Danmarks Tekniske Universitet

DTU

Building a map using SLAM on a Raspberry Pi

Author

Magnus Erler - s174820

December 22, 2022

Contents

1 Introduction

The aim of this project is to build a cheap self-aware and depth-sensing mobile robot using the Raspberry Pi hardware platform.

Performing **S**imultaneous **L**ocalization **A**nd **M**apping, or simply SLAM on a Raspberry Pi is challenging because different software packages have to be used for, e.g. to operate the camera. However, software packages and libraries might not work together in different versions and therefore extensive consultation of release notes and different forums is crucial. Deciding on the **O**peration **S**ystem (OS) for running the SLAM process is one of the important building blocks of the whole project as a sub-optimal choice might prevent some operations to be performed satisfactorily.

Having worked with SLAM before, the task did not seem impossible at first glance. However, using a low-powered board like a Raspberry Pi is challenging as storage space and performance have to be considered. Performance comparisons on another machine are therefore helpful and have been used for verification and testing the robot design.

1.1 Simultaneous Localisation and Mapping (SLAM)

SLAM is a process of mapping and localizing yourself, in unknown surroundings. The algorithm is amongst others used by self-driving cars, **U**ncrewed **A**erial **V**ehicles (UAVs), **A**utonomous **U**nderwater **V**ehicles (AUVs), and robot vacuum cleaners. In most of these cases, SLAM needs to run on a small single-board device, like a Raspberry Pi. The goal of this project is to investigate optimal conditions for running ORB-SLAM3 [Campos et al., 2021] – a popular algorithm used for real-time SLAM with mono, stereo, and RGB-D camera inputs – on a Raspberry Pi device.

1.2 Oriented FAST and Rotated BRIEF (ORB)

ORB is a fast robust local feature detector [Rublee et al., 2011], that can be used in computer vision tasks such as 3D reconstruction and object recognition. It is based on the **F**eatures from **A**ccelerated **S**egment **T**est (FAST) [Rosten and Drummond, 2006] keypoint detector and a modified version of the visual descriptor **B**inary **R**obust **I**ndependent **E**lementary **F**eatures (BRIEF). Its aim is to provide a fast and efficient alternative to **S**cale **I**nvariant **F**eature **T**ransform (SIFT) [Lowe, 1999] which also is a computer vision algorithm to detect, describe, and match local features in images.

2 Decisions for Designing the Robot

Building a robot requires sensors, actuators and, most importantly, a device which controls the whole system. Furthermore, efficient communication between the single elements is important and will be investigated in this project. In the following sections, each of these aspects is described and discussed, with emphasis on challenges and pitfalls.

2.1 Hardware

2.1.1 Raspberry Pi

To fulfil real-time SLAM and to communicate with the robot over a wireless connection using a low-powered system, the latest generation of Raspberry Pi (Raspberry Pi 4 Model B, [1]) is chosen. It includes a 1.5 GHz quad-core ARM Cortex-A72 CPU with 4 GB RAM, and several features like 40 GPIO pins and Camera Serial Interface (CSI). Due to the ARMv7 processor, it can run the full range of ARM GNU/Linux distributions such as Ubuntu OS. Choosing the Raspberry Pi 4 with 4 GB of RAM improves compiling and building of the software, such as OpenCV [1], needed to perform SLAM.

2.1.2 Arduino

Although the Raspberry Pi has up to 40 GPIO pins and is powerful, the communication with the motors and most of the sensors is done through the microcontroller Arduino Mega 2560 [2] in order not to overload the Raspberry Pi. In that way, the Raspberry Pi is able to send commands to the Arduino e.g. about driving a certain distance as part of SLAM.
The minimum requirements for choosing the microcontroller are as follows:

- Ports
  - 4 Digital input with interrupt
  - 6 Digital output (PWM)
  - 2 Analogue input
  - 1 I^2C (SDA and SCL)
- Storage
  - RAM: 3 kB (2837 byte)
  - Flash: 23 kB (22 674 byte)

The four digital input ports are used by the encoders mounted on the motor and are therefore needed to external interrupt the Arduino. Four of the six digital output ports are used by the motor driver to decide the rotation direction of the motors. The two other digital output ports are for a PWM-signal (**P**ulse **W**idth **M**odulation) to the motor driver, telling how fast the motors should turn. The 2 analogue input ports are used as an **A**nalog-to-**D**igital **C**onverter (ADC) with 10 bit resolution for measuring the voltage of the two

batteries - one port for each battery. Lastly, the I^2C port is used by the IMU.

The Teensy 3.5 [PJRC, 2022] was considered as an alternative to the Arduino Mega 2560. The Teensy is smaller, has a more powerful CPU and contains ADCs with resolution up to 16 bit which can improve the voltage measurement of the batteries, and has the required port. Nevertheless, it has not been chosen, since it can not communicate easily with the Raspberry Pi. For now, the Arduino Mega 2560 is sufficient for handling the load of the sensors and actuators, but using a different microcontroller should be straightforward, and would likely only require changing the port numbers.

2.1.3 Motor and motor driver

The robot is designed as a simple differential wheeled robot, i.e. with two independent motors (left and right) [3] supported by a castor wheel in front of the robot, to stabilise the robot. The motor driver L298N [4] has been chosen since it includes two motor drivers on a single board. For controlling the motors, the driver and the encoders from the motors are connected to the Arduino.

2.1.4 Battery

The robot consists of two separate batteries: One to power the motors and a second one to power the Raspberry Pi, the Arduino, and the connected sensors. A separate battery for the motors was chosen in order to guarantee that the other components are not affected by the voltage drop when driving, which may result in failures leading to insecure values. The batteries used are 2200 mAh 11.1 V LiPo batteries [5] and are expected to provide each motor with 5 hours of driving time (without load) and to power the Raspberry Pi and the Arduino for about 2 hours. The step-down voltage regulator D24V22F5 [6] is used to reduce the voltage from the batteries to suit the Raspberry Pi, and subsequently the Arduino through USB.

2.1.5 Camera

The camera has a still resolution of 8 MP, a sensor resolution of 3280 $\times$ 2464 px, and a horizontal and vertical field of view of 62.4° and 48.8°. It is connected to the Raspberry Pi 4 using the Camera Module v2 [7] through the 15-pin Camera Serial Interface (CSI) and thus directly to the GPU [Jones, 2017] of the Raspberry Pi. In this way, the image frames are processed directly in the GPU and not first distributed by the CPU.

2.1.6 Inertial Measurement Unit (IMU)

The IMU is used to help the performance of SLAM, providing information on roll, pitch and yaw. This project uses an MPU9150 [8] that includes a 9-axis MotionTracking MEMS built by a 3-axis gyroscope, a 3-axis accelerometer, a 3-axis digital compass, and a digital thermometer.

2.1.7 Display

A simple 128 × 64 px **O**rganic **L**ight-**E**mitting **D**iode (OLED) display [9] is used to show important information on the robot. This information is shown as eight lines on the display and consists of the Raspberry Pi's IP address, CPU load, memory used, disk usage, and battery voltages. There are furthermore reserved three lines that can be used for debugging. An example of this is shown in fig. 3. The OLED display was chosen as it consumes less power than other typical small displays and is therefore optimal for a mobile robot where the amount of power is important.

2.1.8 Joystick

A joystick with a wireless 2.4 GHz USB dongle [10] is used for debugging the motor activities and moving the robot manually around. This makes it possible to move the robot around with the right knob and start the SLAM operation with a button.

2.2 Software

2.2.1 Robot Operating System (ROS)

Building a robot involves many sensors and actuators from different manufacturers which differ in the used communication protocol and can lead to communication challenges. To solve this, ROS [2] comes into play. ROS is an open-source robotics middleware with sets of software libraries and tools used for communication. In 2017 ROS 2 was released, but was not chosen for this project because ROS 1 is better documented, with more examples compared to ROS 2. Furthermore, many libraries have not been updated in ROS 2. In a few years, ROS 2 might be the default choice of many robotic applications and can be compared to Python 2 and 3 in 2010, when Python 3 was not used that much. Nowadays, almost everyone uses Python 3 when starting a new project. For simplicity, ROS 1 will in the following be referred to as ROS.

