Title: Analysis of mobile platform localization methods using the Intel RealSense T265 sensor

Author: Kamil Goś github.com/KamilGos

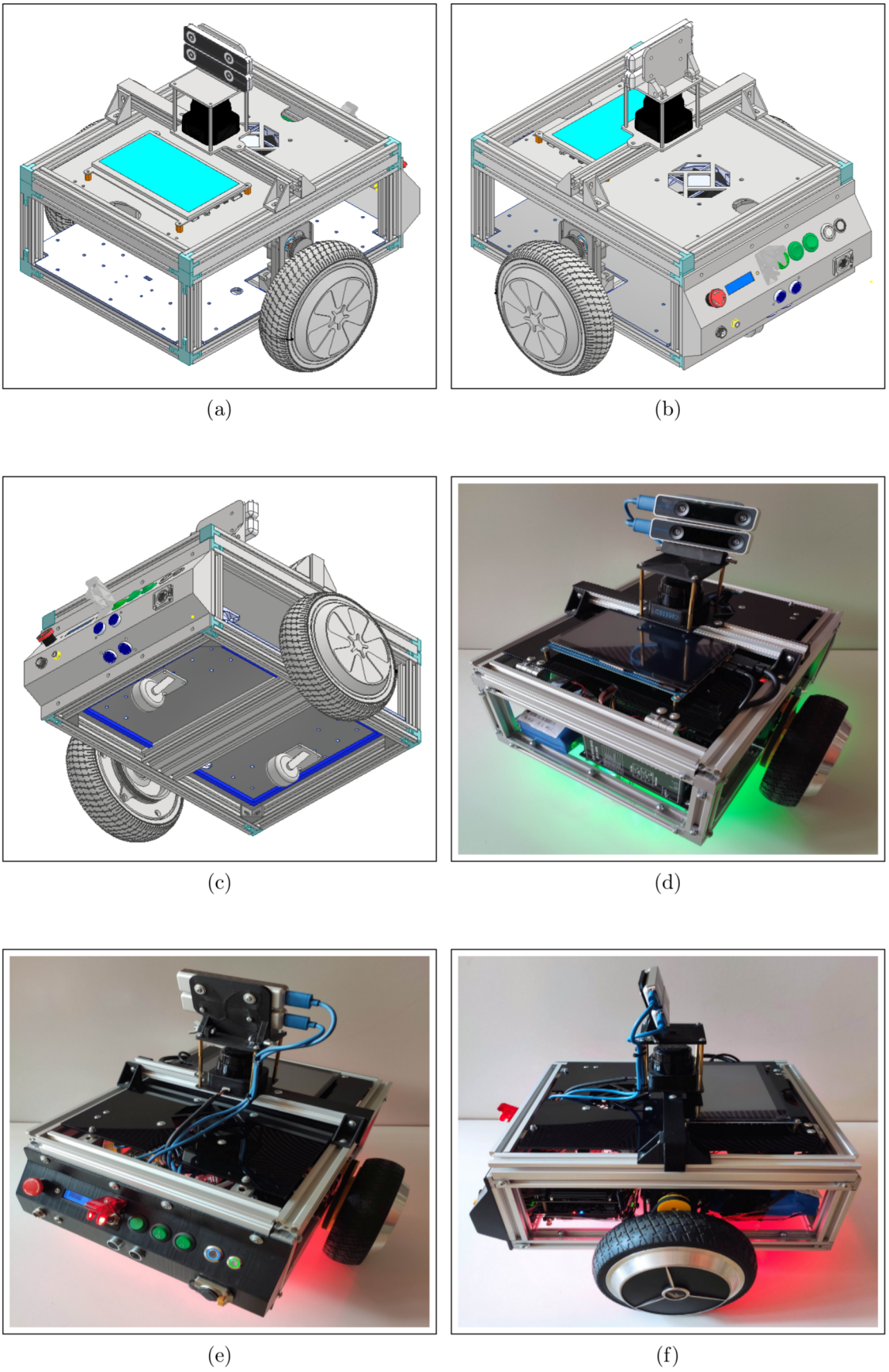
Chapter 1

Implementation

This chapter describes the engineering aspect of this master thesis. It includes hardware and software description. The hardware part illustrates a mobile platform that was built to allow testing the localization algorithms. The 3D model and photos of the built robot are presented, with the description of the most important specification of the built unit. The most important electronic components and their function in the entire system are also explained, as well as the powering system. This chapter includes graphs presenting key connections in the system. Additionally, one subsection is dedicated to the information about software (it describes the Robot Operating System (ROS) and how it is configured in the project).

1.1 Mechanics

Figure 1.1a-c shows the model of designed mobile platform and figure 1.1d-f pictures of the created robot. The 3D model was prepared using the Autodesk Inventor software. The mechanics of the robot have been designed to meet the assumptions of the differential drive model. The robot has two of the same wheels mounted on a common axis at the centre of the robot. This allows the platform to rotate in place. Each wheel is independent. It can spin at different speeds and in different directions (forward and backwards). To stabilise the robot, additional two swivel wheels were bolted to the chassis. The swivel wheels are free to turn and not controlled. The robot is fully consistent with the kinematics from the 2.1-2.5 equations. The body of the robot is a traditional cuboid, which makes the construction rigid and tough. The robot frame was built with aluminium construction profiles. They combine low weight, high strength and easy assembly. The robot's chassis and elevating elements were made of profiles with a diameter of 20x20mm. Additionally, the chassis has been reinforced around the engines using the additional 20x60 profile, which strengthens the structure around the wheels. The elements constituting the top of the robot are made of 20x40 profiles. The use of extended profiles makes it possible to easily install additional elements on the roof. With evenly distributed weight, the robot can carry up to 40 kilograms. The plate constituting the chassis of the robot is made of 4mm plexiglass. Mounting holes have been laser cut, thus obtaining very high accuracy. Plexiglass is a sturdy and durable material, however easy to drill, so the element can be easily modified. The chassis plate is attached to the profiles with bolts.



(a) (b) (c) (d) (e) (f)

Figure 1.1: Prepared robot: a)-c) 3d model, d)-f) built robot

This element is also the basis for the assembly of all electronics. The roof of the robot is also made of plexiglass (3mm, black). It is attached to the robot with hinges and neodymium magnets. This element has an additional cut-out hole for the fan, thanks to which the exchange of air between the robot and the environment is easier. The rear part of the robot is a 3D printed panel with displays, pushbuttons, switches and connectors. The robot is driven by two BLDC wheels. The tyre is made of rubber, which significantly increases the traction of the robot to the ground. This type of wheels is characterized by a fixed shaft. This shaft is attached to the robot's construction with a holder and massive metal screws, which makes them very stable. Additionally, 3D printed gears are mounted to the wheels, which enables the transmission of torque to rotary encoders. The second gear is mounted on the encoder shaft. To connect the shafts the timing belt is used. All elements are GT2 type. The use of such elements made it possible to obtain the gear ratio 142:50, which gives 2.84 revolutions of the encoder shaft per one revolution of the wheel. Table 1.1 shows the basis robot specification.

Table 1.1: Specification of mobile platform

Frame (mm)	
Width	290
Length	360
Height	160
Wheels (mm)	
Diameter	165.1
Width (without shaft)	61
Width (tire)	45
Width (with shaft)	113.4
Encoders	
Resolution (PPR)	1600
Encoder/wheel gear ratio	142:50
Pulses per one revolution of wheel	4544
Pulses per one meter	8672
Frame + wheels (mm)	
Width	440
Height	220
Whole robot	
Mass (kg)	14
Max speed (m/s)	4.2

1.2 Electronics

Figure 1.2 shows the general idea of robot electronic system architecture. It presents the most important elements of which the robot is composed. The whole system is divided into two main parts labelled *ROS* and *Microcontroller*. Elements in the *ROS* area are controlled directly by the Robot Operating System, while the elements in the *Microcontroller* area are controlled by the algorithms implemented on a used microcontroller. The microcontroller is also a bridge between the ROS and motion control algorithms.

