Laboratory exercise 5

SofIA: Calibration and Perception

Name: Dominik Sekula **JMBAG:** 0036519524

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.

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. This structure is able to hold larger loads without losing compliance.

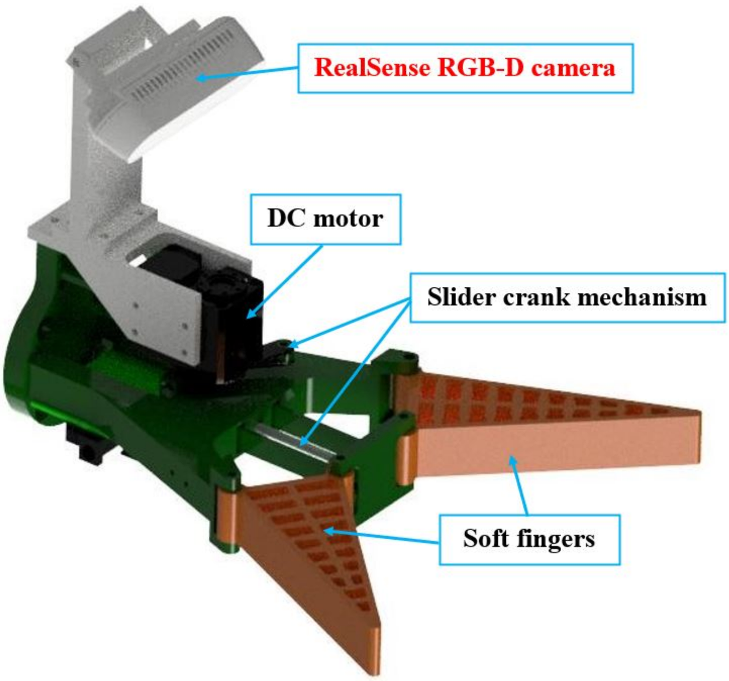

Figure 1: Main components of SofIA. It consists of four main elements: Intel RealSense D435 depth camera (RGB-D camera), Dynamixel MX-64 DC motor, slider crank mechanism and soft fingers.

The hand 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 translation thus achieved actuates 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 modelling of the manipulated environment. In order to achieve accurate and desired camera behaviour, 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. Knowing the transformation from robot flange frame to camera frame, positions of objects detected in camera coordinate frame can be transformed to robot base frame or robot end effector frame and used as a reference for the manipulator. After positioning the manipulator, the gripper closes and grasped object can be moved from location A to location B.

Assignments

Task 1: Calibration (2 points)

In order to properly use the information from the camera for the positioning of the end effector, the robot-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 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). Blob is fixed during the recording procedure and a robot manipulator is moved in order 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 reference frame 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 along with the joint configuration $\mathbf{q} \in \mathbb{R}^7$ in which the frame was recorded. Joint values enable us to calculate the transform $\mathbf{T}_0^f(\mathbf{q}_i)$ between the robot base L_0 and the flange L_f, where the camera is mounted. Finally, treating the transformation between the flange and the camera 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:

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

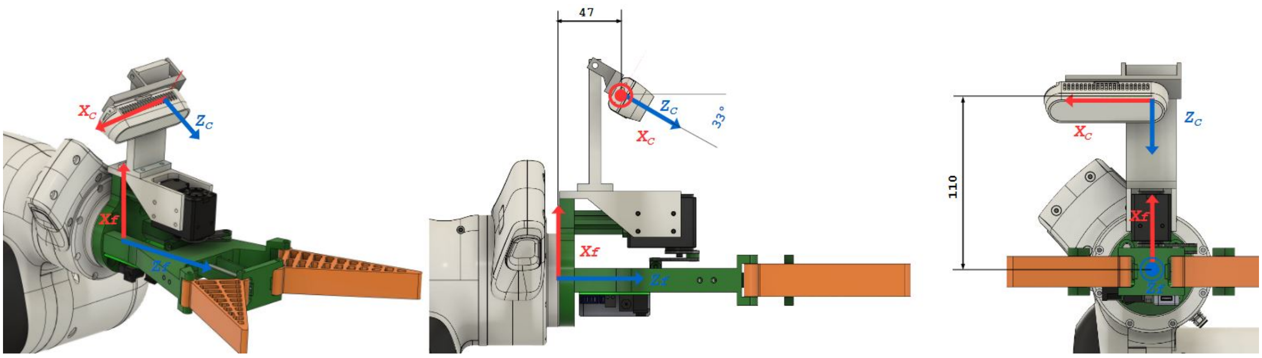

Figure 2: The position of an RGB-D camera optical frame L_c in the robot flange frame L_f.

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 flange frame L_f in the global base 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 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 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 the Figure 2. You can use the existing online tools to convert from axis-angle to quaternion notation (https://www.andre-gaschler.com/rotationconverter/).

Quaternion = [-0.2008292, -0.2008292, 0.6779879, 0.6779879]

b) Implement an objective function `objectiveFunTransformation(...)` for the given optimization (1) within the script *optimize_class.py* and 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 $1cm$ along each axis.

```
f = 0.0
    for i in range(l):
        for j in range(l):
            if i != j:

                a = Teef[i,:,:] * Tx * np.transpose(data_pos[i,:])
                b = Teef[j,:,:] * Tx * np.transpose(data_pos[j,:])

                f += np.linalg.norm(a-b)
return f

[[ 0.39091987 -0.21107367  0.3200223   0.        ]
 [ 0.39027728 -0.21035672  0.32428901  0.        ]
 [ 0.39002987 -0.21175259  0.32658383  0.        ]
 [ 0.39157633 -0.21080349  0.32902237  0.        ]
 [ 0.38937704 -0.21109619  0.32926869  0.        ]
 [ 0.38974995 -0.21364569  0.32677807  0.        ]
 [ 0.38896231 -0.2116962   0.32636367  0.        ]
 [ 0.38898187 -0.21073976  0.32740977  0.        ]
```

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.