2.2.2 Ubuntu 20.04 Desktop

Although ROS contains the abbreviation *OS* (for **O**perating **S**ystem), it is actually not an operating system but a "framework and set of tools that provide the functionality of an operating system" [Ademovic, 2015]. It, therefore, needs a proper OS to be installed on. In this project Ubuntu OS has been used since Raspberry Pi OS (former Raspbian OS), a dedicated OS for Raspberry Pi 4, is not suitable for ROS. In contrast, Ubuntu turned out to work almost out of the box, just like on a standard computer [Back-End, 2020].
Due to the latest ROS 1 distribution (ROS Noetic Ninjemys) not being supported by the latest release of Ubuntu (Ubuntu 22.04.1 LTS Jammy Jellyfish) [ROS, 2020], Ubuntu 20.04 LTS Focal Fossa [3] in its desktop version, was installed, to help to visualise ROS. However, due to the lack of a specific Ubuntu 20.04 desktop version for Raspberry Pi, the server version of Ubuntu 20.04 was installed, followed by the desktop environment from the terminal.

2.2.3 ORB-SLAM3

ORB-SLAM3 [4] is an open-source library for visual, visual-inertial and multi-map SLAM developed by SLAMLab, a research group at the University of Zagreb. The library was chosen because it has the ability to create real-time SLAM with only a single (mono) camera. ORB-SLAM3 is an improvement of ORB-SLAM2, which in turn is an improvement of ORB-SLAM. The main difference between ORB-SLAM3 and ORB-SLAM2 is the ability for relocalisation after lost tracking in case of poor visual information. In that way, it is able to close a map even long after the tracking has been lost [Kumar, 2021].

3 Guideline for Implementation

3.1 Assembling the hardware

Fig. 1 shows the final wiring diagram of the robot. The Arduino is connected to the Raspberry Pi through a serial connection (USB). On the right side, the Raspberry Pi along with the OLED display and a fan can be seen. The fan is mounted on top of the Raspberry Pi to cool the CPU and other hot parts of the board. Below the Raspberry Pi is one of the four USB ports used to communicate with the Arduino, while another one has the USB dongle for the joystick. Also connected to the Arduino is the IMU shown above the Arduino. On the left side of the diagram, the two motors with encoders can be seen along with the dual motor driver circuit board. The two batteries with each voltage divider can be seen in the middle. Here the upper battery is also connected to the step-down voltage regulator which powers the Raspberry Pi.

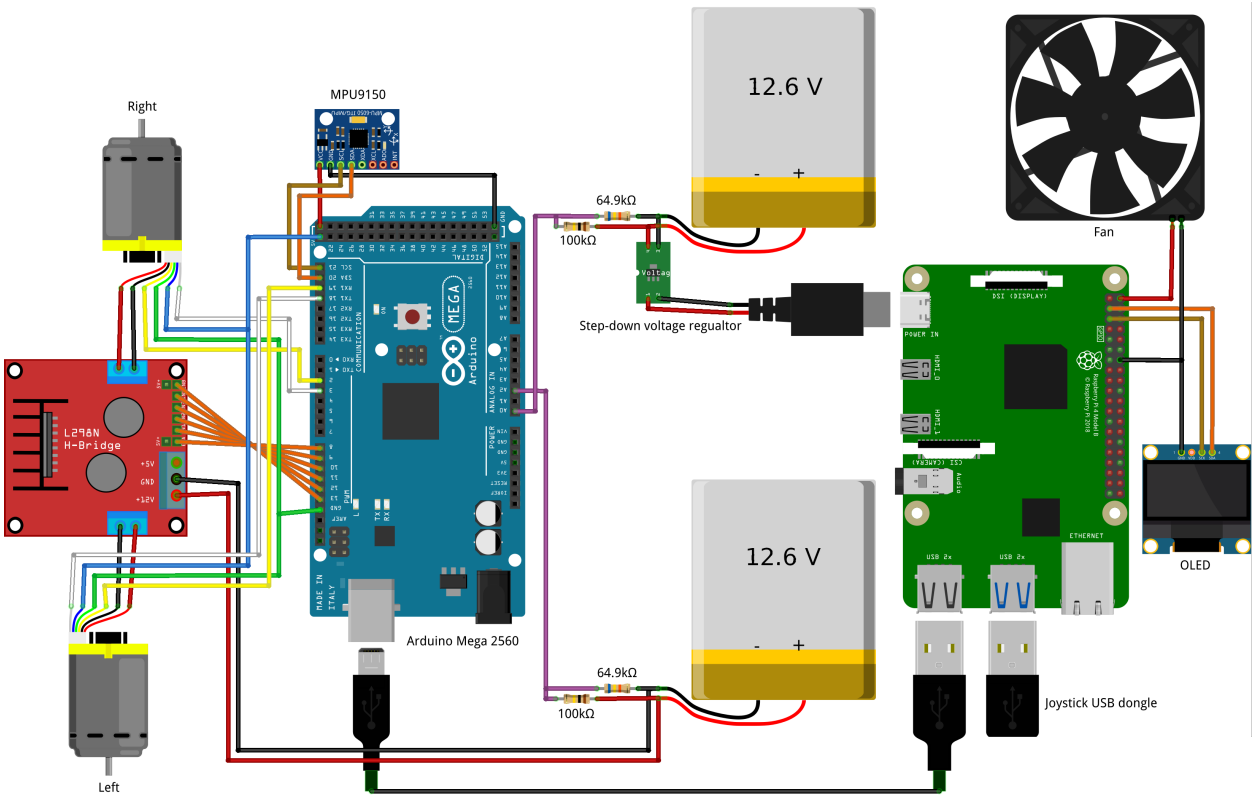

Figure 1: Wiring diagram of the robot. A larger diagram can be found in app. B.

Powering the Raspberry Pi through the USB-C port instead of the GPIO pins benefits the safety of the device. In this manner, a fuse is added and the Raspberry Pi has its own chip to check if enough voltage and most important enough current is available to the board.

For the Arduino to measure the voltage of the batteries, the voltage needs to be reduced to less than 5 V, as the ADC otherwise will shortcut. This implies that the maximum voltage of the batteries of 12.6 V (11.1 V when discharged) will be read by the Arduino as 5 V and then upscaled again to 12.6 V by software. A simple voltage divider consisting of two resistors was implemented to lower the voltage:

$$\begin{aligned} V_{out} &= V_{in} \cdot \frac{R_2}{R_1 + R_2} \\ 5\,\mathrm{V} &= 12.6\,\mathrm{V} \cdot \frac{100\,\mathrm{k\Omega}}{R_1 + 100\,\mathrm{k\Omega}} \\ R_1 &= 65.79\,\mathrm{k\Omega} \end{aligned}$$

In practice, the closest commercial resistor is chosen to be $R_1 = 64.9\,\mathrm{k\Omega}$. The edge values of the voltage divider according to the tolerance of both resistors of 1% will lower the voltage of 12.6 V to 4.90 V and 5.02 V. Upscaling this in the software will give edge values of 12.34 V and 12.65 V. It is concluded that the uncertainties will not affect the ADC of the Arduino. Both voltage dividers are therefore connected directly to each battery and the output voltage is connected to A0 and A2 at the Arduino. It was decided to physically separate the two inputs to reduce noise.

The carrosserie of the robots was borrowed by Robobot [Andersen, 2022] developed by the Technical University of Denmark - DTU and can be seen in fig. 2 with all the designed hardware.