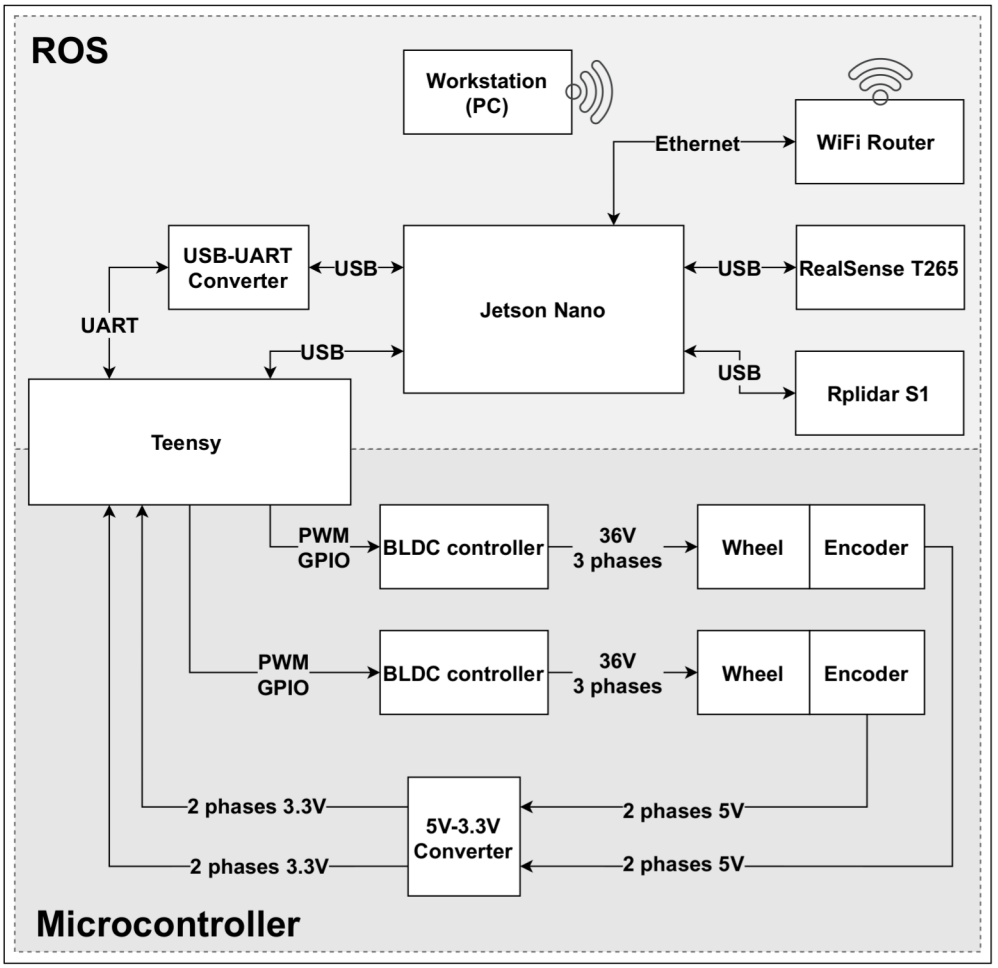


Figure 1.2: Overview of system architecture

1.2.1 Core elements

The following paragraphs present the most important elements and their function in the system.

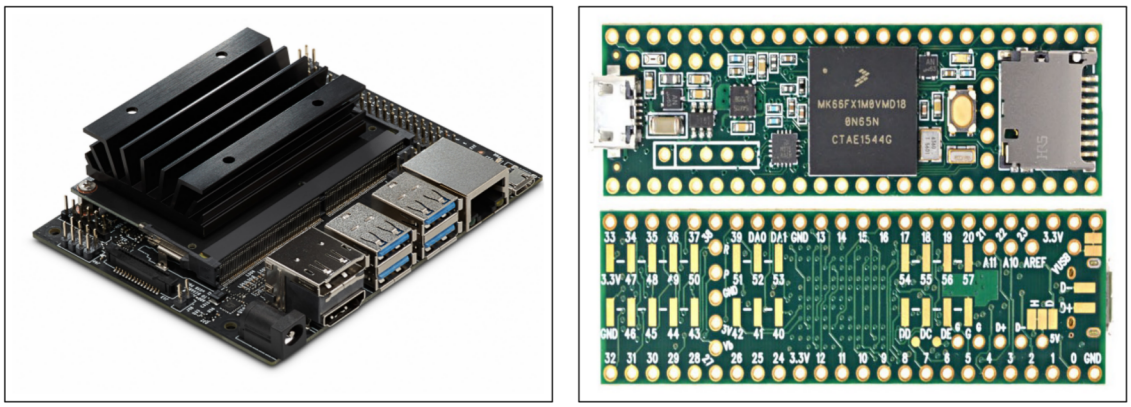
Computer

As a core processing unit the Jetson Nano is used. It is a small, powerful computer from Nvidia, equipped with an ARM Cortex A57 Quad-Core processor with a frequency of 1.43 GHz and an Nvidia Maxwell graphics processor with 128 CUDA cores. The used version has 4 GB of DDR4 RAM. It has all the necessary peripherals such as USB, Ethernet, HDMI, and GPIO. It supports the Ubuntu 18.04 operating system. Everything is delivered in very integrated housing, therefore the dimensions and weight of the device are very small. Figure 1.3a shows the picture of the Jetson Nano Developer Board.

In this thesis, Jetson nano is the main computing unit. All the calculations related to the localization algorithms take place on the Jetson. It is also the core machine for Robot Operating System. The T265 cameras, lidar and Teensy have been connected to the USB ports. The display that is mounted on the robot's roof is connected to the HDMI port, which allows showing the most important information for the user in graphical form. The Ethernet port was used to connect the Jetson to the router, which gives the possibility of wireless control of the robot.

Microcontroller

As a microcontroller board, the Teensy 3.6 is used. It is a small development board designed by the PJRC company. It provides one of the best price/performance ratios on the market. Teensy 3.6 include the impressive specification in a small package (6.1 cm x 1.8 cm) and at a low cost. It has a 180MHz Cortex-M4F processor, Floating Point Unit, 58 Input/Output pins and many other peripherals and interfaces that can be easily used. All pins are rated to 3.3V. The board has a pre-flashed bootloader so it can be programmed using the onboard USB connector. The Teensy 3.6 can be programmed using any C editor or the Arduino software. Figure 1.3b shows the picture of both sides of Teensy 3.6.



(a) (b)

Figure 1.3: Core elements: a) Jetson Nano Developer Kit [?], b) Microcontroller Teensy 3.6 [?]

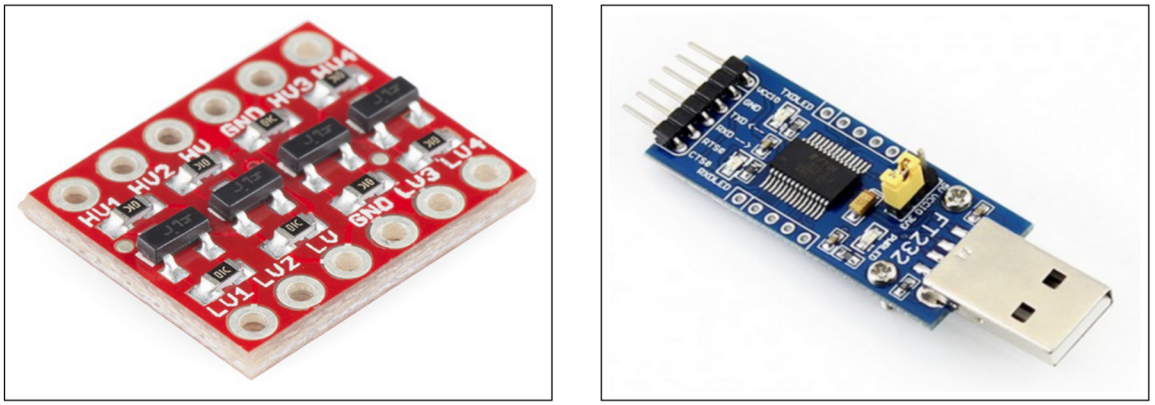
In this project, the teensy microcontroller plays the role of a bridge between electronics and the supervisory control system (ROS). Primarily, the board is responsible for motion control. The control of a motion is based on the closed-loop system with feedback from encoders and a PID regulator. It is responsible to steer the wheels according to the set values. The PID controller calculates the error value as a difference between the desired setpoint and the measured value from the encoders. This controller tries to minimise the error by sending an appropriate control signal to the input of BLDC motor drivers.

The algorithm for calculating the PID controller output contains three separate constant parameters: proportional, integral and derivative, denoted respectively P, I and D. The action of the P element compensates for the current error, the I component compensates for the accumulation of past errors and the D element compensates for the expected shortcomings in the future. The weighted sum of these three variables multiplied by the error values is the output of the regulator. The error value is calculated as a difference between the setpoint and the feedback signal. The feedback signal is the signal from encoders. Each wheel has its control loop.

In addition to key functionalities, teensy is also responsible for blocking the wheels in dangerous situations and controlling the lighting of the robot. Also, it communicates with the ROS run on Jetson Nano using serial communication (rosserial protocol). The connection between Jetson and Teensy is made using the USB interface (the communication is two-way). The Teensy has 3.3V logic. As the used encoders work with a voltage of 5V, it

was required to connect these two devices through a voltage converter. For this purpose, a 4-channel bidirectional logic level converter based on MOSFET transistors with an N channel was used. It is a small device that safely lower 5V signals to 3.3V, so the signals from encoders can be safely connected to Teensy. Figure 1.4a shows the picture of this converter.

