ELEC4010K
Machine Learning for Robotic Perception

Project Report

2019

Dominic Landolf

Team member: Tim Engelmann

Structure

1. Introduction
    1.1. Notes on Demo Video
    1.2. Teamwork distribution
2. Tasks
    2.1. 2D grid Map
    2.2. Robot Control with Keyboard
    2.3. Image Recognition and Localization
        2.3.1. image_converter
        2.3.2. face_recognition
        2.3.3. marker_setter_laser
        2.3.4. marker_setter_map
    2.4. Visual Servoing
        2.4.1. ball_recognition
        2.4.2. ball_tracking
    2.5. Launch File
3. Personal Experience and Learning Outcomes

1. Introduction

A mobile robot equipped with a Lidar sensor and a visual camera is located in a provided simulation environment. The mobile robot can be controlled with the keyboard to drive around. While it is moving, it builds a 2D grid map of the environment using its laserscan data and this grid map is shown in rviz. The 5 images of different people in the environment are detected and the people are recognized. When an image is detected, its location is estimated and markers are shown in rviz to indicate the location in the map as well as the name of the recognized person. In one room of the environment, there is a moving ball, which the robot is able to follow automatically.
All these tasks are demonstrated in a video, to which some remarks are given in the next subsection. The tasks are explained in more detail in section 2. In section 3 each of the group members writes individually about the personal experience and the learning outcomes of this project. Please note that besides that last section the majority of the report was written together and is therefore the same for both team members.

1.1. Notes on Demo Video

In the beginning of the video, a roscore is runned and VREP is started. Then we load the environment and already start the simulation. Using a launch file, all the necessary nodes are runned to fulfill the tasks. With the keyboard, we drive the robot around, while the generated map is shown in rviz. The faces on the walls are detected by a face detector and the person on it is then recognized by a face recognizer (the debug image of the labeling of the face recognizer is shown in the end when Orlando Bloom is recognized). Markers at the correct location on the map and with the correct name of the person are shown in rviz. But in order for the marker to be shown, the distance between the robot and the image needs to be small enough (4.5m). This additional constraint of the maximal distance was added to ensure a good estimate of the location of the images.
After all the images on the wall are correctly marked and labeled on the map, we drive the robot to the room with the moving ball. As soon as the ball is in the current field of view of the camera, we kill all the nodes from before (since they are not needed anymore). With a different launch file we now launch the necessary nodes for tracking and following the ball. The robot is able to nicely follow the ball around, which is shown in the video until we stop the simulation.

1.2. Teamwork distribution

To achieve all the given tasks we decided to work together most of the time. However, as soon as we managed to setup the ROS-VREP environment on both our PCs, we were able to work and test ROS nodes individually. So when trying to find a solution to a given challenge we sometimes tried two different approaches simultaneously and would carry on as soon as one had found a good solution. Then we made sure that both our programs were up to date and proceeded to tackle the next task.
However, given that way of work distribution, it is hard to say, who wrote which node, since we collaborated on all of them and often even wrote the code together.

2. Tasks

In order to fulfill all tasks we decided to split the logic in several ROS nodes. Furthermore, we divided the demo in two parts. Demo Part I includes the keyboard control, mapping and image recognition. Demo Part II focuses on the ball recognition and following. For both Demo parts, the full ROS-Graph can be found in Figure 1 and 2 respectively. On the first look both Graphs look quite complicated, so in the following sections each part will be explained Task by Task.