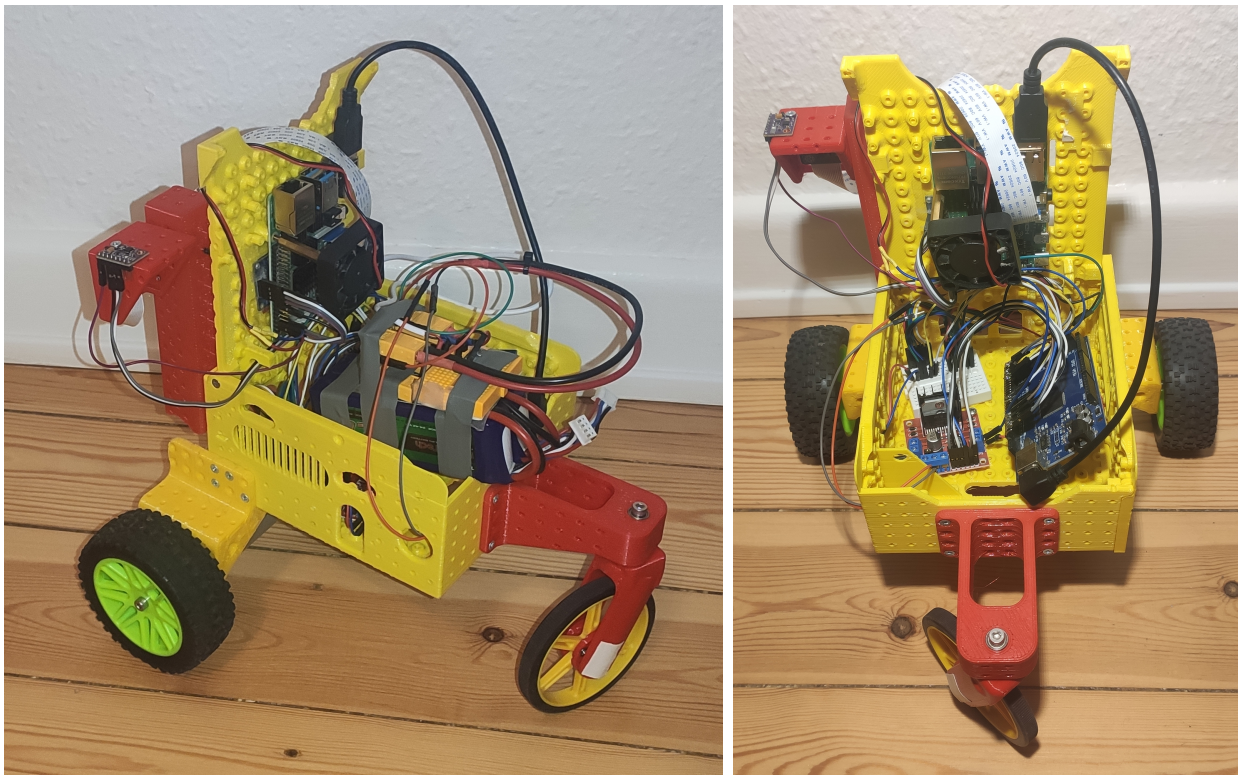

Figure 2: Robot with the carrosserie of the DTU Robobot. The right figure shows the robot without the two batteries.

The Raspberry Pi 4 can be seen in the figure vertically mounted to the lid. Next to the robot on the right side is the camera placed inside the red house, the IMU is mounted on its top. In the right image, the batteries have been removed to show the motor driver on the left side and the Arduino Mega 2560 on the right side. The wires from the sensors and to the motor driver are soldered to an array of pins which then are placed into the different female headers of the Arduino.

3.2 Software setup

The aim of the following paragraphs is to provide a guideline for setting up the software to perform SLAM on a Raspberry Pi 4.

3.2.1 Setup Ubuntu 20.04 Desktop

Start with downloading Raspberry Pi Imager on your personal computer, insert an SD card and flash it with Ubuntu 20.04 Server. Insert the SD card into the Raspberry Pi, connect a display with HDMI, log in (login: ubuntu, password: ubuntu), and change the password. Now install the desktop version (without all the bloat to have a minimalistic version) with the following:

```
sudo apt-get update && upgrade
sudo apt-get install --no-install-recommends ubuntu-desktop
```

Install and setup LightDM as a light-weight Desktop Environment/GUI [wiki.debian, 2022]:

```
sudo apt-get install lightdm
```

Then choose LightDM in the popup window with the following commands:

```
sudo systemctl start lightdm.service
sudo service ligthdm start
sudo reboot
```

After reboot, add the below lines to `/boot/firmware/usercfg.txt`.

```
hdmi_drive=2
hdmi_safe=1
dtoverlay=vc4-fkms-v3d
```

Be aware that the Raspberry Pi 4 has 2 HDMI outputs and some of the programs might open in HDMI1. Make therefore sure to be connected to HDMI0.

3.2.2 Setup ROS Noetic Ninjemys

After Ubuntu 20.04 has been installed, ROS Noetic Ninjemys[1] needs to be installed. Then, install further packages `python3-roslaunch` [5] to run ROS launch files, `rosserial` [6] and `rosserial_arduino` [7] to communicate with the Arduino:

```
sudo apt-get install python3-roslaunch
sudo apt-get install ros-noetic-rosserial && ros-noetic-rosserial-arduino
```

3.2.3 Setup PlatformIO

For uploading code to the Arduino Mega 2560 easily, without an IDE, PlatformIO [8] was used. Here the code can be uploaded directly to the Arduino through the terminal. To set up PlatformIO, use the following commands:

```
sudo apt-get install python3 python3-pip
sudo python3 -m pip install -U platformio
```

To upload the code to the Arduino Mega 2560, go to the specific folder where the code is located and use the following command in the terminal:

```
pio run -e megaatmega2560 -t upload
```

Using PlatformIO reduces the amount of graphical interface which is needed to perform SLAM. In this way, the user is able to SHH onto the robot and upload the code all from the terminal.

[1] Install ROS Noetic Ninjemys on Ubuntu: http://wiki.ros.org/noetic/Installation/Ubuntu

3.2.4 Setup display

The installation of the connection between the OLED display and the Raspberry Pi can be followed below:

```
pip3 install Adafruit_GPIO
pip3 install adafruit-ssd1306
sudo apt-get install python3-dev python3-rpi.gpio

wget https://archive.raspberrypi.org/debian/pool/main/r/raspi-config/raspi-config_20200601
    ↪ _all.deb -P /tmp
sudo apt-get install libnewt0.52 whiptail parted triggerhappy lua5.1 alsa-utils
sudo apt-get install -fy
sudo dpkg -i /tmp/raspi-config_20200601_all.deb
```

Enable the I^{2}C interface[2] and implement a permanent solution of setting I^{2}C permissions for non-root users[3].
Install the ROS package `robot_upstart` [9] and create an Ubuntu Linux upstart jobs[4] for the ROS node[5] `node_OLED` (see tab. 2). This will then display the IP address the Raspberry Pi is connected with when the Raspberry Pi is booting up as seen in fig. 3.

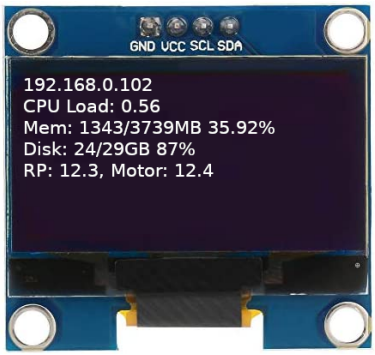

Figure 3: The OLED display shows important information such as IP address and battery voltage.

3.2.5 Setup Raspberry Pi camera

Enable the camera[6] and remove the comment of `dtparam=i2c_arm=on` when inside `/boot/firmware/config.txt`. Furthermore, give non-root users access to the camera device with:

```
echo 'SUBSYSTEM==\"vchiq\",GROUP=\"video\",MODE=\"0660\"' > /etc/udev/rules.d/10-vchiq-
    ↪ permissions.rules
sudo usermod -a -G video $USER
sudo reboot
```

[2] https://www.instructables.com/Raspberry-Pi-Monitoring-System-Via-OLED-Display-Mo/
[3] https://lexruee.ch/setting-i2c-permissions-for-non-root-users.html
[4] https://roboticsbackend.com/make-ros-launch-start-on-boot-with-robot_upstart/
[5] http://wiki.ros.org/rosnode
[6] https://zengliyang.wordpress.com/2021/01/04/raspberry-pi-4b-ubuntu-20-04-camera/

After having rebooted the Raspberry Pi, the camera may be tested with `sudo apt-get install ros-noetic-rqt-image-view && rqt_image_view`.

Camera calibration

The calibration of the Raspberry Pi camera is performed using OpenCV as shown in fig. 4. Firstly, install the calibration package with the following commands:

```
sudo apt-get install ros-noetic-camera-calibration
sudo apt-get install python3-rosdep
sudo rosdep init
rosdep update
rosdep install camera_calibration
```

Then publish the image feed over ROS and run the calibration software.

```
rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.026 image:=/camera/image_raw camera
```

Here the calibration is done using an 8×6 checkerboard where each checkerboard field has a width and height of 0.026 m. The generated calibration file is then modified to suit the needs of the ORB-SLAM3 implementation. The calibration file is obtained after the completion of calibration with over 80 readings and can be found in app. A.