Another element used to improve the work with the teensy board is the USB/UART converter, which enables the communication between those two serial interfaces. The main serial interface available in Teensy is occupied by communication with ROS, so to be able to debug the device without interruptions, it was necessary to use another serial connection. As USB and UART are not fully compatible a converter was required. For this purpose, a Waveshare converter based on the FT232RL chip is used. In the operating system, it is visible as a virtual COM port, which makes everything very easy to use. The picture of this converter can be seen in the figure 1.4b.



(a) (b)

Figure 1.4: Required converters: a) logic lever converter [?], b) USB-UART converter [?]

Wireless connection

A TP-Link TL-WR841N WiFi router is used for communication between the workstation (PC) and the robot. It is a high-speed solution that is compatible with IEEE 802.11 b/g/n. Based on 802.11n technology, it gives users wireless performance at up to 300Mbps. The router allows the creation of wired and wireless local networks. This allows the robot to transmit information without internet access. To reduce the dimensions of the router, the outer plastic housing was disassembled and only the electronics board is placed inside the robot. The router is equipped with 2 integrated antennas, which were placed inside the robot. Figure 1.5a shows the TL-WR841N router in its original housing. Additionally, Bluetooth is used as the second method of wireless communication. It is only used to connect to the Jeoystik, which is used to manually control the robot.

Motors and wheels

In this robot, it was decided to use a specific drive that combines an engine and a wheel as one element. The two BLDC hoverboard-type wheels were used. Each wheel has 350W power. The diameter is 165.1mm (6.5 inches). The wheels are powered by three phases with a voltage of 36V. They are made to carry weights ranging to 100kg, which is very

(a)

Figure 1.5: Tp-link TL-WE841N router [?]

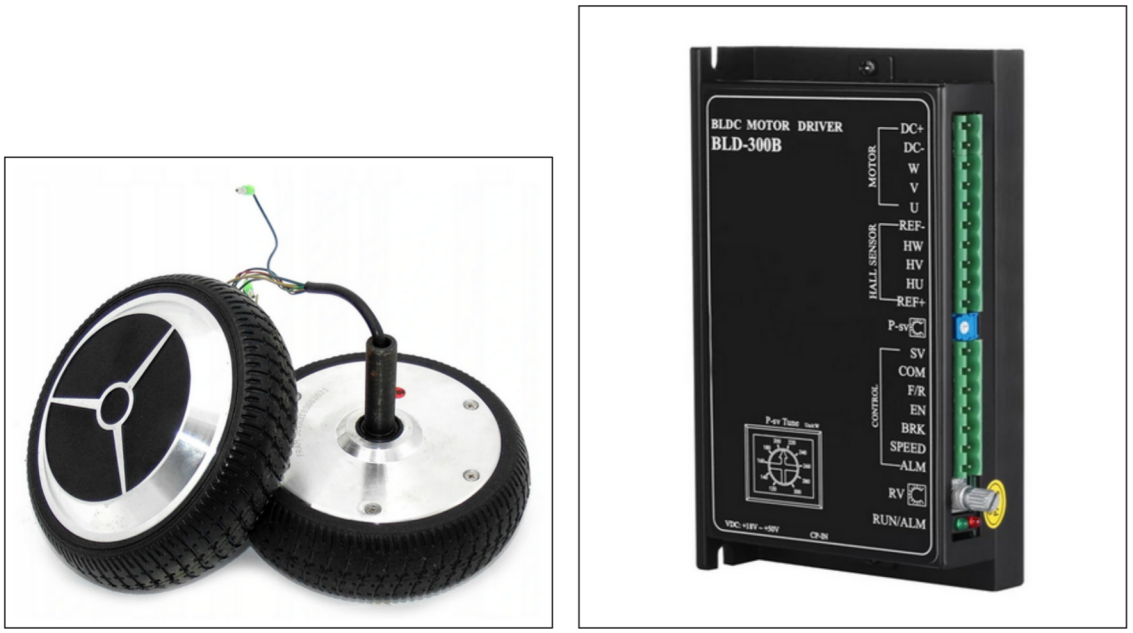
enough for this mobile platform. The wheel is assembled with the tyre. It is made of good quality rubber, which significantly increases the traction of the robot to the ground. Good traction is necessary for odometry calculation. Figure 1.6 shows the image of used wheels.

Brushless DC Motor Driver

The BLDC (BrushLess Direct-Current motor) motor is a specific type of electric motor that uses an electrically controlled commutator instead of a mechanical commutator with brushes. The coils are stationary and the magnets are on the rotor. The main advantage of BLDC motors is much higher durability and reliability resulting from the elimination of brushes. The result is also a quieter engine operation and higher energy efficiency. Another advantage of brushless motors is the ability to control the speed almost independently of the motor torque. The electronic commutator is powered by a direct current. The commutator system turns on and off the power supply to the coils, the magnetic field of which causes the rotor to rotate. This type of work requires the use of special drivers. BLD-300B drivers are used in this project. Those are high performance, cost-effective 3 phase BLDC motor drivers, which can provide power output up to 300VA. The design is based on advanced DSP technology. It has multiple functions: high torque, low noise, low vibration, PID speed loop, PID current loop, over-current protection and overload protection. It also gives the possibility of using the Pulse-Width Modulation (PWM) signal to control the motor speed, which was used in the robot. Each wheel requires its own driver. Figure 1.6b shows the used driver.

Power Supply

Figure 1.7 shows the general overview of the power system in the robot. It consists of two independent energy sources. The first source is the Power Battery Pack that is used to energise the wheels and the drivers. The 10S2P Li-ion battery is used. It is the self-balancing battery pack, that delivers 36V voltage and 4.4 amp-hours. It is made up of 10 cells connected in series and two in parallel, so it has 20 batteries inside. The use



(a) (b)

Figure 1.6: Elements of drive system: a) BLDC wheels [?], b) BLD300B BLDC Motor Driver [?]

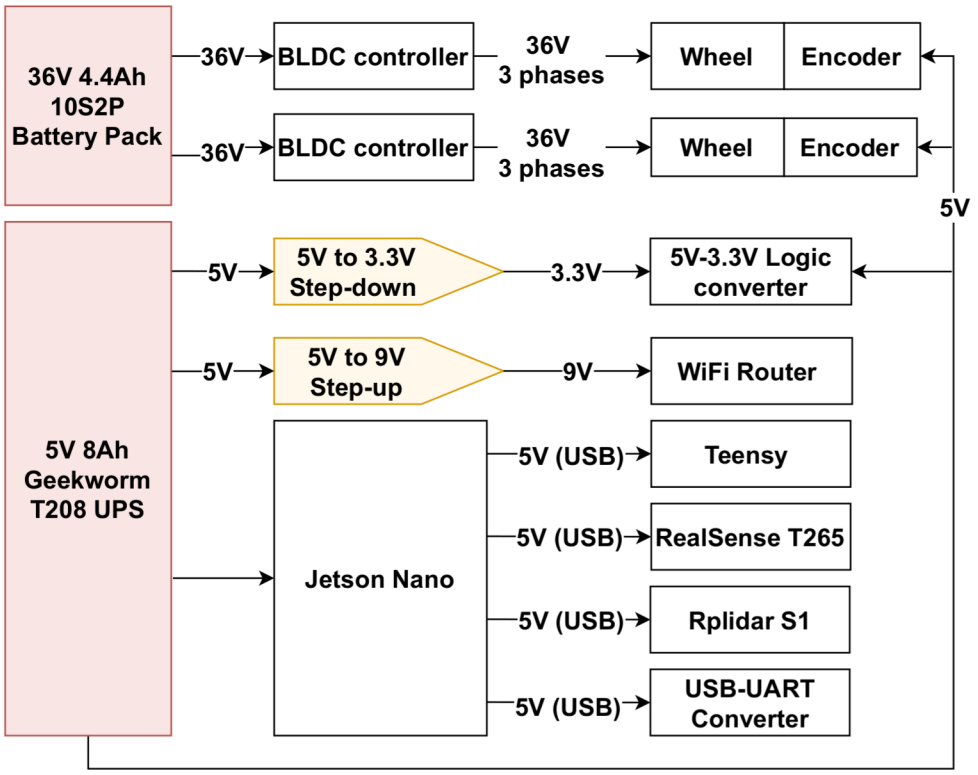
of such a battery makes it possible to efficiently power the wheels. Figure 1.8a shows the used battery. The second power source is the T208 UPS from Geekworm, which provides a very stable voltage of 5V. It consists of six 18650-type 3.7V batteries with a capacity of 3500mAh each. It provides enough power to supply the rest of the system. Figure 1.8b shows the picture of this device. Additionally, the system includes two voltage converters. One step-down the voltage from 5V to 3.3V, while the other step-up the voltage to 9V.

1.2.2 Sensors

Sensors are the key components for the robot to observe the environment and estimate its state. The first sensors are the encoders needed to test the localization algorithm based on the robot model (described in section **??**). The second sensor is the lidar RPLIDAR S1 from Slamtec, used to test the RF2O localization method. The last sensor is the Intel RealSense T265 camera, which is needed to test the Visual-Inertial Odometry algorithm, which is the main topic of this thesis.

