Laboratory exercise 5

SofIA: Calibration and Perception

Name: Luka Vukelić **JMBAG:** 0036530728

Preparation

- Review ROS tutorials and lecture slides.
- If you haven't already done so, set up ROS environment.
- Clone an existing sofia_perception git repository: **$ git clone https://github.com/larics/sofia_perception**. The repository *sofia_perception* contains three python scripts, launch file and resource files.
- **Answer all the tasks in this document before the presentation of the laboratory exercise in your time slot**

Introduction

SofIA (Soft Finger AI-enabled Hand) is a passive, adaptive, bioinspired gripper that uses monolithic urethane rubber fingers with visual feedback to measure finger compliance. It is based on the Fin-Ray ® effect discovered by biologist Leif Kniese of Evologics while fishing, which is based on the deformation of fish fins under load.

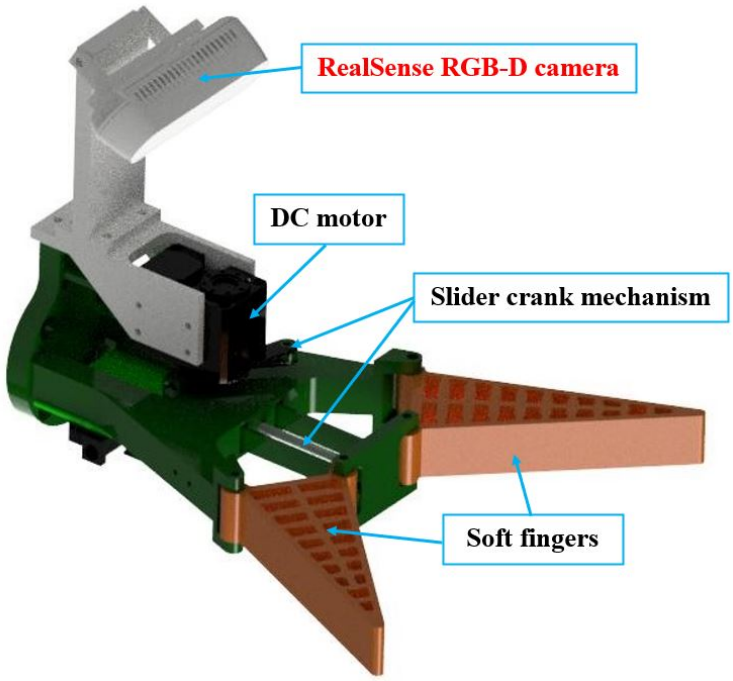

Figure 1: Main components of SofIA. It consists of four main elements: Intel RealSense D435 depth camera (RGB-D camera), Dynamixel MX-64 DC motor, slider crank mechanism, and soft fingers.

The gripper is powered by the Dynamixel MX-64 DC motor, which drives the slider crank mechanism. The slider crank mechanism converts the rotational motion of the motor into a translational motion of the slider

(see Figure 1). The achieved translation activates the fingers, which close/open according to the direction of rotation of the DC motor. Since the fingers are soft, they passively adapt to the objects and do not harm them while holding them firmly during gripping.

SofIA is also equipped with an Intel RealSense D435 depth camera (RGB-D camera) that provides visual feedback on the state of the fingers and enables complex and detailed modeling of the manipulated environment. To achieve accurate and desired camera behavior, the RGB-D camera must be properly calibrated before putting SofIA into operation.

In our setup, SofIA is mounted on Franka Emika Panda collaborative manipulator (manipulator). Knowing the transformation from robot flange frame L_f to camera frame L_c, the positions of objects detected in the camera coordinate frame L_c can be transformed to robot base frame L_0 or robot flange frame L_f and used as a reference for the trajectory planning of the manipulator. After positioning the manipulator accordingly, the gripper closes, and grasped object can be moved from location A to location B.

Assignments

Task 1: Calibration (2 points)

To properly use the information from the camera for the positioning of the manipulator, the manipulator-camera system has to be calibrated. The result of the autonomous calibration process is a transformation $\mathbf{T}_f^c \in \mathbb{R}^{4\times4}$ between the robot flange frame L_f and the camera (optical) frame L_c.