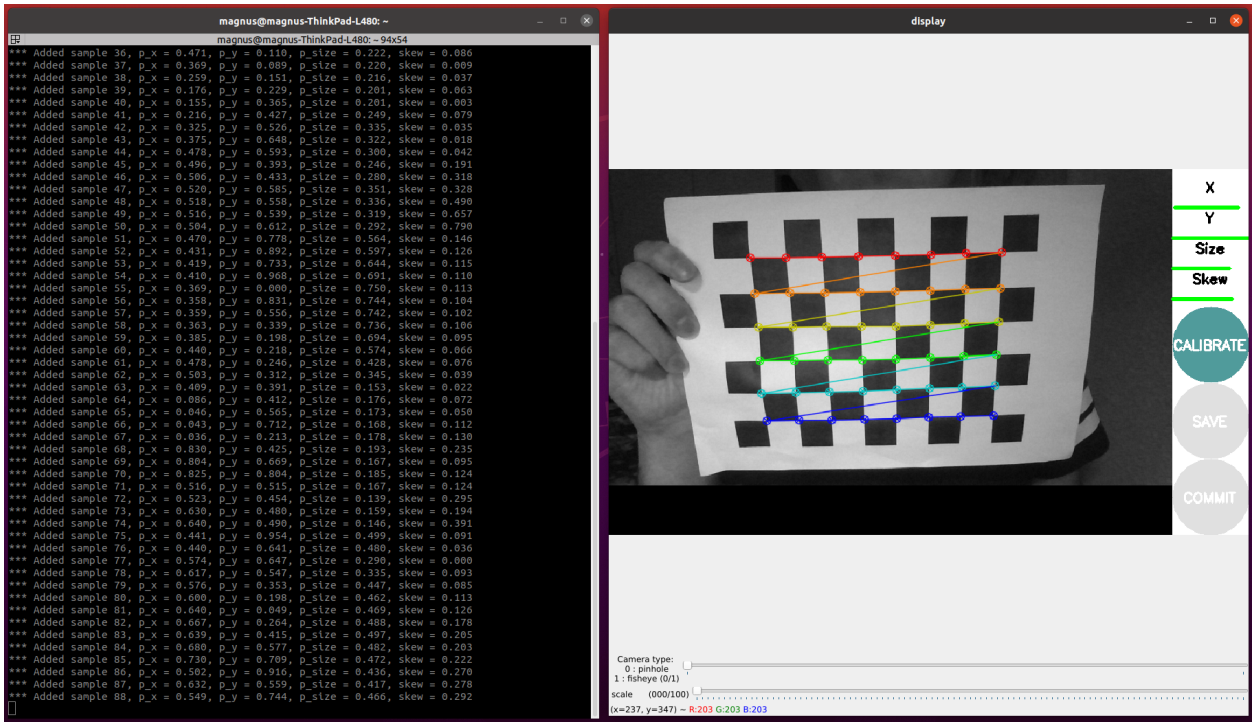

Figure 4: Calibration of the Raspberry Pi Camera. The left window shows the constant measured calibration values, whereas the right window shows the output of the camera. The camera calibration file can be found in app. A.

3.2.6 Setup ORB-SLAM3

To run ORB-SLAM3, the libraries OpenCV and Pangolin [10] need to be installed. OpenCV is used to manipulate images and features, whereas Pangolin is used for visualisation and user interface. Having a Raspberry Pi with 4 GB improves the building time as this process uses a lot of RAM. This was tried with 2 GB RAM Raspberry Pi 4, which caused the board to crash and be unresponsive, which is more uncommon with the new board.

```
# Installing OpenCV
git clone https://github.com/opencv/opencv.git
cd opencv
mkdir build && cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -D
    ↪ BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_
    EXAMPLES=ON -D WITH_QT=ON -D WITH_GTK=ON -D WITH_OPENGL=ON ..
make -j4    # -j4 will speed up the process
sudo make install

# Installing Pangolin
git clone https://github.com/stevenlovegrove/Pangolin.git
cd Pangolin
mkdir build && cd build
cmake -B build
cmake --build build
```

When the two libraries are installed, the actual ORB-SLAM can be installed with the following commands:

```
git clone https://github.com/UZ-SLAMLab/ORB_SLAM3.git ORB_SLAM3
cd ORB_SLAM3
sed -i 's/++11/++14/g' CMakeLists.txt   # Change the compiler version from c++11 to c++14
chmod +x build.sh
./build.sh
chmod +x build_ros.sh
./build_ros.sh
```

ROS wrapper for ORB-SLAM3

To perform SLAM with ORB-SLAM3 and to take advantage of ROS, a ROS wrapper has to be installed. The `orb_slam3_ros_wrapper` [11] is a ROS wrapper for ORB-SLAM3 and has the idea to use the ORB-SLAM3 as a standalone library and interface with it, instead of putting everything together. The wrapper picks out important information that ORB-SLAM3 produces, calculates further information, and prints it out as ROS topics[7], which can be seen in tab. 1:

[7] http://wiki.ros.org/rostopic

Table 1: ROS topics published by `orb_slam3_ros_wrapper`.

ROS topic	Data type	Purpose
`/orb_slam3/camera_pose`	PointCloud2	All keypoints being tracked.
`/orb_slam3/map_points`	PoseStamped	Current camera pose in world frame.
`/tf`	TFMessage	Transformation from camera frame to world frame.

The three ROS topics can then be viewed in RViz [12] as in fig. 6. To run ORB-SLAM with this wrapper, use the below command with the launch file of the camera:

```
roslaunch orb_slam3_ros_wrapper RP4.launch
```

Open then another terminal and publish the video feed through a ROS topic. It is also possible to play a ROSbag[8] with already recorded data. A simple test where this is used can be found in test 2. The whole setup with installing and adding libraries is further explained online[9].

3.3 Communication between hardware and software

The communication between Raspberry Pi and the Arduino is done through USB with `rosserial`, while the connection from the Raspberry Pi to the machine that visualises SLAM is via WLAN. The software architecture of the robot consists of several ROS nodes which can be seen below in tab. 2.

Table 2: Implemented ROS nodes.

ROS node	Purpose
`/node_image`	Image from Raspberry Pi camera
`/node_joystick`	Give pressed and released keys and values from the joystick
`/orb_slam3_ros`	Prints out information as ROS topics that ORB-SLAM3 produces
`/node_motor`	Controlling the motors: velocity, turning radius, resetting the odometry
`/serial_node`	Protocol for wrapping standard ROS serialised messages and multiplexing multiple topics and services over the serial port
`/node_OLED`	Showing information on the OLED display

How each ROS node (oval) is connected, and the ROS topics (square) for each ROS node can be seen in fig. 5.

[8] http://wiki.ros.org/rosbag
[9] https://github.com/MagnusErler/Building-a-map-using-SLAM-on-a-Raspberry-Pi