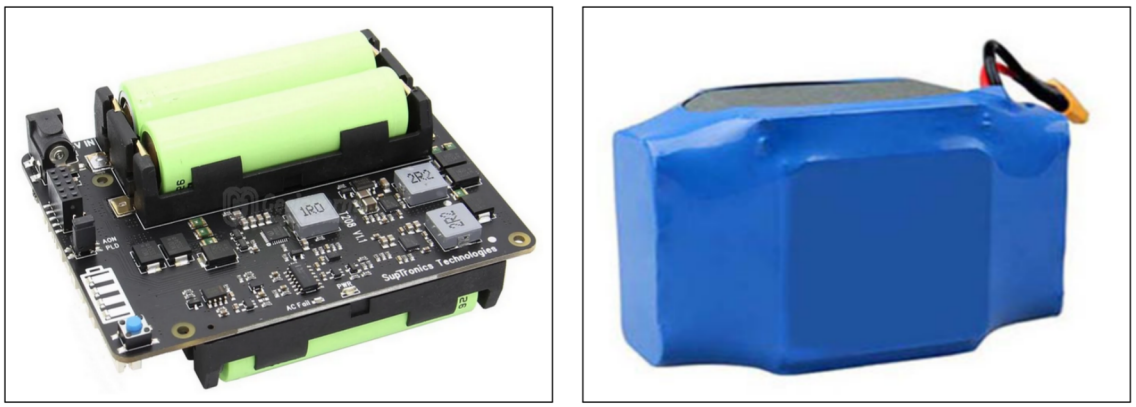
Encoders

The robot used incremental photoelectric rotary encoders. They generate a two-phase rectangular pulse signal by turning the dial with an optocoupler. The resolution is 400 pulses per phase, which makes it possible to obtain 1600 pulses per revolution using both phases. The maximum speed of the encoder is 5000 rpm. There is an NPN transistor (open collector configuration) at the output of the circuit, so a pull-up resistor is needed for correct work. The module has a voltage regulation system 750L05, so the encoder can be supplied with DC voltage from 4.8V to 24V. Figure 1.9a shows used encoders. To transfer the torque from the wheel to the encoder shaft, a set of gears and timing belt was used. A specially designed GT2 142 gear has been mounted on the wheel. On the other



(a)

Figure 1.7: Structure of power system



(a) (b)

Figure 1.8: Power sources: a) battery pack [?], b) T208 UPS [?]

hand, a GT2 50T gear was mounted on the encoder shaft. Both gears are connected using the 348x6mm timing belt. This configuration allows increasing the resolution of the encoder. The gear ratio is 142:50, which gives 2.84 revolutions of the encoder shaft per one revolution of the wheel. It gives 4544 Pulses Per Revolution, which is enough to implement a high accuracy control loop. As mentioned in the part related to the microcontroller, the encoders provide feedback to the control loop. Figure 1.9b shows the designed gear set.

(a) (b)

Figure 1.9: Used elements: a) encoder [**?**], b) gear set model

Slamtec RPLIDAR S1

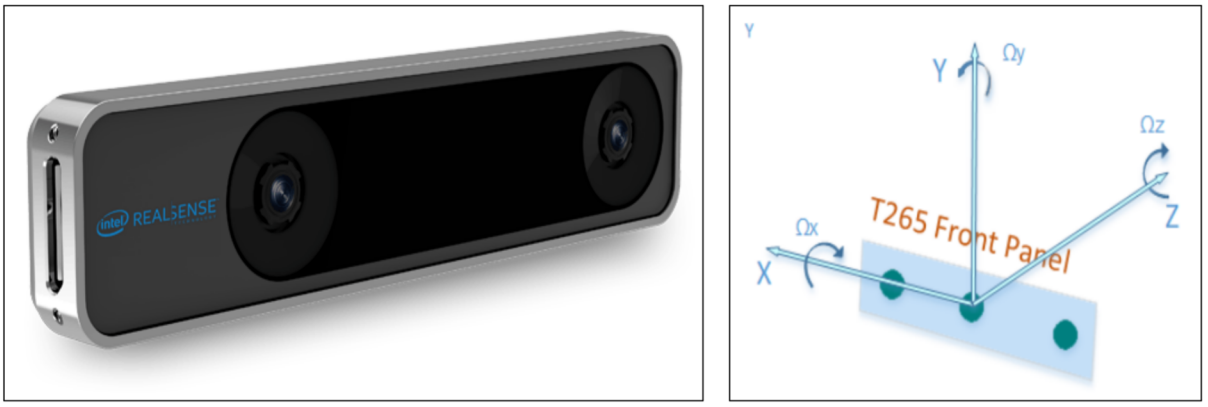
The RPLIDAR S1 is the 360 degrees 2D laser scanner developed by SLAMTEC. The typical scanning frequency is 10Hz (600rpm), and it can be freely adjusted within the 8-15Hz range according to the specific requirements. With the 10Hz scanning frequency, the sampling rate is 9.2kHz and the angular resolution is 0.391 degree. The system can perform scan within a 40-meter range. This scanner has stable performance when detecting objects in long-distance, objects in white or black alternatively and objects under direct sunlight. Due to these features, the RPLIDAR can operate in all kinds of indoor environment and outdoor environment. The RPLIDAR S1 is based on laser flight-of-time (TOF) ranging principle (described in details in section **??**). The device is prepared to work with ROS. It delivers *rplidarNode*, which reads RPLIDAR raw scan result using RPLIDAR's SDK and convert to ROS *LaserScan* message. Each subsequent scan is used by the RF2O algorithm to calculate the robot odometry. Figure 1.10 shows the device.

(a)

Figure 1.10: Slamtec RPLIDAR S1 [**?**]

Intel RealSense T265

The RealSense T265 from Intel is a tracking camera, powered by the Intel Movidius Myriad 2 Vision Processing Unit (VPU), that deliver from-the-shelf 6 DoF, inside-out tracking for highly accurate localization. It was released by Intel in March 2019 for 199 USD, which was accepted by the community as a cheap solution. This stand-alone V-SLAM device runs all algorithms directly on VPU, allowing for very low latency and it does not require any computations to be held on the master computer. It has a low power consumption that stays around 1.5 W, so it can be used for battery-powered solutions. It offers 6ms latency between movement and reflection of movement in the pose. The whole device is delivered in package 108x25x13mm size only 55g weight (figure 1.11a), so it can be used for small-footprint mobile devices such as lightweight robots and drones. It is also optimised for small-scale computers such as Jetson Nano. The camera includes two fisheye lenses (with resolution 848 x 800 each) for a combined, close to hemispherical 163±5°field of view for robust tracking even with fast motion. The T265 was also equipped with the BMI055 Inertial Measurement Unit, which allows for accurate measurement of rotation and acceleration of the device, to feed into the V-SLAM algorithms. The considered camera used the Virtual Reality standard local coordinate system as shown in figure 1.11b. The positive X direction is toward the centre between the lenses, the Y direction is upwards at the top of the device, and the Z direction is inward toward the rear of the device. This coordinate system does not match the local system of coordinates of the robot, therefore the output of the pose from the T265 must be converted from the original frame to the platform frame. Additionally, the vehicle coordinate system is related to the centre of gravity, which generally does not coincide with the T265 coordinate centre. Therefore, both rotation and translations corrections must be used. The camera is an element in the system that performs the VIO algorithm.



(a) (b)

Figure 1.11: Intel RealSense Tracking Camera T265: a) housing [?], b) coordinate system [?]

1.3 Software

1.3.1 Robot Operating System (ROS)

The Robot Operating System (ROS) is a framework used for creating robot software by connecting different modules to create complex robots. It includes many useful tools

and libraries that make the development of robotic system very simple and fast. The conventions used in ROS allows the creation of a complex and robust robot environment across a wide variety of robotic platforms. Creating truly robust robot software is a hard task. It requires a huge amount of knowledge about every aspect of robotics. ROS is a tool that enables collaborative robotics software development. It enables the use of ready-made, previously implemented and tested robotic modules, which significantly reduces the time and costs needed to run a given application. Thanks to the use of ROS the community can share their achievements (modules) which can be used by others to create the complete robotic system. The ability to share software component packages is the core power of ROS. For now[1] ROS contain over 3000 packages providing a very huge number of functionalities. It also supports multiple programming languages: C++, Python and Java. ROS is a perfect tool for use with platforms such as mobile robots, manipulators, autonomous cars, social robots, humanoids and many others. The entire project released under the BSD license, so it can be used for both scientific and commercial purposes for free.

ROS is not an actual operating system, however, it provides inter-process communication and support for low-level device control. The greatest advantage of ROS is its distributed, hybrid Peer-to-Peer architecture. It provides all the tools needed for creating a fully functioning robotic system. Figure 1.12 shows the general concept of ROS architecture. It can be explained as a system made of blocks (nodes) that exchange information (messages) with each other using the topics. The main node, called master, handle the correct exchange of information.