A simple contrasting blob (L_B) is used as a target in the camera calibration procedure (see Figure 4). The blob is fixed during the recording procedure and the manipulator is moved to record the blob. The position of the blob $\mathbf{p}_W^B$ in the world frame L_W does not need to be *a priori* known. Once the target is recorded and detected in the image from up to n poses $i = 1, ..., n$, its 3D position in the camera frame L_c is extracted from the organized point cloud. The resulting set of detections contains positions $\mathbf{p}_{c\ i}^B \in \mathbb{R}^3$ of the target in the local camera frame L_c along with the joint configuration of the manipulator $\mathbf{q} \in \mathbb{R}^7$ in which the frame was recorded. Joint values enable us to calculate the transformation matrix $\mathbf{T}_0^f(\mathbf{q}_i)$ between the robot base frame L_0 and the robot flange frame L_f, where the gripper is mounted. Finally, treating the transformation matrix between the robot flange frame and the camera frame as an unknown $\mathbf{T}_f^c$, the optimization procedure can be employed. The optimization procedure minimizes the target position dissipation when the target is transformed into the global coordinate frame L_0:

$$\underset{\mathbf{T}_f^c \in \mathbb{R}^{4\times4}}{\arg\min} \sum_{i\neq j} \left\| \mathbf{T}_0^f(q_i)\mathbf{T}_f^c\mathbf{p}_{c\ i}^B - \mathbf{T}_0^f(q_j)\mathbf{T}_f^c\mathbf{p}_{c\ j}^B \right\|. \tag{1}$$

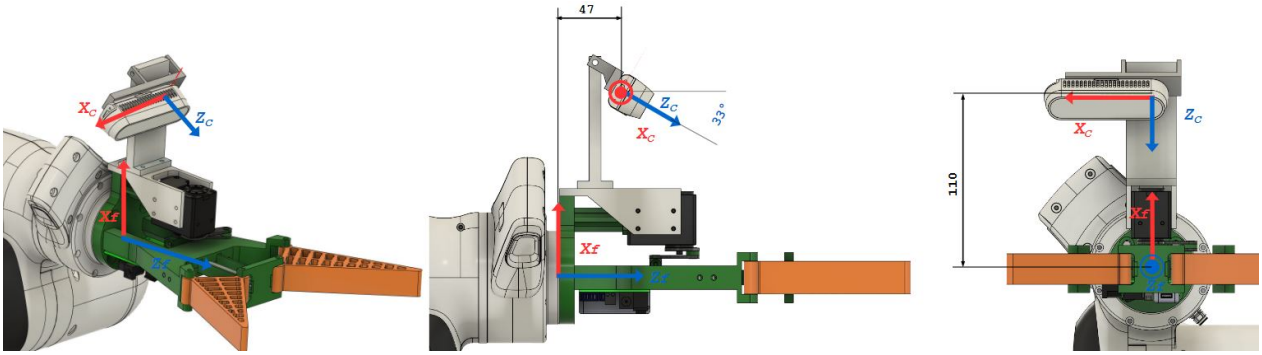

Figure 2: The position of the RGB-D camera (optical) frame L_c in the robot flange frame L_f.

a) In this laboratory exercise, you will implement the optimization procedure within the provided *sofia_perception* package and use it to calibrate the RealSense D435 depth camera. In the given package you will find the following scripts: *optimize_class.py*, *perception_server.py* and *test_perception.py*. Along with the scripts you will find *perception.bag*, *poses.txt* and *tfs.txt* files in the *resources* folder. File *poses.txt* contains the positions of the detected blob in the camera optical frame L_c for the total of 11 cases, and file *tfs.txt*

contains the corresponding poses of the robot flange frame L_f in the global coordinate frame L_0. The orientation is given in the form of quaternion ($q = [q_x\ g_y\ g_z\ g_w]^T$).

The optimization method `do_optimize_T(t1)` implemented in *optimize_class.py* takes an initial value (a guess value) of the robot flange-camera transformation as an input. An initial value (`t1`) needs to be written in vector notation as position + quaternion. Enter the initial value for the optimization procedure in the text box below. Initial values can be read from Figure 2.

