps.sh`.

The debug view shows some of the topics mentioned in Table A.2, especially the robot's current configuration in the OctoMap (main window, colored cubes). Below we show the depth image, URDF filter mask, filtered depth image and semantic segementation (from left to right). The perceived point cloud is shown as colored markers in the main window, visible in black and green in the background. Debug information regarding the segmentation of the table and cans can be added as *Marker Arrays*, which show the segmented candidates, final selected set and for the cans the fitted cylinder.

After the simulation is started one can initiate the scene exploration by running `rosrun enhanced_sim random_explorer2`. The robot will start moving through the simulation and the OctoMap should start to reflect the surrounding environment.

Once the generated map is informative enough and the table has been found with a high enough score, the robot will start tactile exploration. During tactile exploration

[4]`https://www.ros.org`

the robot will touch different parts of the visually found table and thereby refine its knowledge of the table's location. Afterwards, the highest scoring can will be probed, too.