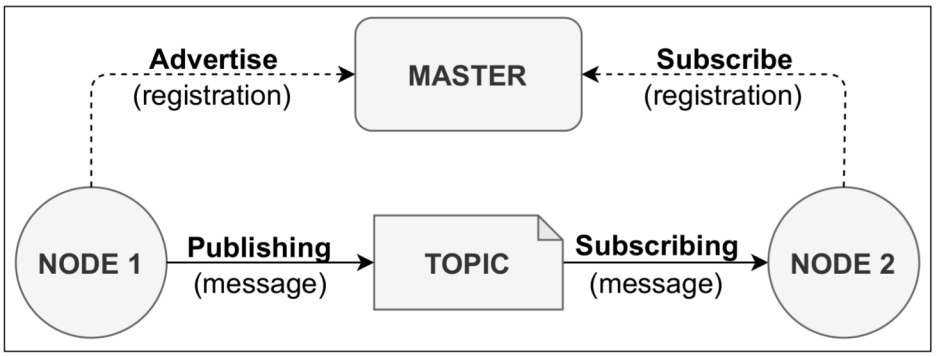


Figure 1.12: Basic ROS architecture model

Nodes are the executable parts of the system. They perform computational tasks. They represent single processes with a high degree of autonomy. ROS is designed to be modular, so usually, the entire system is build of many nodes. For example, one node can control a lidar, one control the camera and the other one can control the wheels. Every ROS system must include the master node. It is the supervisor for inter-process communication. It provides name registration, which enables exchanging data between nodes. The nodes communicate with each other by passing messages, which are data structure with predefined type. Messages using the publisher/subscriber communication model. A node sends a message by publishing it on a specific topic. Another node may read this message by subscribing to the same topic. There can be many publishers and subscribers for a single topic. Also, the single node may publish and/or subscribe to multiple topics. This makes it possible to create a very complex, asynchronous communication structure

[1] 04 May 2021

in a simple way. In this thesis, ROS is used to coordinate the work of the entire robot. All the localisation algorithms are executed using this system. It also allows recording all data appearing in the system, which helps data analysis.

1.3.2 Configuration

Figure 1.13 present the core nodes and messages that build the system. One can distinguish 8 elements:

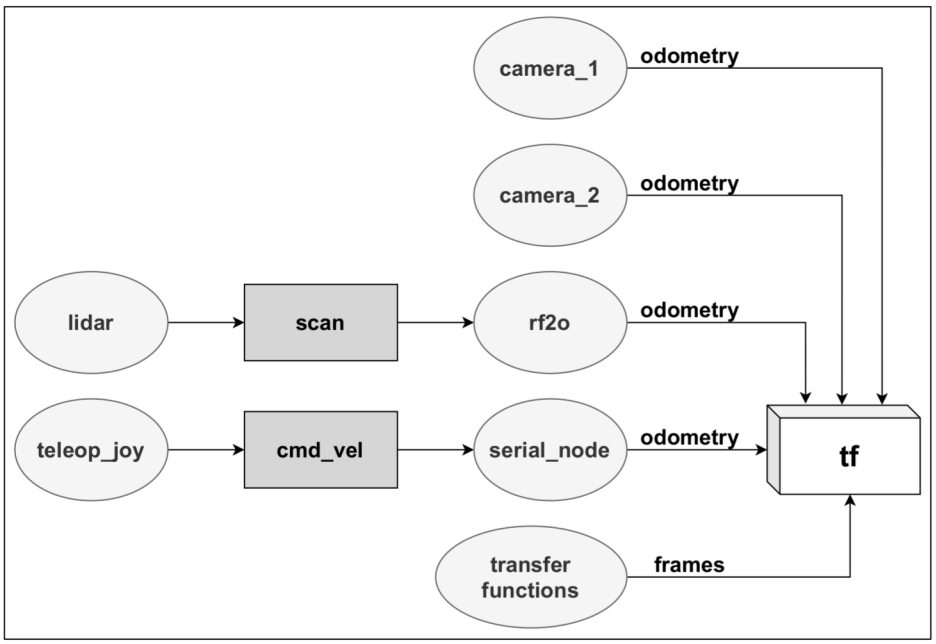


Figure 1.13: Core nodes and messages in system

- *tf* is a package that allows keeping track of multiple coordinate frames over time. It handles all the coordinate systems relations and lets the user transform points between them. There are 5 main frames in the robot: map frame (global coordinate system), base frame (mobile platform local coordinate system), lidar frame (lidar local coordinate system) and cameras frames (T256 local coordinate systems).
- *transfer functions* is the node that broadcast the transform between frames.
- *teleop_joy* is a node that maintains the connection with the joystick. It is used to allow wireless control of the robot using a classical gamepad with two potentiometers. It reads the signal from a joystick, processes it and publishes the control values using the *cmd_vel* topic. *cmd_vel* is a topic that sends the information about angular and linear velocity using standard *geometry_msgs/Twist* message.
- *serial_node* is a node that handles the serial communication with the microcontroller. It subscribes to the *cmd_vel* topic which allows to control the robot directly from the gamepad or using commands sending from the workstation. It sends the velocity to Teensy, which process the data and produce the control signal. On the other hand, *serial_node* topic is responsible for publishing the odometry information obtained from a microcontroller. Those information are sending using standard *nav_msgs/Odometry* message. *nav_msgs/Odometry* is the estimate, that contains the pose and twist information.

- *lidar* node is a programme that maintains the connection with lidar. It is supplied by Slamtec together with the lidar, so it is optimised for the given lidar model. It publish the *scan* topic which is a standard *sensor_msgs/LaserScan* message that contains information about distances to the obstacle for each measurement.

- *rf2o* is a node that execute the rf2o algorithm. It subscribes to the *scan* topic and processes it to obtain the odometry information.

- *camera_1* and *camera_2* are the same (but independent) topics that handle connection with RealSense T265 camera. This node is provided by Intel company, so as well as for lidar, it is fully optimised for this camera. Both of those topics publish several different topics but only the odometry topic is used.

During the tests of the localization algorithms, the *rosbag* package was also used, which enables the record of the messages that appear in the system. This package makes it possible to recreate the entire course of the experiment at any time without losing any information. The data collected in his way (called *rosbag*) is also easy to analyse because all the data can be exported to a standard CSV file or directly imported into a Python script. Also, it is important to note, that all the nodes responsible for odometry calculations run in parallel and independent of each other and exposed to the same environment at the same time, which ensure that the results are fair.

Chapter 2

Methodology

This chapter describes the methodology that was used to run the experiments. It describes the environment in which the robot moved, the scenarios that were used to obtain the most valuable information and methods for evaluating the results. It also describes the OptiTrack system, which was used as a reference system. All experiments were conducted at the Wrocław University of Science and Technology.

2.1 Environment

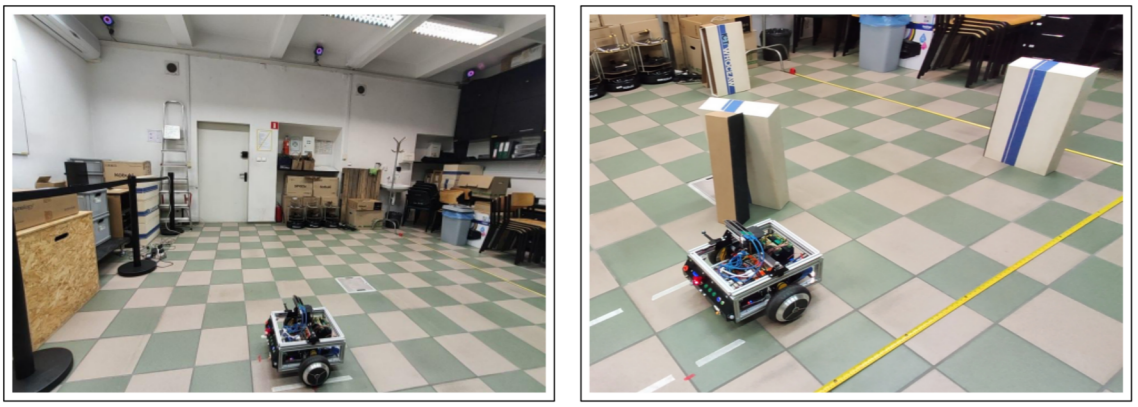
All the experiments were conducted inside the building, in a controlled area, in one of the classrooms. This room was chosen because it is equipped with the OptiTrack system. It has a tiled floor, which is not perfectly flat and a quite slippery surface. As visible in figure 2.1a this room has a lot of characteristic points which can be easily tracked by the VIO algorithm. In the middle of the room, there is a large, free space where the experiments took place. During some scenarios, additional obstacles were placed in this space, between which the robot was moving (without collisions), which is shown in the figure 2.1b. The environment was naturally static. There were no moving objects around. In this room, light can be almost eliminated, which made it possible to test the algorithms in different lighting conditions. Figure 2.1c shows a room with the lights off. When the lights are on, it is evenly distributed.