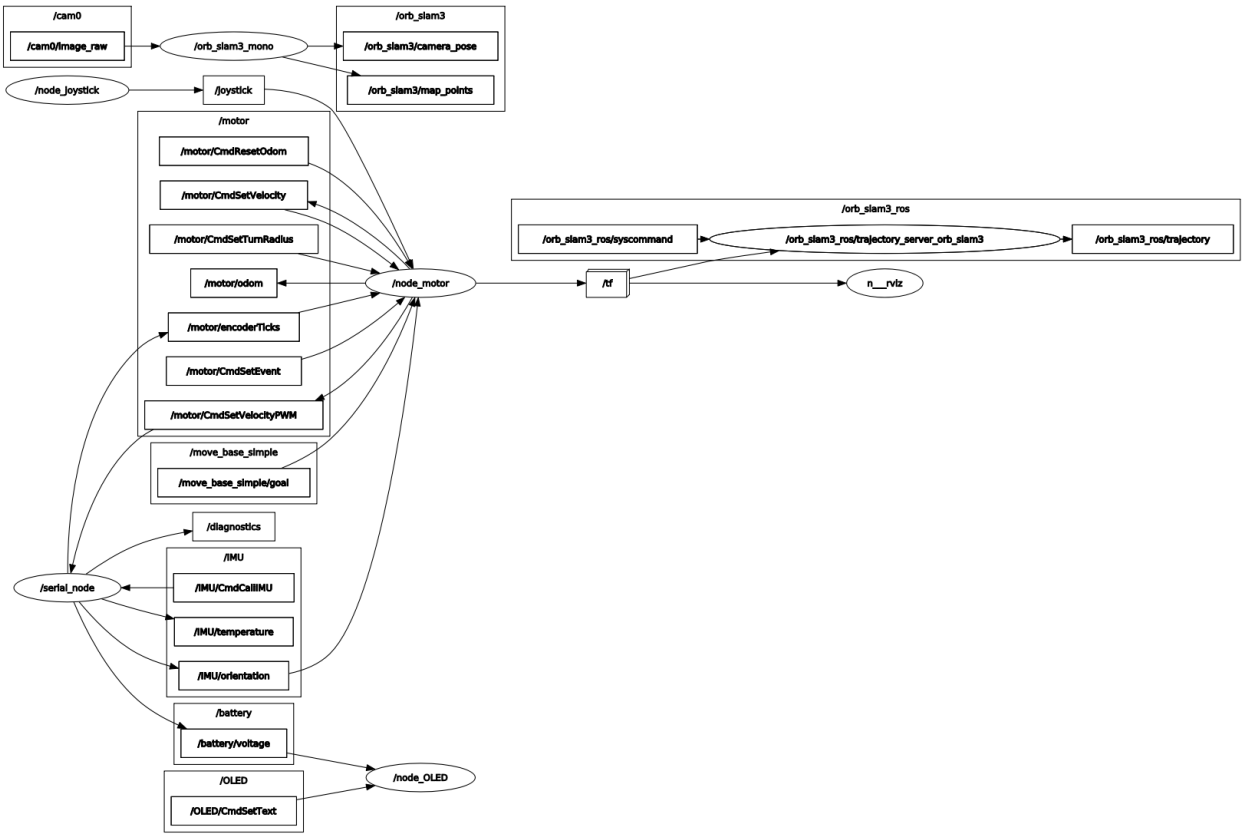

Figure 5: RQt graph [13] of active ROS nodes with `orb_slam3_ros_wrapper`. A larger figure can be found in app. C

The ROS nodes at the far top and on the right side are used to perform SLAM. The other nodes are used to communicate with the other parts of the robot. It may look complicated but the RQt graph visualises the connection very well.

Every ROS topic is categorised into *publisher* and *subscriber* and can be seen in tab. 3 and tab. 4. Here the name and the datatype of the ROS topics are described. Furthermore, does the table explain the purpose with a small description.

Table 3: Publisher.

ROS topic	Data type	Purpose
/battery/voltage	Float32MultiArray	Give the voltage of the battery for the motors and RP
/camera/image_raw	Image	Give image from Raspberry Pi camera.
/IMU/orientation	Int16MultiArray	Give the orientation in degrees [pitch, roll, yaw]
/IMU/temperature	Float32	Give the temperature (from the MPU9150-chip)
/joystick	String	Give pressed and released keys and values from the joystick
/motor/encoderTick	Int16MultiArray	Give the encoder ticks for the two wheels [L, R]
/motor/odom	Odometry	Give the odometry of the robot (position, orientation, and linear- and angular velocity)
/move_base_simple/goal	PoseStamped	Give the position and orientation of the desired location
/tf	TFMessage	Give the odometry of the robot (position, orientation)

Table 4: Subscriber.

ROS topic	Data type	Purpose
/IMU/CmdCaliIMU	Empty	Calibrate the IMU
/motor/CmdResetOdom	Empty	Reset the odometry
/motor/CmdSetTurnRadius	Float32	Set the turning radius [m]. O.B.S. set velocity first
/motor/CmdSetVelocity	Float32	Set the velocity of both motors [m/s]
/motor/CmdSetVelocityPWM	Int16MultiArray	Set the velocity of both motors (0 = off and 255 = max speed). Negative values will drive the motor backwards.
/OLED/CmdSetText	String	Write 1 line of text to one of the 8 lines on the OLED display. Lines 1-5 are reserved for IP address, CPU Load, Memory, Disk and Voltage. The display updates every 1 sec.

The datatypes of each ROS topic have been carefully chosen to use as little bandwidth as possible. Likewise, not having a rostopic for every battery voltage, but combining this in a single topic, may reduce the amount of communication. How each ROS topic is used with examples can be found online[10].

[10] https://github.com/MagnusErler/Building-a-map-using-SLAM-on-a-Raspberry-Pi

4 Testing of the Design

The design and its implementation have been thoroughly tested. Two tests on the camera performance are described in detail in the following. The first test investigates the time between camera image capturing and SLAM processing in RViz. The second test practises how effective the Raspberry Pi 4, with attached camera and ORB-SLAM3, describes its path and determines its position.
In order to verify and test the robot design, a performance comparison on another machine has been performed. Therefore the two machines, that are used for the testing, are:

1. Raspberry Pi 4 Model B as described in section 2.1.1
2. Lenovo ThinkPad L480 (Intel i5-7200U 2.5 GHz; 32 GB RAM; Ubuntu 20.04)

The Raspberry Pi 4 has its camera connected directly to the GPU whereas the webcam of the Lenovo Thinkpad is connected through USB. Both machines though use the same python-script to publish the camera feed through ROS, which is shown below in listing 1.

Listing 1: Publishing camera feed through ROS.

```
10 # ROS
11 rospy.init_node('node_image', anonymous=True)
12 pub = rospy.Publisher('/cam0/image_raw', Image, queue_size=1)
13
14 rate = rospy.Rate(50)
15
16 # CAMERA
17 cap = cv2.VideoCapture('/dev/video0', cv2.CAP_V4L)
18 cap.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'))
19 cap.set(cv2.CAP_PROP_FRAME_WIDTH, 500)
20 cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 300)
21 cap.set(cv2.CAP_PROP_FPS,30)
22
23 br = CvBridge()     # Used to convert between ROS and OpenCV images
24
... ...
32
33 while not rospy.is_shutdown():
34
35     if (cap.isOpened()): # Checking if cap object has started capturing the frames
36
37         ret, frame = cap.read()
38
39         if not ret:     # Checking if the actual frame is not corrupt
40             break
41
42         # Converting OpenCV image to ROS image and publishing as an 8bit BGR image
43         pub.publish(br.cv2_to_imgmsg(frame, "bgr8"))
44     else:
45         rospy.loginfo("Can not open camera video feed")
46
47     rate.sleep()
```

Lines 11 and 12 initialises the ROS node and sets up the ROS publisher with a queue size of 1 [wiki.ros, n.d.]. Line 14 sets the rate of the while-loop in line 36 and therefore the publishing rate of the ROS images. Line 17 adds the image input and converts the image from Linux images to OpenCV-image (V4L = **V**ideo **4** (For) **L**inux). Instead of choosing a physical camera ('/dev/video0'), it may also be useful to publish a video by writing the path to the video instead. Lines 18 to 21 then sets the video format to Motion Jpeg, the frame width and height, and the frames per second. Afterwards, the publishing begins by checking if ROS has; not been shut down, the camera is open, and the frames are not corrupt. When this is checked the OpenCV images are converted to ROS images and published as an 8 bit BGR image.

After having started the publishing of the camera feed, ORB-SLAM3 is run with `orb_slam3_ros_wrapper` and can then be viewed in RViz (fig. 6).