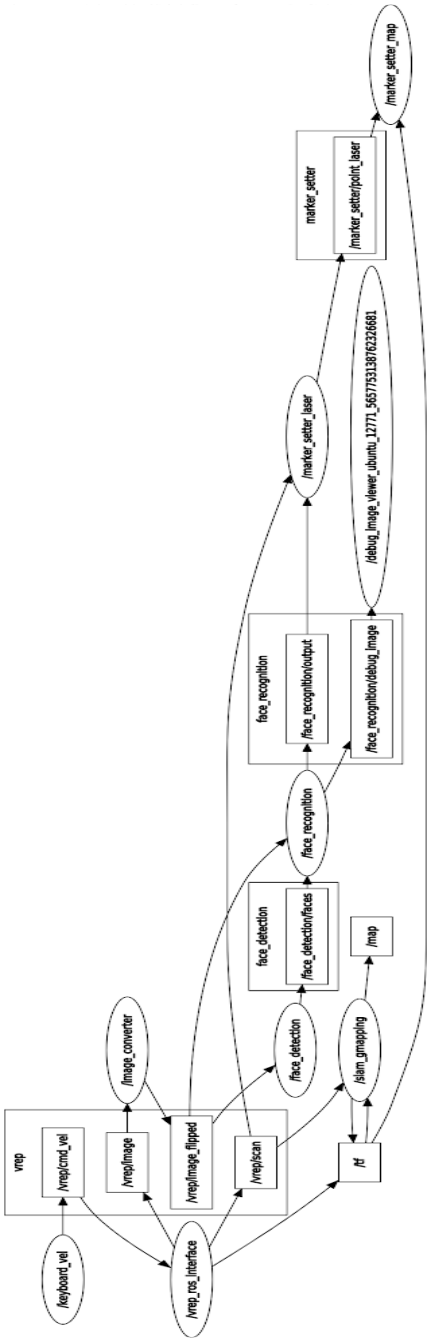

Figure 1: ROS graph Demo Part I

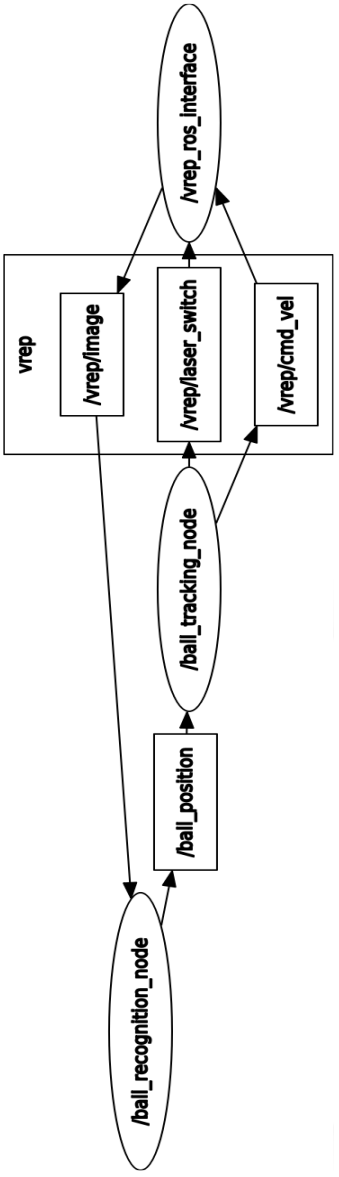

Figure 2: ROS graph Demo Part II

2.1. 2D grid Map

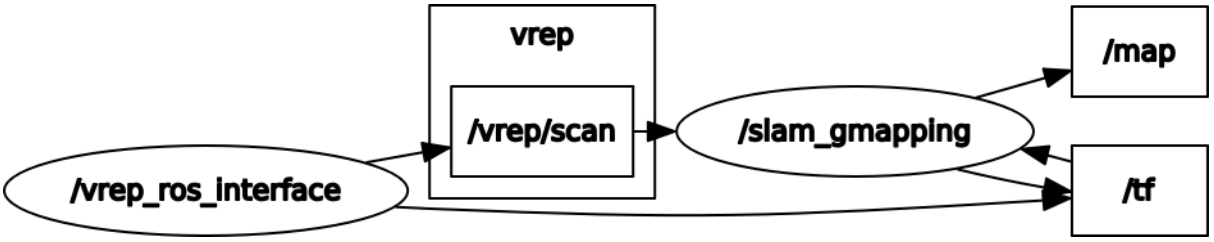

Figure 3: ROS graph task 2D grid map

For the task of building a 2D grid map, the nodes /vrep_ros_interface as well as /slam_gmapping are needed. The /vrep_ros_interface node publishes the laserscan data to the topic /vrep/scan. It also publishes transforms between the different frames of the robot: base_link, laser_link and camera_link to the topic /tf.
The node /slam_gmapping subscribes to the laserscan data and the transforms of /tf. Using these inputs it performs simultaneous localization and mapping (SLAM), so it publishes a map (to the topic /map) as well as an estimate of the robot's location and orientation. The map that is generated from the initial robot position is shown in figure 4. The robot's location and orientation is published as a transform from the map frame to the base_link frame of the robot. This transform is also published to the topic /tf. The relationship between the transforms is shown in figure 5.