[0.11489888 -0.00425727 0.04696682 -0.20219018 -0.21587602 0.66919297
 0.6816872]

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

U komentarima pored su rješenja povezanih textboxeva

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. Point cloud shows object placed on a wooden board, as it will be during the presentation of the laboratory exercise. However, 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.

U komentarima pored su rješenja povezanih textboxeva

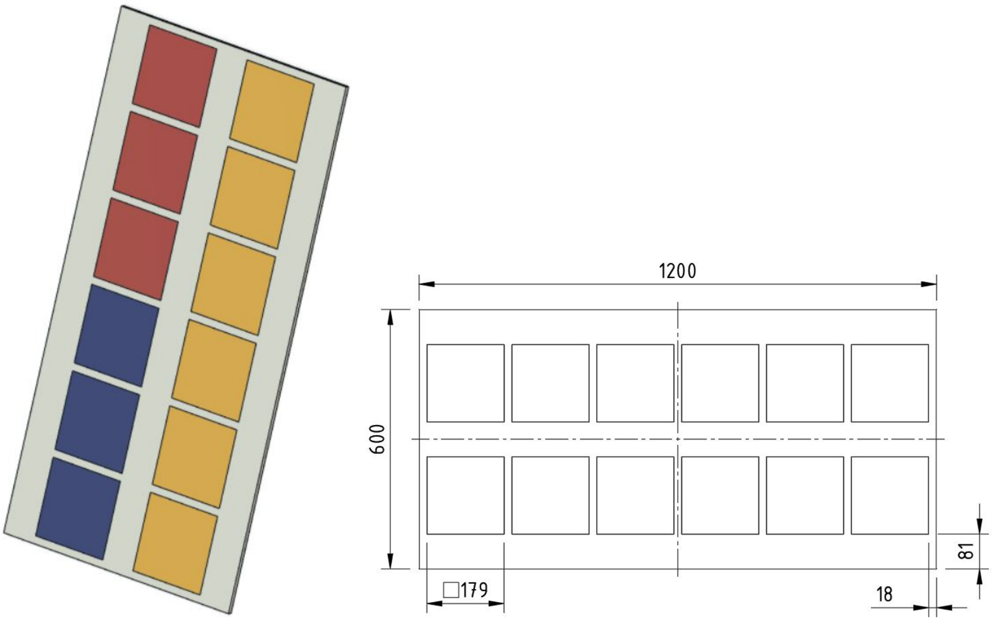

Figure 3: Layout of the environment on which the objects will be placed. It consists of 12 fields of the same size.

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.

Centroid:
x: 0.2997716213446868
y: 0.2666314483795057
z: 0.3564950424292358

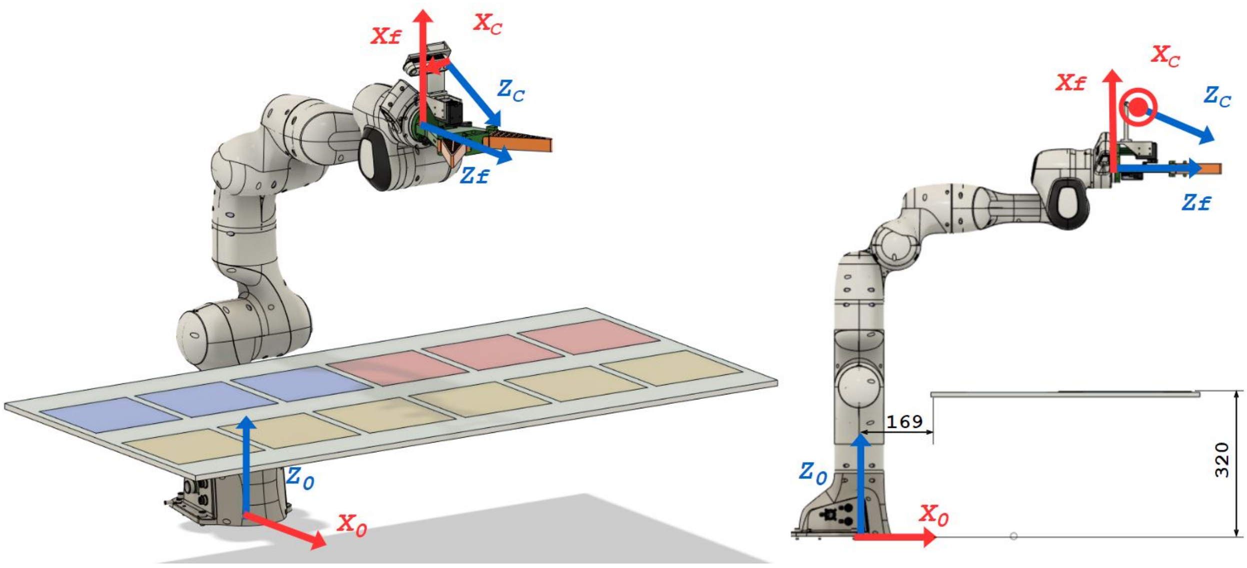

Figure 4: Robot setup consisting of Franka Emika Panda robot arm and SofIA gripper in a given environment. The wooden board (Figure 3) is simetrically arranged with respect to the X_0 axis. **Objects that need to be grasped can be exclusively placed on RED fields. You will have only 1 object on the field during the solution presentation scenario.**

Exercise submission

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