2.2 Scenarios

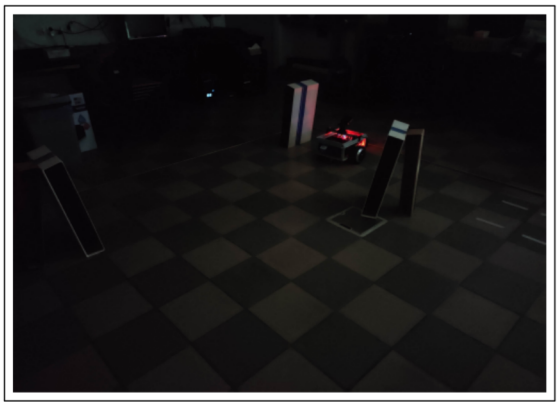
To eliminate the influence of a given trajectory, the robot moved more or less the same in all experiments. The task of the robot was to ride a path resembling a square with a side of 1.5m and return to its initial pose (x, y, θ). In order to obtain the most valuable results, the experiments were repeated for several scenarios:

1. Number of cameras: to check the impact of randomness in the operation of the cameras, three identical ReasSense T265 cameras were used during the experiments. To simplify the distinction between the cameras, they have been named as 'camera1', 'camera2' and 'camera3'. During one experiment, only two cameras could be used at the same time, so the experiments were carried out for set 'camera1' and 'camera2' as well as 'camera2' and 'camera3'.

2. Robot speed: to check the influence of robot speed, the experiments were conducted for a robot moving with a linear velocity equal to 0.2m/s ('slow') and 0.5m/s ('fast').



(a) (b)



(c)

Figure 2.1: The environment a) room with lights on and no obstacles, b) with obstacles, c) lights off and obstacles

The angular velocity was constant and equal to $\pi/4$ (≈ 0.79) rad/s.

3. Moving visual elements: the experiments were also conducted taking into account the nature of the environment. The results were compared for a stable environment, where there were no moving objects ('stable'), and for an environment with a large number of moving objects (one person moving around and waving cardboard in front of camera and lidar, 'dynamic').

4. Obstacles: also introduced a scenario in which there are several nearby obstacles on the path of the robot. These obstacles were put a few centimetres beside the robot path (no collisions). The used objects exceeded robot height, which means that they were visible to both the lidar and the cameras.

5. Lightning conditions: the effect of light on the results was also examined. The experiments were carried out for ideal light conditions (evenly distributed light, 'good') and for unfavourable light conditions (almost dark, only a few bright points, 'poor').

6. Camera position: the influence of the camera position on the quality of the localisation data was also measured. The camera was set in two positions: 45 degrees to the ground ('45 deg') and 90 degrees ('90 deg') to the ground when the cameras could see the robot frame.

7. External disturbance to the robot's movement: an experiment was also conducted to check the impact of external disturbances to the robot's movement on the quality of the location. The robot was pushed while driving. This experiment was designed to simulate uneven terrain and skidding.

2.3 Ground truth

To correctly compare the localization methods, a reference system is needed, which can be considered as a system that provides fully correct information (ground truth). During the experiments, the OptiTrack system was used. It is a motion capture system, that is capable to track the object with very high accuracy. It consistently produces positional error less than 0.3mm and rotational error less than 0.05°, with is sufficient to treat it as a reference. OptiTrack system is easy to use to track rigid body position. It delivers the ROS node that publishes the message with the actual position. The entire configuration with the robot is based on attaching 6 markers to the robot's housing. These markets are then tracked by cameras placed around the area where the robot moves. Figure 2.2 shows the robot setup during the experiments (the white balls are the OptiTrack markers).

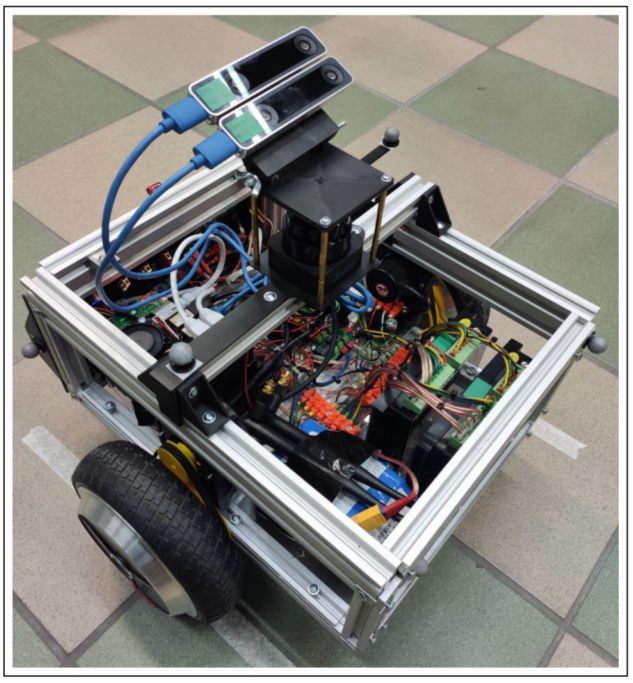


Figure 2.2: Robot setup during the experiments

2.4 Evaluation

All the odometry data from the robot and positions from the OptiTrack system were recorded using the rosbag package. After the experiment finished all data were exported to CSV files and processed using custom software written in Python. The directly recorded data series were inconsistent with each other and therefore required some pre-processing steps to be carried out. To compare them fairly, they must be matched with each other. Figure 2.3 shows the key steps that have been done for every data set:

(a) This figure shows the raw data. It is visible that those data contains many artefacts that must be eliminated before comparison of the methods. The artefacts result from the fact that each tested localization method has its characteristics. The figure also shows the way how the robot moved during the experiments. It is a square with a 1.5m side. There is some discrepancy between the starting point and the endpoint (robot doesn't back perfectly to initial position), however, it is irrelevant to the method comparison. The most important is trajectory itself.

(b) In the first step, the paths generated by the cameras are transformed. This is necessary since during the experiments the cameras were not mounted perfectly on the axis of the robot. To solve this problem, a translation in the direction of the robot movement is applied to the data. The places marked with red circles show what has changed. The arcs present during rotation have been eliminated.

(c) The next step is related to resampling. Each algorithm generates data at a different frequency. To be able to compare the paths, it is necessary to align the data to one size. The data are aligned to the method which generates the poses with the lowest frequency (RF2O, 10Hz). It means that a 60-second data set contains 600 localization samples for each method. This is enough data to be able to compare the methods.

(d) This figure shows the relationship between the original and the resampled data. It can be observed that the characteristic courses of the paths have been mapped well. The loss of data is acceptable.

(e) The next step in data processing is to eliminate parts of the paths in which the robot was not moving. These are the starting and ending points that refer to the moments between the start of data recording and the start of the robot movement, as well as the end of the robot movement and the end of data recording. This figure shows the region where the robot starts and finish its movement. One may notice that excess data has been deleted.

(f) The last step is to align all paths to the starting position. All paths are translated and rotated, so the first point is moved to (0, 0) position with a 0-degree rotation. This step ensures that the data is not distorted and the performance metrics work correctly.

The shown preprocessing procedure was repeated for all data for all scenarios. The figures shown in the 3 chapter are already preprocessed and the whole procedure is not shown again.

To compare the localization methods several metrics were used to quantify the most important aspects of performance. An overview of these metrics is shown in table 2.1, where:

- L_O – the length of trajectory generated by OptiTrack,
- L – the length of trajectory generated by compared localization method,
- d_t – the Euclidean distance between point generated by compared localization method and point generated by OptiTrack in time instance t,