Make sure to write the orientation of the camera (optical) frame L_c in the robot flange frame L_f in the form of quaternion. Read the initial values from Figure 2. You can use the existing online tools to convert from axis-angle to quaternion notation (https://www.andre-gaschler.com/rotationconverter/).

t1 = [110, 0, 47, 0.0, -0.383, 0.924, 0.0] camera -> flange
t1 = [-25.5962, 110, -39.2179, 0.0, -0.383, 0.924, 0.0] flange -> camera (used in the task)

b) Implement an objective function `objectiveFunTransformation(...)` for the given optimization (1) within the script *optimize_class.py* plot the target position dissipation after the target is transformed into the global coordinate frame L_0 by running ROS node: **$ rosrun sofia_perception optimize_class.py**. Copy the code you have written in the text box below along with the dissipation plot. The maximum difference between the transformed positions should be under 1 cm (or 0.01 m!) along each axis.

```
f = 0
    for i in range(l):
        for j in range(i+1, l):  # dist(i, j) == dist(j, i)

            transformed_pos_i = np.matmul(Teef[i], Tx)
            transformed_pos_j = np.matmul(Teef[j], Tx)

            diff = np.matmul(transformed_pos_i, data_pos[i].T) - np.matmul(transformed_pos_j,
data_pos[j].T)

            f += np.linalg.norm(diff)
    return f
```

Replace the placeholder transformation from the panda_link8 to the panda_camera frame in the provided launch file (*sofia_perception.launch*) with the optimized transformation. Copy the transformation to the text box below as well.

Optimized transform: [0.06557232 0.04289257 -0.00638682 -0.00144021
-0.24910939 0.96846931 0.00310212]

Also, leave a comment in the following text box referring to the dissipation plot you have created.

The maximum difference between the transformed positions is around 0.1 on the x-axis, 0.03 on the y-axis, and 0.07 on the z-axis. The optimization can be considered successful. The blobs are transformed as intended on the dissipation plot.

Task 2: **Perception - Pick and Place (2 points)**

a) Implement a custom ROS service server in *perception_server.py*. The proposed service is defined in `DetectObject.srv` file within the provided package. The service takes in a single point cloud message in the robot base frame (an example point cloud can be found in *perception.bag* file) and returns the centroid of an object located on the wooden board (Figure 3), also in the robot base frame. In *perception.bag* you will find the point cloud message recorded with the RealSense camera and transformed to the robot base frame. The point cloud shows an object placed on a wooden board, as it will be during the presentation of the laboratory exercise. However, the object might not be placed on the same position and it can vary in size. Objects can be placed only on red fields (Figure 4) and only one object will be placed on the board during the presentation of the laboratory exercise. Copy the written code in the text box below.

```
def detect_object(self, req):
        pc_data = ros_numpy.point_cloud2.pointcloud2_to_array(req.input_pc)
        self.x=0.169
        self.y=0.0
        self.z=0.32

        mask = (pc_data['x'] > self.x) & (pc_data['x'] < self.x + 0.081) & (pc_data['y'] > self.y) & (pc_data
['y'] < self.y + 0.6) & (~np.isnan(pc_data['x'])) & (~np.isnan(pc_data['y'])) & (~np.isnan(pc_data['z']))
        filtered_points = pc_data[mask]

        if len(filtered_points) == 0:
            rospy.logerr("No points left after filtering")
            raise rospy.ServiceException("No points left after filtering")

        x_sum = np.sum(filtered_points['x'])
```