An overview of the nodes used for this task is shown in figure 3. What is not shown is that rviz subscribes to the /map as well as the /tf to be able to show the map and the location of the robot in the map.

In order to fulfill this task, we had to remap the topic name for the laserscan data used by the /slam_gmapping node to /vrep/scan. We also had to change the frame name parameters of the gmapping so that the odom_frame as well as the base_frame are called base_link (because our odometry frame coincides with the base_link frame).

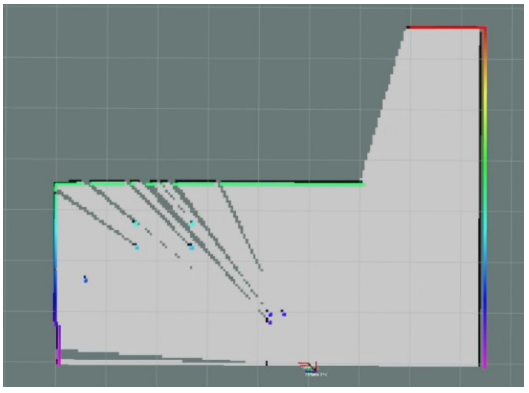

Figure 4: Generated map at launch

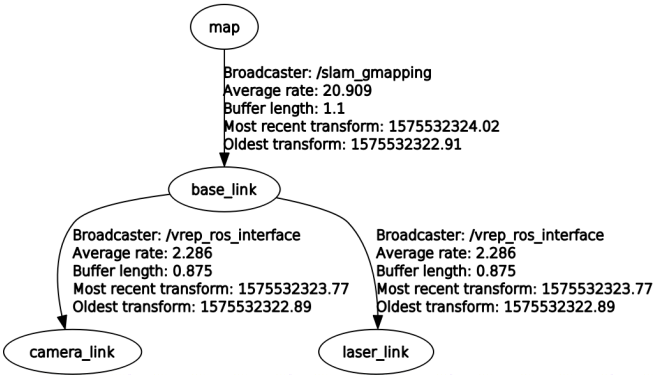

Figure 5: tf tree of the frame transforms

2.2. Robot Control with Keyboard

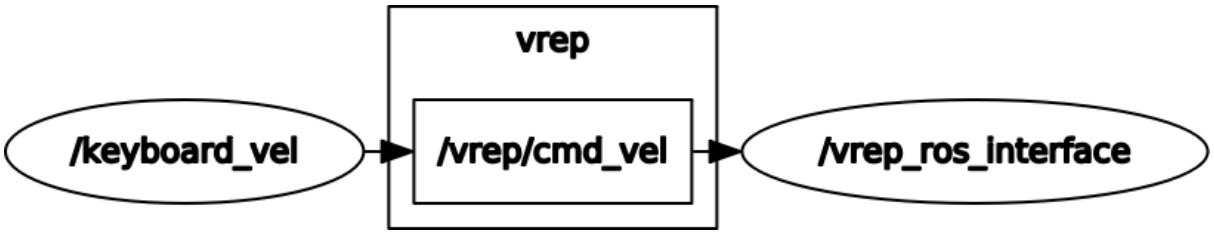

Figure 6: ROS graph task keyboard control

The robot in the vrep environment should be controlled by the keyboard. Therefore, a node /keyboard_vel is used to transform keyboard inputs into linear and angular velocity commands, which are published to the topic /vrep/cmd_vel. The /vrep_ros_interface subscribes to these velocity commands and sets the velocities of the robot in the simulation accordingly.