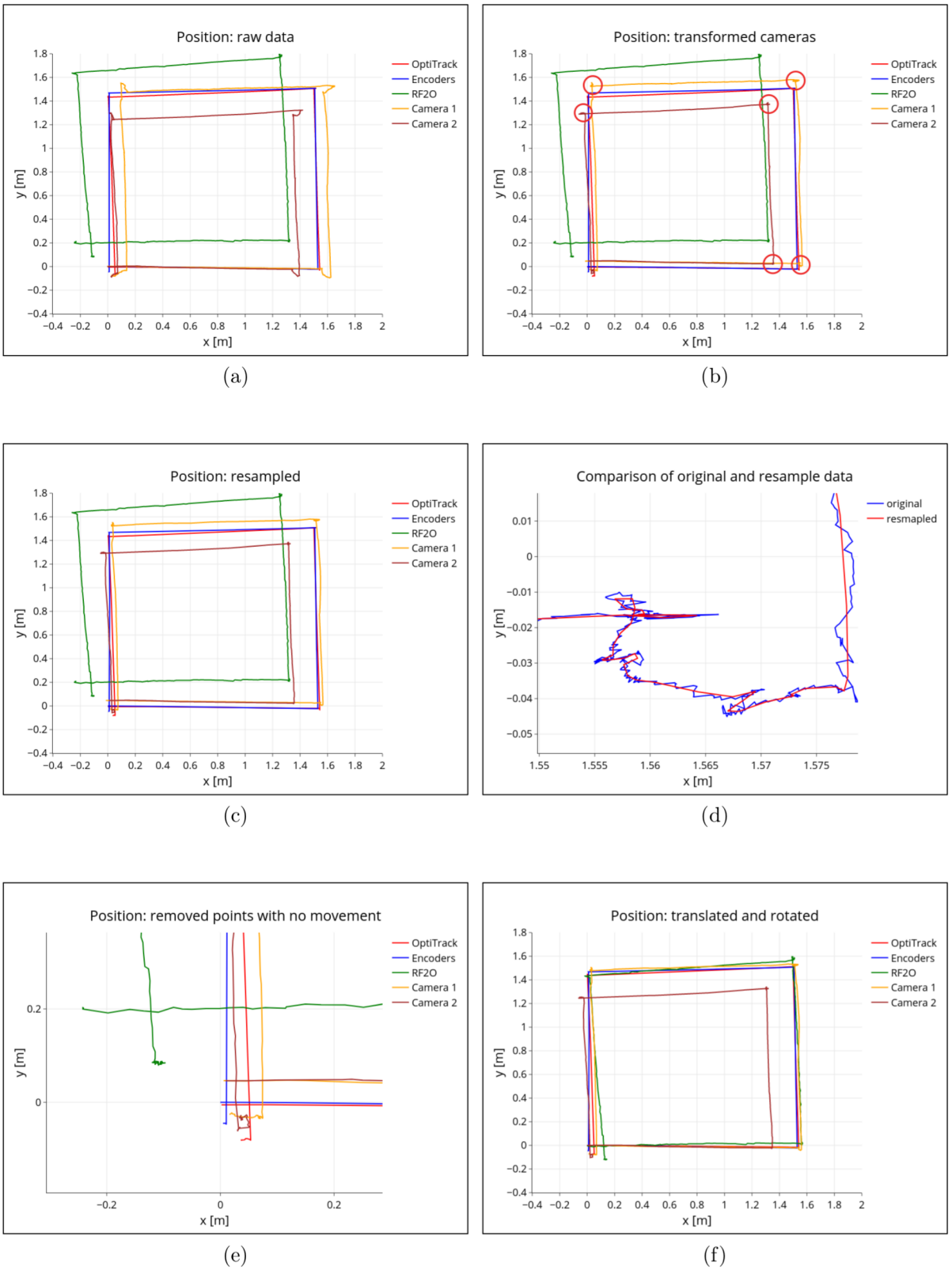


Figure 2.3: Preprocessing steps: a) raw data, b) cameras transformation, c) adjusting to initial position, d) resampling, e) resampling magnified, f) final data set

- N – the number of points for each path,
- Θ_t – the angle difference between angle generated by compared localization method

and point generated by OptiTrack in time instance t,

All the distances between two points were computed as a Euclidean distance in Euclidean space. The path length L was computed as the sum of distances between every two consecutive samples. All distances where the instantaneous speed between the points was more than 1m/s were excluded from this sum, to minimise the measurement error related to the re-localization of the VI-SLAM algorithm, which could cause overestimation. The speed was calculated using the equation $\frac{d_t}{t-t_{-1}}$, where $t-t_{-1}$ is the time difference between two time instances for which the d_t distance was calculated. Similarly for speed error.

Table 2.1: Performance metrics overview

Metric	Unit	Decription	Equation
Path Length Error (PLE)	%	Relative difference between length of a path generated by compared method and OptiTrack	$\frac{L_O-L}{L}$
Translation Error (TE)	m	Root Mean Square Error (RMSE) of distances between points generated by compared method and OptiTrack	$\sqrt{\frac{1}{N}\sum_{t=0}^{N}(d_i)^2}$
Translation Speed Error (TSE)	%	Root Mean Square Error (RMSE) of distances speed between two points generated by compared method and OptiTrack	$\frac{d_t}{t-t_{-1}}$
Final Translation Error (FTE)	m	Absolute Euclidean distance between final point generated by compared method and OptiTrack	$\lvert d_N \rvert$
Translation Drift (TD)	%	Distance between final point generated by compared method and OptiTrack relative to path length	$\frac{\lvert d_N \rvert}{L}$
Rotation Error (RE)	°	Root Mean Square Error (RMSE) of angle difference between points generated by compared method and OptiTrack	$\sqrt{\frac{1}{N}\sum_{t=0}^{N}(\Theta_i)^2}$
Final Rotation Error (FRE)	°	Difference between final angle for compared method and OptiTrack	$\lvert \Theta_N \rvert$

Chapter 3

Data Analysis and Results

This chapter contains a detailed presentation and discussion of data analysis and the results of this study. Each section describes an experiment that was conducted to validate the behaviour of the localization methods described in chapter **??**. The major method to be compared is the Visual-Inertial Odometry method, which is delivered with RealSense T265 cameras. To minimize the influence of the random factor on the results of the experiments, three identical cameras were used during the experiments. The data obtained from this method were compared with two more traditional approaches: odometry from encoders and odometry from the RF2O method, which uses the lidar. As ground truth, the OptiTrack motion capture system was used. It delivers very accurate data and can be considered as the reference one. Two approaches were used to evaluate the results. The first one is a graphical presentation of the trajectories. It gives the possibility to compare the methods visually, so one can easily observe the behaviour of given methods under various conditions. The second method is to use the metrics described in chapter 2. They allow to compare evaluated localization algorithms quantitatively, not just visually. All statistical analysis was done using the custom Python script. The description of each experiment contains information about the purpose for which it was performed, what was the scenario and what are the observations and conclusions.

3.1 Experiment 1: reference, perfect conditions

Scenario:

- Cameras used: camera1, camera2
- Robot speed: slow
- Obstacles: no
- Environment: stable
- Lighting: good
- External disturbances: no
- Cameras position: 45 deg

The first experiment was carried out to test how the methods work in a theoretically perfect environment. The results obtained during this experiment can be treated as a

reference for subsequent experiments. The robot was moving with 0.2m/s linear and $45°/s$ angular velocity, which minimises the skids during its motion, especially during the start-up acceleration. During the movement of the robot, there were not any dynamic objects and obstacles around. The light conditions were very good (evenly distributed artificial light). The cameras were rotated 45 degrees to the ground, so the robot was not visible to them. Figures 3.1a-c shows the results in visual form. All those trajectories are already preprocessed using the procedure described in section 2. Figure 3.1a presents the odometry coordinates on a plane. Figure 3.1b shows the coordinated and the rotation values over time. Time 0 corresponds to the moment when the robot started its movement, and the last measurement was made when both wheels of the robot stopped (50.9 seconds in this case). Figure 3.1c is about the absolute pose error over time. The error is calculated as an absolute difference between each method pose and the reference method (OptiTrack).

