Exercise 4: Face detection and transform frames

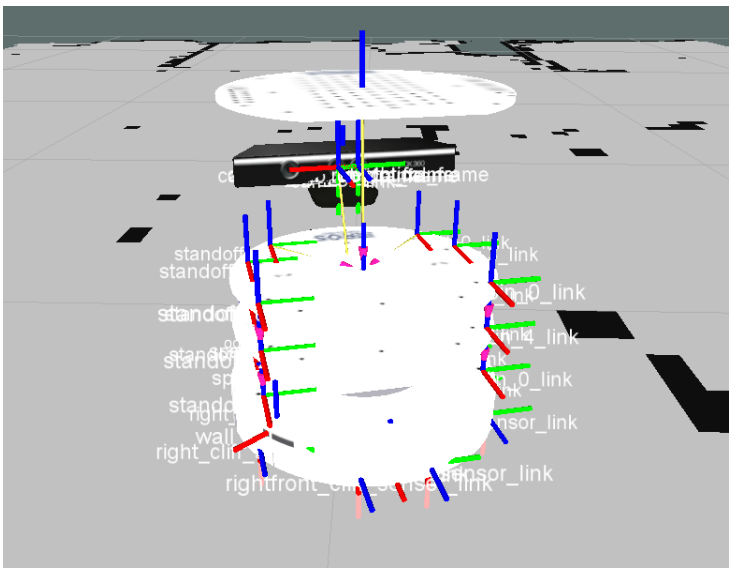
Development of Intelligent Systems

2021

In this exercise you will get introduced to some ways in which you can do face detection as well as working with transform frames in ROS.

1 Transform frames - tf2

One of the cornerstones of the ROS framework is the *tf2* (the next version of *tf*) library which handles chains of coordinate transforms. The main idea of coordinate transform chains is that individual components (camera, wheel, arm, etc.) of the robot are described using their own coordinate spaces which are hierarchically linked between themselves.



1.1 Playing with coordinate transforms

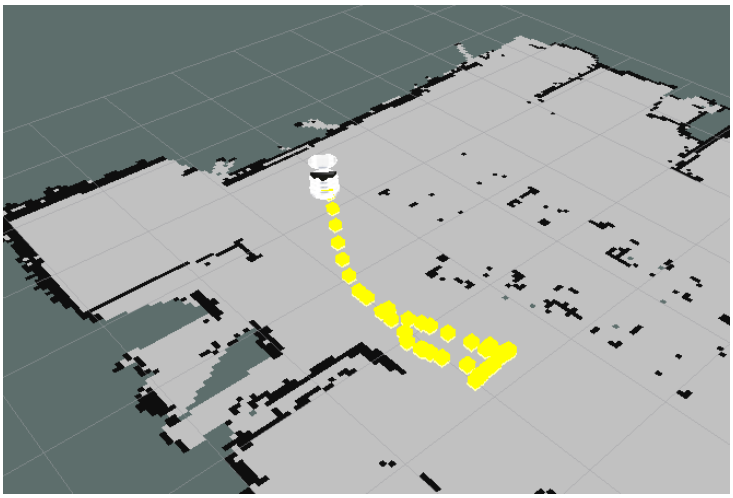
Read and complete the following tutorials that introduce one of the core ROS libraries, the *tf2* library, that handles coordinate frames and their translations. The tutorials use a simulated turtle in a 2D space so no real robot is required at this point (which makes the development much easier).

- The turtlesim package (which we have used several times)
- Main page for *tf2*
- Demo introduction to *tf2*

- Python
  - Getting transformations: A simple transform listener, A listener with time, A listener with time travel.
  - Adding a transform frame to the transform frame tree: Adding a frame, Static frame broadcaster, Dynamic frame broadcaster.
- C++
  - Getting transformations: A simple transform listener, A listener with time, A listener with time travel.
  - Adding a transform frame to the transform frame tree: Adding a frame, Static frame broadcaster, Dynamic frame broadcaster.
- Debugging *tf2*

1.2 Breadcrumb trail

One of the examples for this exercise is a *breadcrumbs* node that listens for current position of the robot and displays a new marker in map coordinate system at the location of the robot at a given interval. It uses the `TransformListener` class to obtain a transformation from the *base_link* to the *map* coordinate system. It then accumulates these positions over time and publishes them for visualization in Rviz as Markers. Get familiar with markers, since you will need to use them to successfully complete the tasks in this course. In the comments of the `breadcrumbs` node you also have many different ways of using the `TransformListener` class.



☞ This assignment will be very useful later on when you will frequently have to transform points from various detections from camera coordinate space to the map because the camera moves around with the robot so the position of the detection may change all the time.

2 Face detection

Additionally in the provided package for this exercise you have two minimalistic Python implementation of a face detector and localizer. The first face detector is based on the dlib library and to get it working you need to install the dlib library (pip3 install dlib). The second face detector is a deep neural network that we use through the DNN (Deep Neural Network) module in OpenCV. More detailed discussion the these modules will be available in the video tutorial for this exercise.

3 Message filtering

Sometimes, we want to carefully handle the order in which we process messages. For example, when we want to localize a detected face in the map, we need to make sure that the RGB image we used for detecting the face, the depth image we used for measuring the distance to the face, and the position of the robot we used for converting the relative position to a global position, were all acquired at the same time. For applications like this, you can use the `message_filters` package, which containes utility functions for dealing with timestamps of ROS mesages. You can read the documentation on the package here.

4 Homework

- You should create a program that drives the robot around the polygon. The robot should detect all the faces in the polygon and place a Rviz marker where each face was detected. The robot should place only one marker for each face that was detected!