The robot can be moved forwards and backwards by pressing the buttons “w” and “s” respectively. This sets the linear velocity to a predefined positive or negative value. In order to turn the robot to the left or to the right, the buttons “a” or “d” can be pressed. This sets the angular velocity around the z-axis of the robot to a predefined positive or negative value. By pressing “q” or “e”, the predefined values for both linear and angular velocities are increased/decreased by 10%. For accurate results during mapping, these velocities should not be increased.

2.3. Image Recognition and Localization

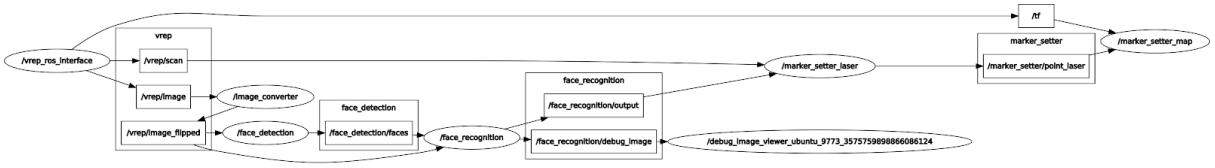

Figure 7: ROS graph task image recognition and localization

The image recognition and marking task was surely one of the most challenging parts of the project. As it can be seen in Figure 7 the process involves several ROS nodes, which will be further explained in the following sections.

2.3.1. image_converter

The vrep simulation publishes a topic called /vrep/image which includes the camera image of the robot. However, that image is left-right reversed, so before usage it had to be inverted. By writing a simple image_converter node that subscribes to the /vrep/image and publishes /vrep/image_flipped, that issue could be resolved. In the node the image is converted to an OpenCV image using CV bridge, is flipped and afterwards converted back to a ROS image before published as sensor_msgs/Image again.

2.3.2. face_recognition

The next challenge was to recognize if a face occurs in the current camera view or not. At first we tried using the face_detector node from the “people” stack, but encountered many problems in its application. The main issue was that we did not have a depth_registered image, that the face_detector could subscribe to.
So we proceeded to try a different package. We decided to use the face_detection of the OpenCV_apps library. Later on we became eager to master the extra task of recognizing the faces as well, so we switched to using the face_recogniton node of the same library. The transition could be done fairly easily, since both nodes publish the same topics and actually the face_recogniton node launches the face_detection anyway.
So finally, the face_detection node subscribes to the /vrep/image_flipped and publishes an array of detected faces to the face_recogniton node. By providing training images from different angles of each of the 5 faces, a text label can be added to each face. Finally an opencv_apps/FaceArrayStamped is published.

2.3.3. marker_setter_laser

The marker_setter_laser subscribes to the opencv_apps/FaceArrayStamped topic and the sensor_msgs/LaserScan topic published by VREP. By synchronizing and combining the information of both, the goal is to determine the position of a potential marker in the Laser_link coordinate system. To prevent wrong detection, several conditions are checked.
A detection can only be valid if exactly one face is recognized. Furthermore, only detection in the upper half of the image are considered, since there are no faces on the floor. If these preconditions are fulfilled the detected position in the camera image had to be converted to an actual position relative to the robot.