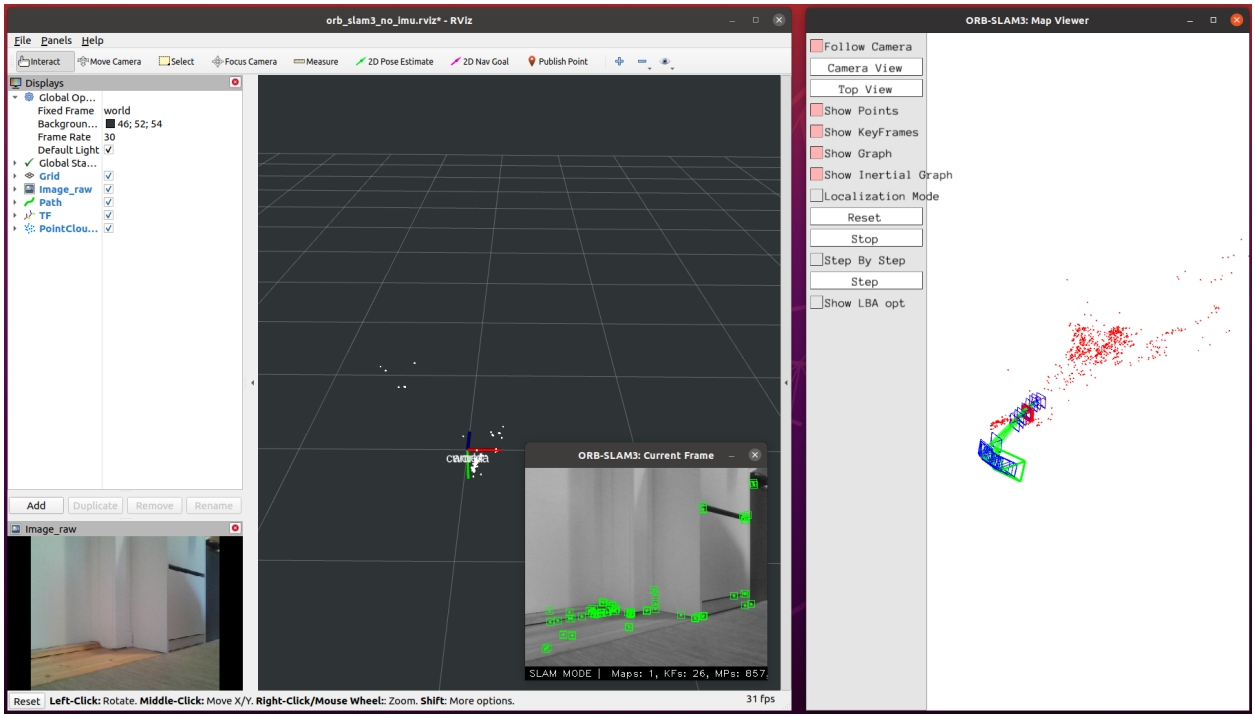

Figure 6: Screenshot of ORB-SLAM3 in action inside RViz.

4.1 First Test: Measuring camera frame rate

In the first test, the response time is measured for the Raspberry Pi or the Lenovo ThinkPad. The measurements of the Raspberry Pi are also categorised according to the type of connection (WLAN or LAN) to the computer visualising SLAM. WLAN means that the Raspberry Pi is connected to the same local network as the computer through a router. When using LAN, the Raspberry Pi is connected to the router through an Ethernet cable.

The response time is reported as frame rate measured in **F**rames **P**er **S**econd (FPS) in tab. 5. The frame rate is taken from RViz (lower right corner of the left window of fig. 6).

Table 5: Test 1 measures the response time (given in **F**rames **P**er **S**econd, FPS) for each setup.

Dataset	Raspberry Pi 4		Lenovo ThinkPad
	WLAN	LAN	
Live video (publishing frames through ROS)	∼3 FPS	25 FPS	33 FPS
Live video (with ORB-SLAM3)	∼2 FPS	24 FPS	31 FPS
MH_01_easy.bag	∼5 FPS	24 FPS	34 FPS

The initiated resolution and the frame rate for the Raspberry Pi camera are respectively set to 500 × 300 px and 30 FPS to have the best outcome of the SLAM. However, the resolution and the frame rate for the webcam on the Lenovo ThinkPad are set to 1280 × 720 px and 50 FPS.

Live streaming through LAN with the Raspberry Pi or using the ThinkPad gives a nice response time with almost no latency. However, the response time is not great when viewing the stream or dataset as a ROSbag in RViz when using a WLAN connection.

It can be concluded that visualising SLAM done by a mono camera on a mobile robot connected wireless can work but is not optimal.

4.2 Second Test: Moving in a rectangle

A common and simple test of a mobile robot is letting it move in a rectangle. In practice, the robot will rarely return to its starting position, but the error can be used to correct the driving forward and the turning to make it better next time. Moving in a rectangle can also be used when testing visual SLAM. The goal of the test is to merge the images at the start and end positions and map the journey of the robot. The test result is shown in fig. 7 where the red and black PointCloud, respectively shows what the camera is currently seeing and what it has saved. Furthermore, can each frame be seen as blue squares. It should be noted that the IMU is not used in this test, and therefore only the camera is able to determine if moving and turning.

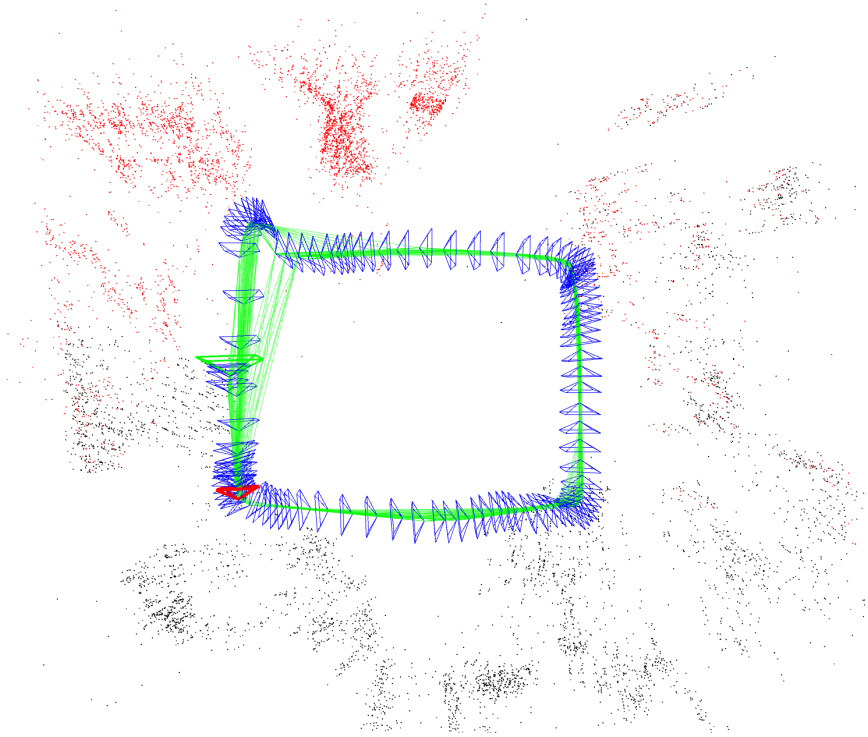

Figure 7: Moving the camera in a rectangle.

The true and measured moving in a rectangle of 220 × 200 cm can be seen in fig. 8.

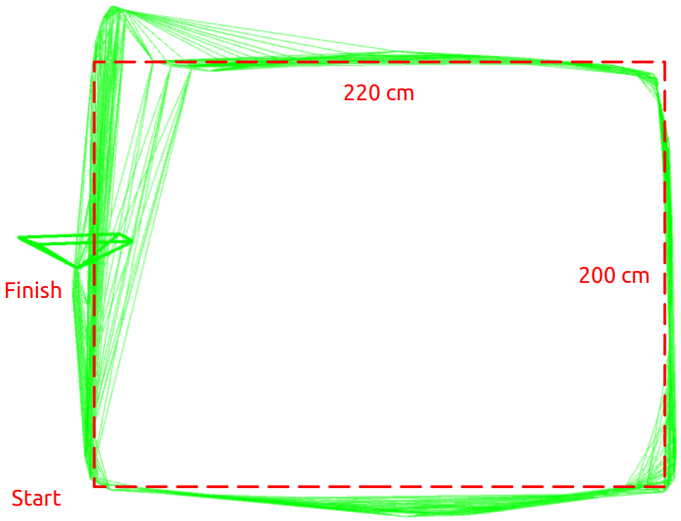

Figure 8: Moving the camera in a rectangle: True and measured journey.