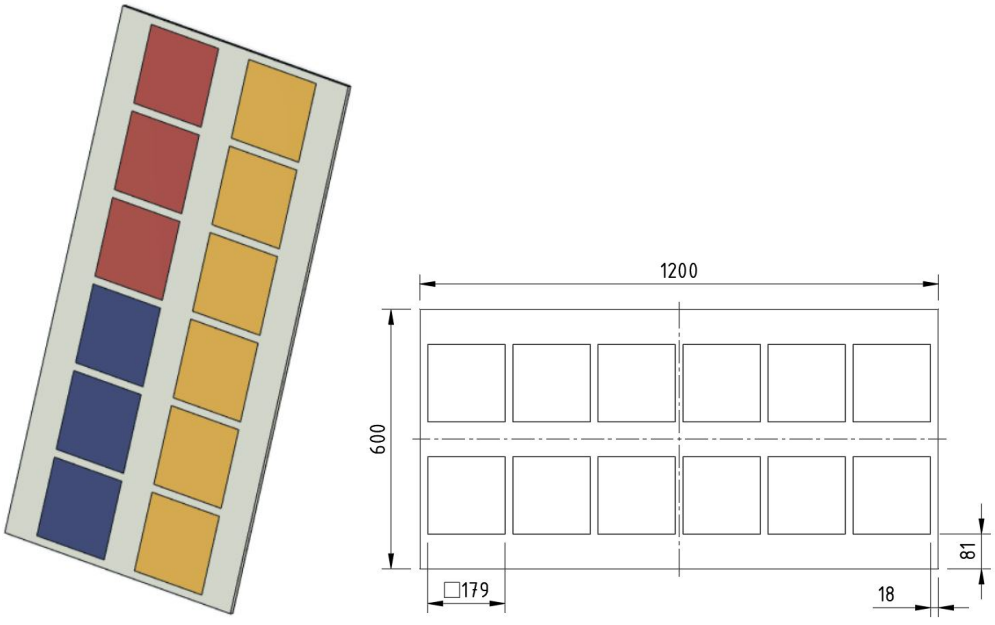

Figure 3: Layout of the environment on which the objects will be placed. It consists of 12 fields of the same size.

b) After writing and implementing the ROS service, test it using the *test_perception.py* node found in the given package on the point cloud from *perception.bag* by running: **$ rosrun sofia_perception test_perception.py**. Copy the obtained centroid into the text box below.

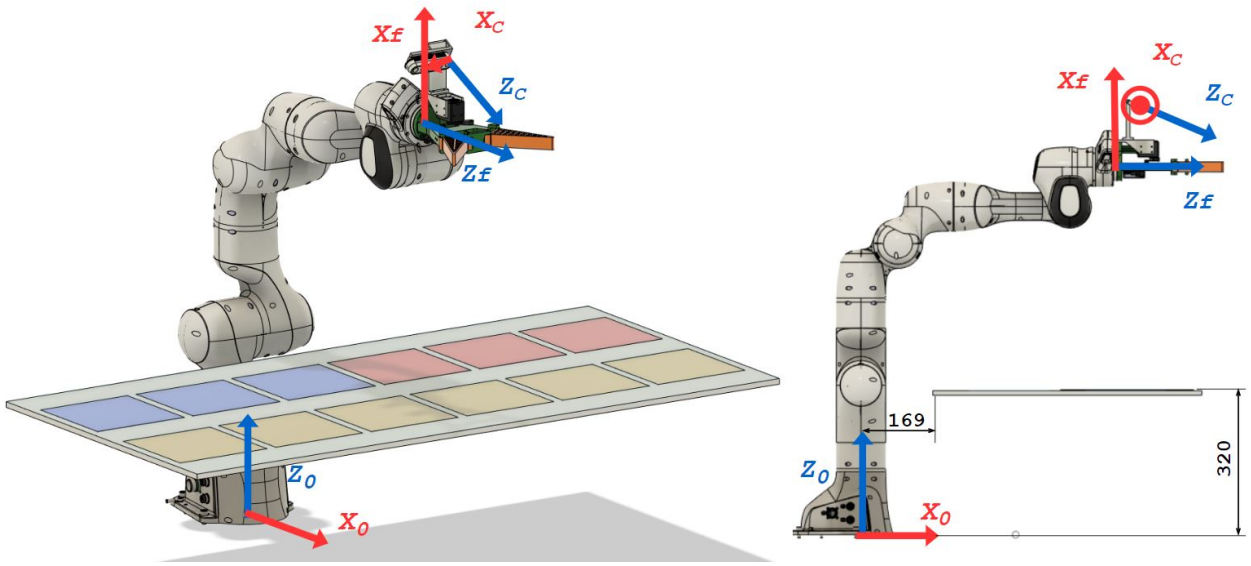

Figure 4: Robot setup consisting of Franka Emika Panda robot arm and SofIA gripper in a given environment. The wooden board (Figure 3) is symmetrically arranged with respect to the x_0 axis. **Objects that need to be grasped can be exclusively placed on RED fields. You will have only 1 object on the field during the solution presentation scenario.**

```
object_centroid:
  x: 0.21372009780496823
  y: 0.27217419402098675
  z: 0.5544274274746712
```

Exercise submission

Create a zip archive containing this pdf file with the completed answers and the updated *sofia_perception* package. Upload the archive to Moodle by the deadline.