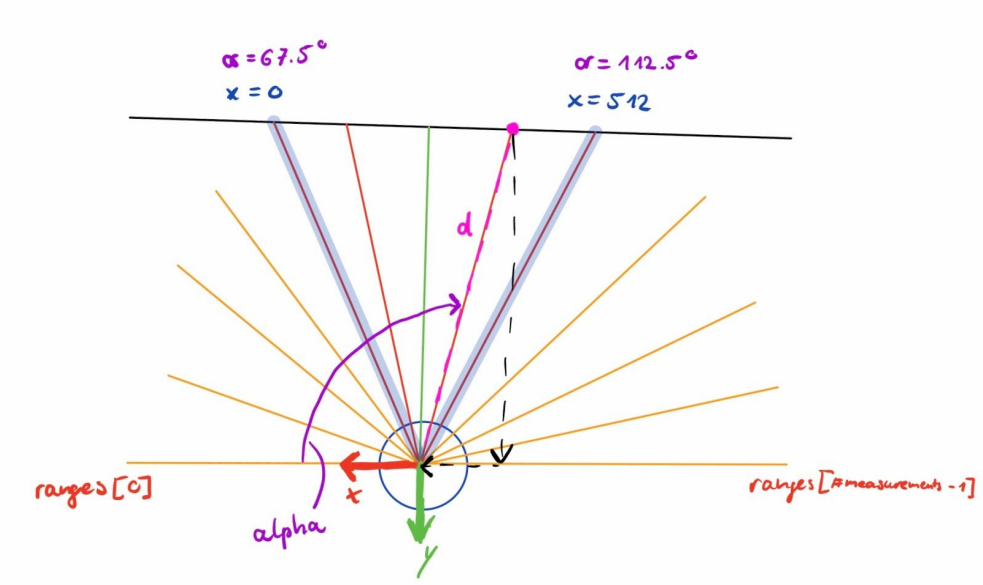

Figure 8: Computation of x and y position in laser_frame

As it can be seen in the above figure the key was to find the correct angle called *alpha* and then the right distance measurement in the ranges_array of the laser_scan. With the knowledge that a camera angle of 45° correspond to 512px and that the laser has a total coverage of 180°, alpha could be computed. Using trigonometry and by looking at the axis definition of the laser link, the corresponding x and y coordinates were found.
To ensure that images were not recognized from too far away, only points less than 4.5m and corresponding to a large enough face detection rectangle were further processed. Finally a custom message type, containing information about a geometry_msgs/PointStamped in laser_coordinates, a string with the name of the person detected and a header for message synchronisation, is published.

2.3.4. marker_setter_map

The last step was setting the marker on the map. Therefore the marker_setter_map node subscribes to the topic containing the position of the detected face in laser_coordinates and to the /tf topic published by VREP. Using these inputs, the point could be transformed to the map frame coordinate system. Before the marker is then actually set, two more conditions have to be fulfilled. Firstly, no other marker is allowed to have already been set in a radius of 2m around the position of the potential new marker. Secondly, the marker is only set, if in a queue, containing prior detections, there are already at least 3 within a radius of 0.25m from that position. This ensures that one picture is only marked once and that the position is accurate.
If so, a sphere marker and a text marker are added to the marker_array, containing all markers to be published. The sphere shows the actual position of the detected face, the text states which person was recognized.
Please note that by defining namespaces it is ensured that only one pair of markers can be set for a specific person.