Fig. 8 shows that the moving of the camera starts in the lower left corner looking forward (up in the figure). The camera then moves forward 200 cm before it slowly turns with a small radius to the right. Afterwards, it then proceeds moving forward (220 cm), before turning

again. It repeats this pattern until it has reached the starting position again, where it again moves a bit forward to align the last points with the first processed point. The measurements look very much like a rectangle with the given dimensions as shown in the figure.

The frames without the PointCloud and the graph can be seen in fig. 9 from above and the side.

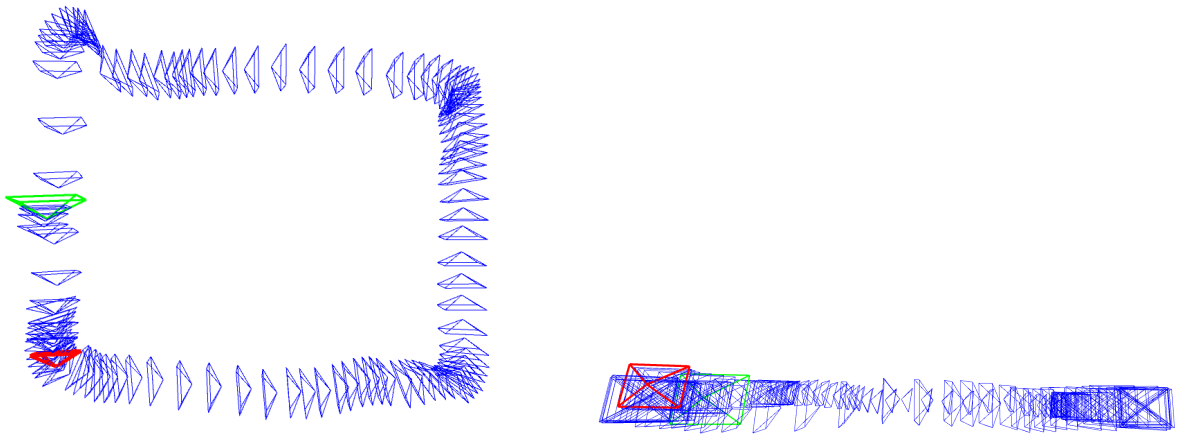

Figure 9: Movement without PointCloud and graph. The left figure shows the movement from above, while the right one shows it from the side.

The frames seen from the side are showing that the camera was actually moving horizontally, as expected.

It can therefore be concluded that visualising SLAM done by a mono camera can determine if the camera is moving and turning, and is able to connect the previous PointClouds when seeing those again.

5 Conclusion and suggestions for future research

The project has been a challenging task with many decisions to consider. It could be demonstrated that running visual SLAM with a mono camera on a Raspberry Pi 4 is achievable, although performance would benefit from a more powerful processor and further optimised code. Running an operating system with a graphical interface is convenient to perform heavy calculations such as SLAM, and ROS turns out to be a suitable tool for performing SLAM. The obtained image capture response time is slow but reasonable. The chosen design is able to determine all locations, combine that with old information, and move to a specific location and orientation if given inside RViz. However, it is still neither possible to perform path planning around obstacles or localisation in a given, predefined, map.

In overall conclusion, it is possible to perform SLAM on a Raspberry Pi 4 and to generate a map with a single camera only.

Potential further research includes custom specialised **B**ag **o**f **W**ords (BoW) vocabulary input for specific environments in which the robot operates and not just the vocabulary that comes with ORB-SLAM3. Convert PointClouds over e.g. walls to surfaces, so only a few PointClouds are needed. Furthermore, describing objects viewed by the camera as simpler objects (e.g. a vase to a cube) could be considered. Alternative visual SLAM libraries such as `ORB—VSLAM`[11] or `stella_vslam`[12] may increase performance. Other types of sensors than an IMU, e.g. LiDAR (**Li**ght **D**etection **A**nd **R**anging), may improve SLAM performance. A setup of performing SLAM solely with a LiDAR (no camera) has already been tested with good results.

[11] https://github.com/OpenSLAM/openvslam
[12] https://github.com/stella-cv/stella_vslam

References

A. Ademovic. An Introduction to Robot Operating System: The Ultimate Robot Application Framework. https://www.toptal.com/robotics/introduction-to-robot-operating-system#:~:text=The%20Robot%20Operating%20System%20(ROS,on%20working%20with%20peripheral%20hardware., 2015. Accessed: 22.12.2022.

J. C. Andersen. Robobot. http://rsewiki.elektro.dtu.dk/index.php/Robobot, 2022. Accessed: 01.05.2022.

T. R. Back-End. Using ROS on Raspberry Pi: Best Practices. https://roboticsbackend.com/using-ros-on-raspberry-pi-best-practices/, 2020. Accessed: 22.12.2022.

C. Campos, R. Elvira, J. J. G. Rodriguez, J. M. M. Montiel, and J. D. Tardos. ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, dec 2021. doi: 10.1109/tro.2021.3075644. URL https://doi.org/10.1109%2Ftro.2021.3075644.

D. Jones. Picamera. https://picamera.readthedocs.io/en/release-1.13/fov.html, 2017. Accessed: 04.12.2022.

G. A. Kumar. Is there any difference between ORB_SLAM1 and ORB_SLAM2 (Monocular)? https://github.com/raulmur/ORB_SLAM2/issues/333#issuecomment-813128785, 2021. Accessed: 19.12.2022.

D. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999. doi: 10.1109/ICCV.1999.790410.

PJRC. Teensy 3.5. https://www.pjrc.com/store/teensy35.html, 2022. Accessed: 18.12.2022.

ROS. Noetic Ninjemys. https://www.ros.org/reps/rep-0003.html#noetic-ninjemys-may-2020-may-2025, 2020. Accessed: 04.12.2022.

E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33833-8.

E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011. doi: 10.1109/ICCV.2011.6126544.

wiki.debian. LightDM. https://wiki.debian.org/LightDM, 2022. Accessed: 04.12.2022.

wiki.ros. Publishers and Subscribers. http://wiki.ros.org/rospy/Overview/Publishers%20and%20Subscribers, n.d. Accessed: 06.12.2022.

Appendices

Code

All code is available on GitHub at https://github.com/MagnusErler/Building-a-map-using-SLAM-on-a-Raspberry-Pi

Hardware

1. Raspberry Pi 4 Model B : https://www.raspberrypi.com/documentation/computers/raspberry-pi.html
2. Arduino Mega 2560 Rev3 : https://docs.arduino.cc/hardware/mega-2560
3. 12 V DC Motor with encoder : https://p.globalsources.com/IMAGES/PDT/SPEC/067/K1139071067.pdf
4. Dual H-Bridge Motor driver (L298N) : https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf
5. Battery : LiPo battery, 11.1 V, 2200 mAh
6. Step-down voltage regulator (D24V22F5) : https://www.pololu.com/product/2858
7. Raspberry Pi Camera : https://www.raspberrypi.com/documentation/accessories/camera.html
8. IMU (MPU9150) : https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-9150-Datasheet.pdf
9. OLED display : https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf
10. 2.4 GHz Wireless Joystick : https://www.aliexpress.com/item/1005003150723573.html?spm=a2g0o.order_list.order_list_main.112.3e991802LSe25a

Software

1. OpenCV : https://opencv.org/opencv-4-4-0/
2. ROS Noetic Ninjemys : http://wiki.ros.org/noetic/Installation
3. Ubuntu 20.04.5 LTS (Focal Fossa) : https://releases.ubuntu.com/focal/
4. ORB-SLAM3 : https://github.com/UZ-SLAMLab/ORB_SLAM3

5. python3-roslaunch : https://packages.debian.org/sid/python3-roslaunch
6. rosserial : http://wiki.ros.org/rosserial
7. rosserial_arduino : http://wiki.ros.org/rosserial_arduino
8. PlatformIO : https://platformio.org/
9. robot_upstart : http://wiki.ros.org/robot_upstart
10. Pangolin : https://github.com/stevenlovegrove/Pangolin
11. ORB-SLAM3 Ros Wrapper : https://github.com/thien94/orb_slam3_ros_wrapper
12. RViz : http://wiki.ros.org/rviz
13. RQt : http://wiki.ros.org/rqt_graph