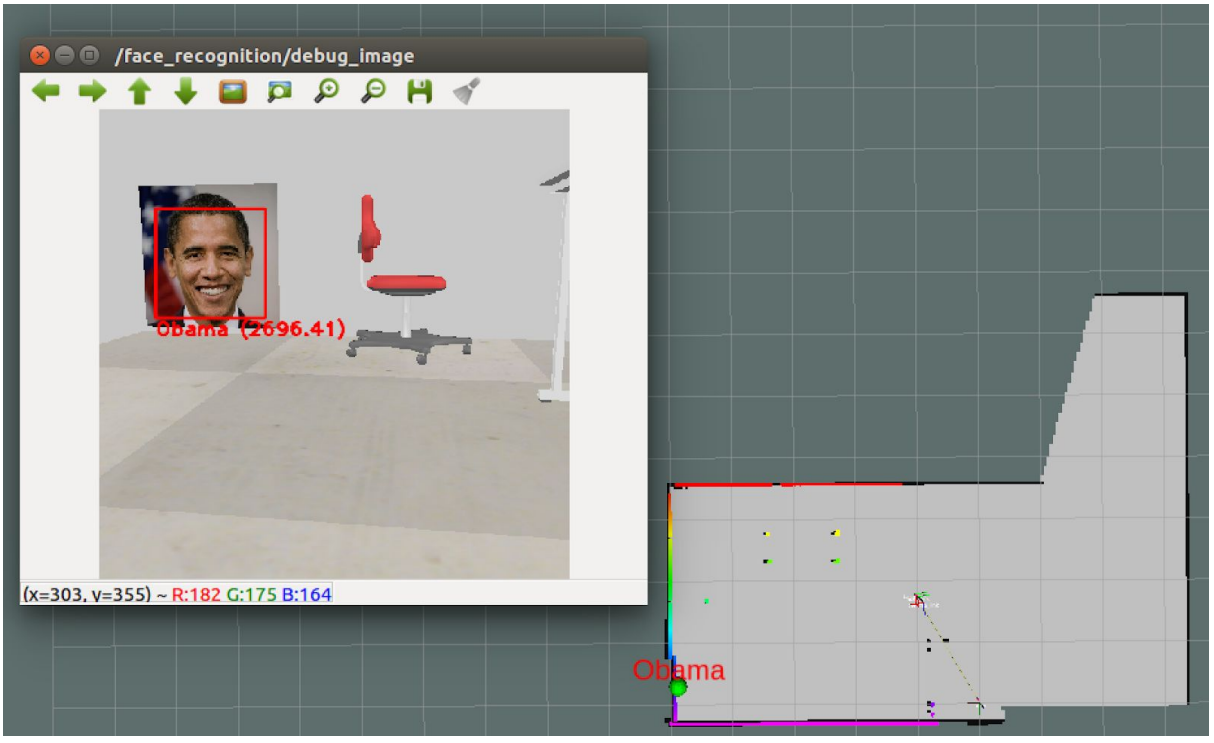

Figure 9: face_recogniton and marker_setter_map

2.4. Visual Servoing

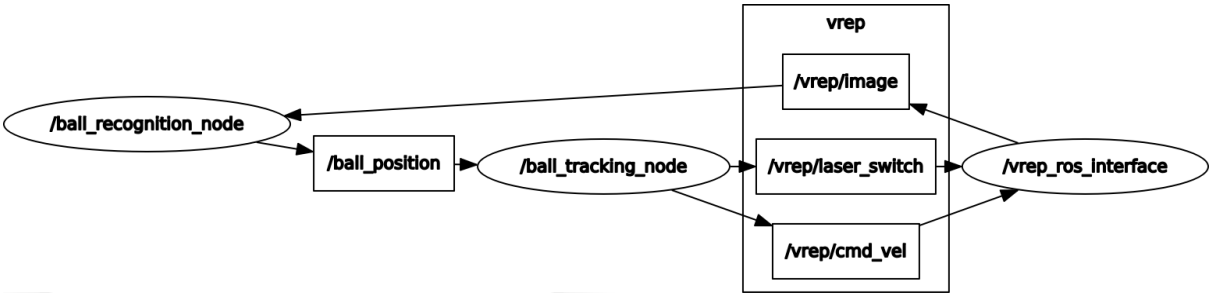

Figure 10: ROS graph task visual servoing

The robot should be able to follow the yellow ball that is moving around in the environment. To fulfill this task, we created a package called ball_recognition with two nodes: ball_recognition_node and ball_tracking_node. The ball_recognition_node is responsible for detecting and localizing the ball in the camera image and the ball_tracking_node computes the command velocities needed to follow the ball. In the following, the two nodes are explained in more detail.

2.4.1. ball_recognition

The /ball_recognition_node subscribes to /vrep/image and flips the image along the horizontal direction. This is done directly inside this node and not via the /image_converter node since it is only one additional line in the code to flip it.

In order to detect the ball, we filter out colors we are not interested in and only keep the yellow colors. For better performance we converted the image to HSV color space. To make sure that the whole ball is visible, we chose a wide range of values that are not filtered out. This wide range is good enough since most of the colors in the room are gray. In the resulting filtered image only the ball is visible (see figure 11). Some other colors are also not filtered out entirely, but since the performance of the ball detector is accurate enough for this environment, the color filter doesn't need to be improved. For debugging purposes the processed, color filtered image is also published to the topic /image_processed.

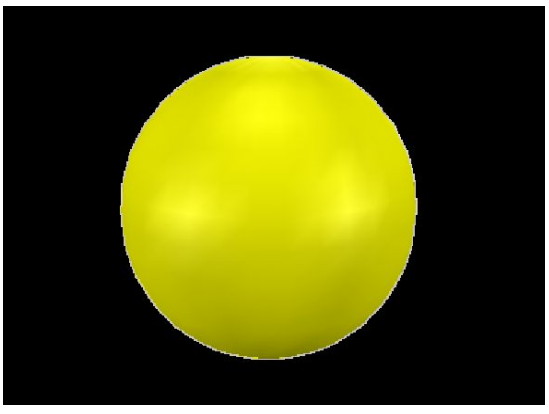

Figure 11: Color filtered image of the ball.

The flipped and color filtered image is then converted to a grayscale image and smoothed with a gaussian filter to enhance the performance of the detector. Using Hough circles, the coordinates of the center of the ball in the image and its radius can be found. This is done very reliably, since there is nothing else visible in the processed image except for the ball. As a last step, the relative position of the center of the ball to the center of the image is calculated and published to the topic /ball_position. The radius of the ball in the image is also published in /ball_position.

2.4.2. ball_tracking

The /ball_tracking_node subscribes to the topic /ball_position in which the error of the ball position that needs to be minimized by the tracking controller is stored. Two simple p-controllers are used to find the linear and angular velocities that need to be applied in order to reduce this error. The goal is to keep the ball roughly in the center of the image and at the same distance (so the radius of the ball should stay the same).

If the center of the ball is in horizontal direction too far away from the center of the image, the angular velocity is set by the angular p-gain times the negative error (ball position with respect to the center of the image). So the robot turns until the ball is roughly in the center of the image again.
If the ball is roughly in the center of the image, the linear velocity is calculated depending on the desired radius of the ball. If the radius of the ball in the image is smaller than the desired radius, a positive linear velocity is applied to drive closer to the ball. If the radius of the ball is too big, a negative linear velocity is chosen. This velocity increases if the error of the ball size is bigger. We defined the maximal linear velocity of the robot to be 1.5m/s, otherwise the motion of the robot is not easily controllable because it would move too fast.

The calculated velocities are then published to the topic /vrep/cmd_vel, which is used to set the velocities in the vrep simulation. This node also turns off the Lidar by publishing “false” to the /vrep/laser_switch topic, because the laserscan data is not needed for this task.

2.5. Launch File

There are separate launch files for the two parts of the Demo. In the first file project_launch_all.launch all required and described nodes for Demo Part I are launched. In the second file project_launch_ball.launch the nodes for Demo Part II are included. Please note that it is better to kill the nodes from Demo Part I before proceeding to Part II. Also please refer to the README.txt file for further information on installation and correct launch procedure.

3. Personal Experience and Learning Outcomes

As I used ROS before in a small project, I was familiar with the concept of nodes and topics. However, I never had to write my own node or package before, so I am glad that I learned it during this project. We wrote our own publishers, subscribers and created our own packages. We also created nodes which both publish as well as subscribe and even subscribe to multiple topics at the same time. We also wrote a small message type by ourselves consisting of a geometry_msgs/PointStamped and a string to store the position of a detected picture on the wall as well as the name of the person in one message.

I also noticed that it can be very complicated and time consuming to integrate existing packages into our own project, because one needs to understand what that package and the nodes are doing. Furthermore, the message types, the topic names and a lot more have to match or need to be adjusted.

We encountered a lot of problems and we sometimes had to try many different approaches on how to solve a task, especially for the image recognition and localization task. Sometimes even a very small bug took us a long time to find. For example, the markers on the map where always set at the wrong position, so we tried to figure out what was wrong with our calculation of the angle at which the picture was seen. In the end it turned out that the calculations were correct, but there was a problem with the data type (int instead of float). But we always motivated each other and finally managed to solve all the tasks including an additional task very well, so the effort was worth it.

Overall, it was a time consuming, but educational project. We were very interested in this topic and therefore motivated to fulfill an additional task of recognizing the different faces on the wall. We worked together as a team and both of us learned a lot during this project.