A Camera calibration file

```
[image]

width
500

height
300

[narrow_stereo]

camera matrix
456.534579 0.000000 245.994761
0.000000 456.762686 160.478440
0.000000 0.000000 1.000000

distortion
0.204413 -0.761642 -0.002920 0.002635 0.000000

rectification
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000

projection
445.967407 0.000000 233.011128 0.000000
0.000000 451.602966 156.575597 0.000000
0.000000 0.000000 1.000000 0.000000
```

B Wiring diagram

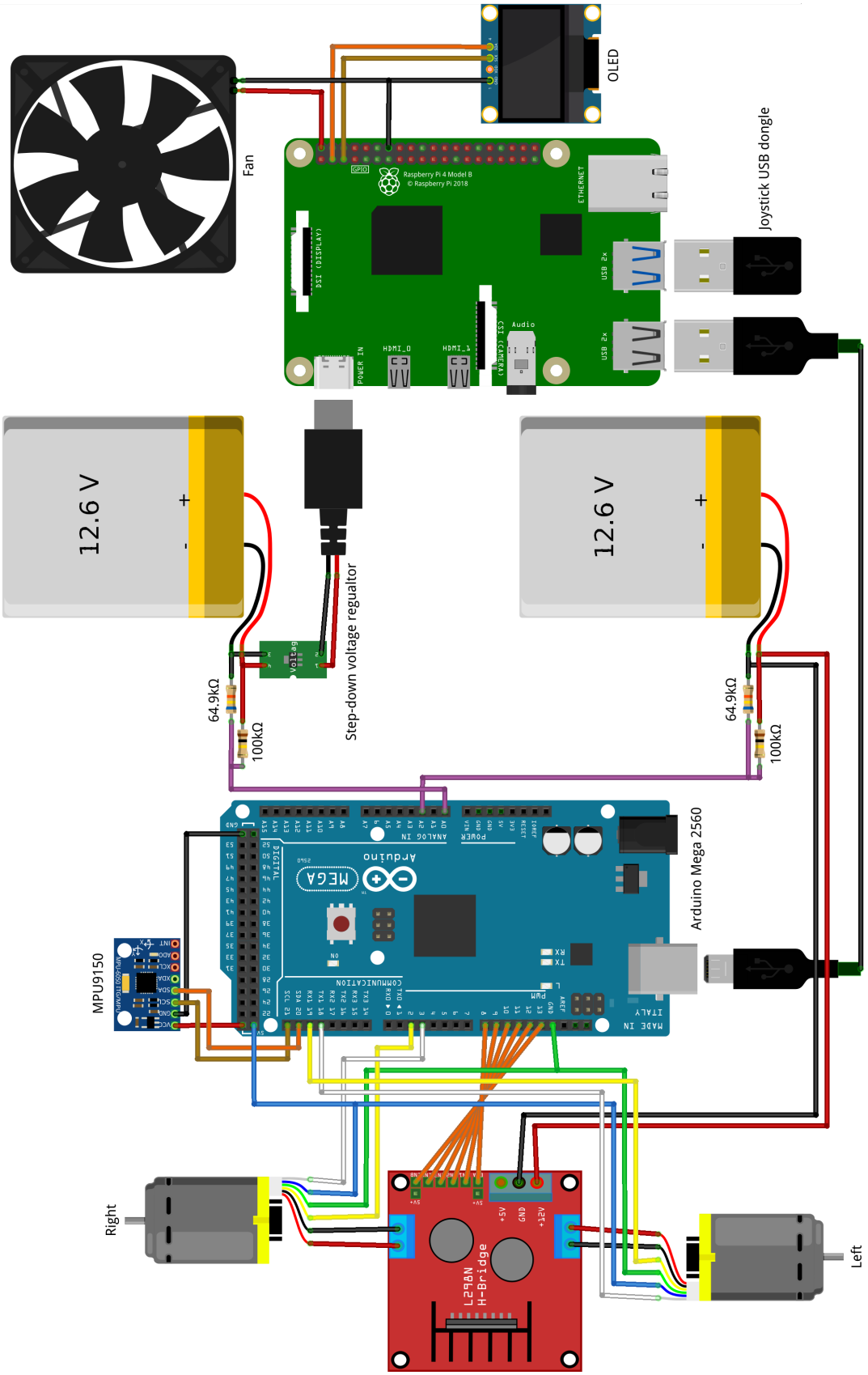

Figure 10: Wiring diagram of the robot.

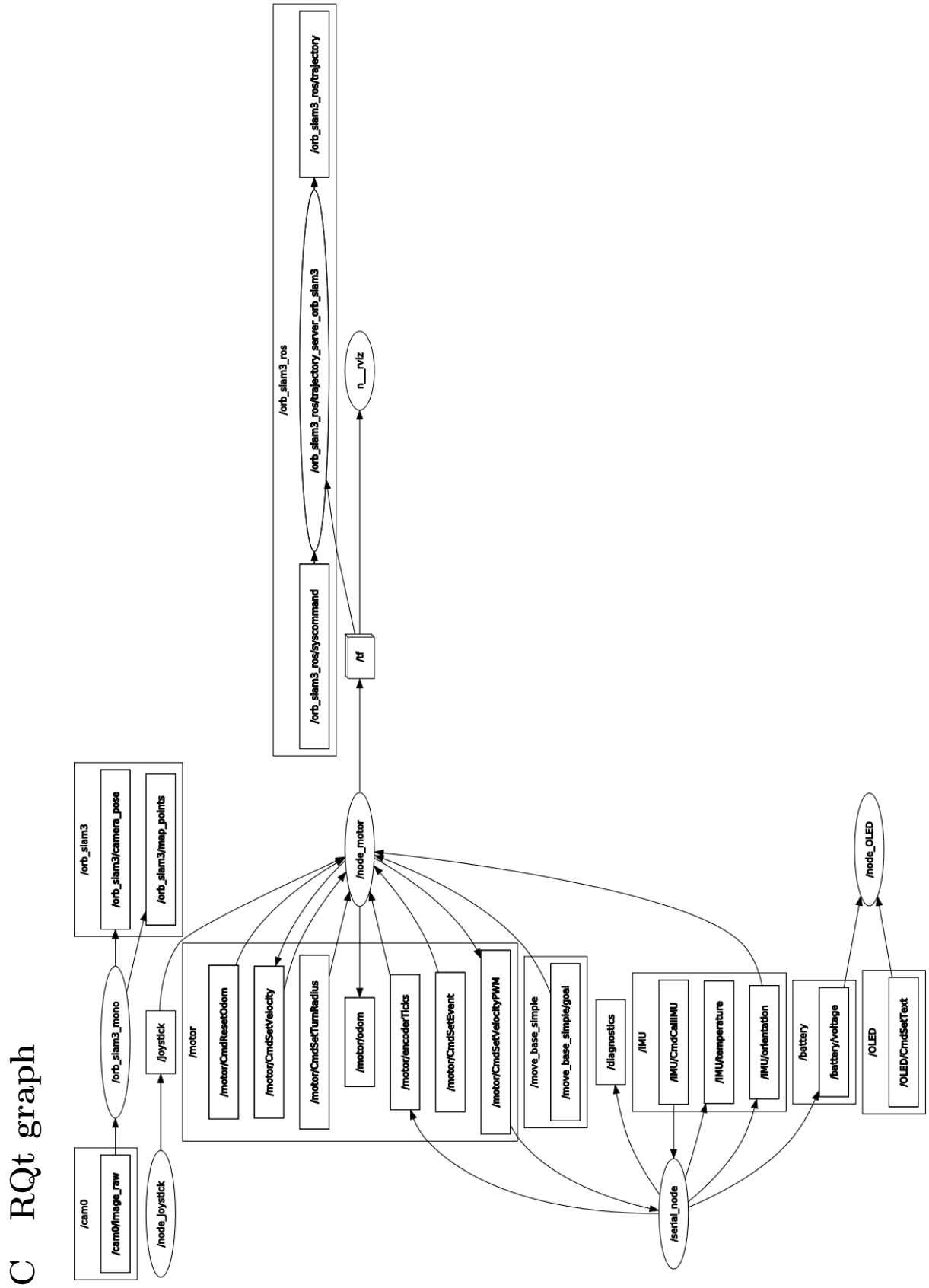

Figure 11: RQt graph of active ROS nodes with `orb_slam3_ros_wrapper`.