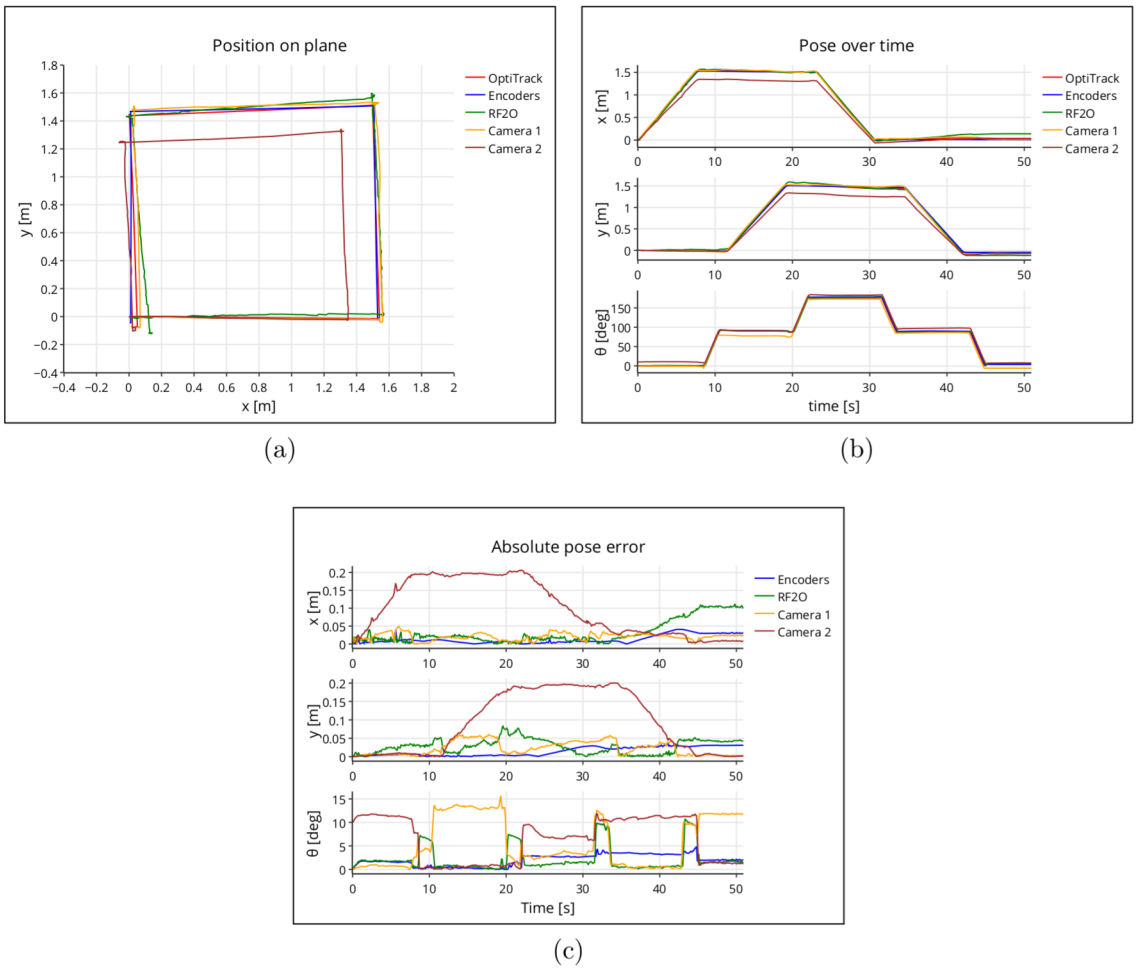


Figure 3.1: Experiment 1: a) trajectories on plane, b) pose over time, c) absolute error over time

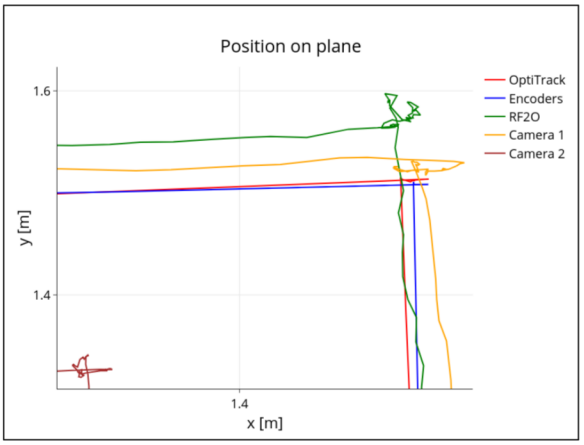
The first thing to notice is that there is a significant difference in the data obtained from the two cameras. The trajectory obtained from camera 2 differs significantly from the rest of the results. It also has visible position errors when the robot is rotating. Additionally, it can be seen that the position error generated by camera 2 is very deterministic. Both in the x-axis and y-axis, the error is equal and amounts to approximately 20 centimetres, which would mean that this camera generates a stable error during its operation. From the graph showing the rotation error over time, it can be seen that both cameras generate

Table 3.1: Experiment 1: performance metrics

Metric	Encoders	RF2O	Cam1	Cam2	Unit
Path Length Error (PLE)	-1.85	9.72	5.84	-7.43	%
Translation Speed Error (TSE)	0.02	0.05	0.04	0.03	m/s
Translation Error (TE)	0.03	0.06	0.04	0.18	m
Final Translation Error (FTE)	0.04	0.11	0.02	0.01	m
Translation Drift (TD)	0.64	1.76	0.32	0.16	%
Rotation Error (RE)	2.34	3.32	7.75	7.93	°
Final Rotation Error (FRE)	2.07	1.61	11.88	1.22	°

significant errors. In the case of camera 1, the error is over 15 degrees in peak. The trajectory generated by the RF2O algorithm has a visible accumulation of error over time. The biggest error was generated in the last phase of the movement (between 35 and 51 seconds). The error in the x-axis is greater than the error in the y axis. The trajectory generated by wheel odometry is the closest to the reference trajectory.

Table 3.1 shows the results obtained from metrics. The obtained results show new observations. The largest error in the length of the generated trajectory was recorded for the RF2O method. A positive error value means that the path is longer than the reference by 9.72% (approximately 60 centimetres), which is not visible in the graphs. This is because this method generates very unstable results when the robot is stationary and during rotation. Figure 3.2 shows a close-up of the trajectories generated during rotation. For both the LIDAR-based and camera-based methods, these errors are large.



(a)

Figure 3.2: Experiment 1: magnification of the trajectories generated during the robot's rotation

Camera 2 generated a track 7.43% shorter than the reference track which is in line with the observations based on the graphs. Translation Speed Error is similar for each method and ranges from 0.02-0.05m/s. The Root Mean Square Error (TL) for camera2 is greatest for both rotation and translation, however, Final Translation Error is the largest for the RF2O method. Comparing the TL and FTE errors, it can be observed that even though camera 2 generates the highest mean error, the final trajectory error is not that big, which confirms the deterministic error of this camera. This is also evident for the TD meter, which indicates that the drift of the RD2O method is much greater than that of the VIO

method. The last metric, FRE shows that the Final Rotation Error was the highest for camera1, which is also visible on the graphs.

Based on the observations, it can be concluded that despite the use of two identical cameras, one of them generates completely different data from the other one. This can be caused by damage to the camera, incorrect calibration, or random error. Also, both the cameras generate huge rotation errors. Moreover, it can be concluded that the RF2O method exhibit the greatest drift, while the data obtained from the encoders are the closest to that obtained from the reference method (under ideal surface conditions).

3.2 Experiment 2: used cameras

Scenario:

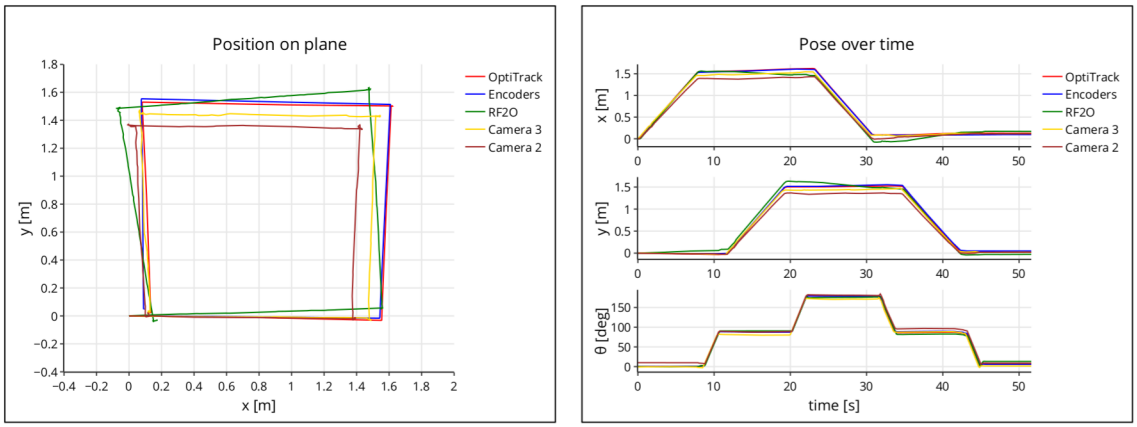
- **Cameras used:** camera2, camera3
- Robot speed: slow
- Obstacles: no
- Environment: stable
- Lighting: good
- External disturbances: no
- Cameras position: 45 deg

This experiment was to check if the behaviour recorded for camera 2 during the first experiment would be reproducible and what would be the results for the new camera (camera1 was replaced with a new camera [camera3]). This experiment was performed for the same scenario as in the case of experiment number 1.

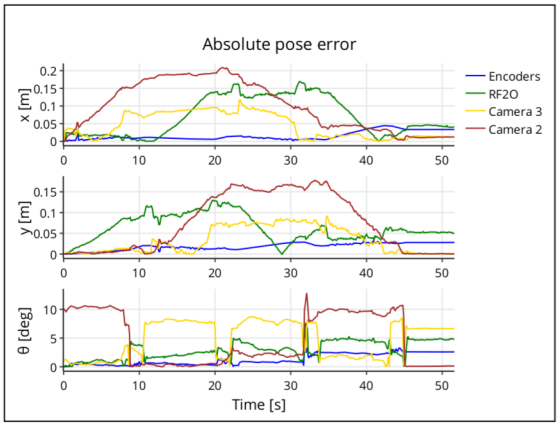
Table 3.2: Experiment 2: performance metrics. Camera1 replaced with Camera3

Metric	Encoders	RF2O	Cam3	Cam2	Unit
Path Length Error (PLE)	-1.03	6.31	0.73	-4.4	%
Translation Speed Error (TSE)	0.01	0.03	0.04	0.02	m/s
Translation Error (TE)	0.03	0.11	0.07	0.16	m
Final Translation Error (FTE)	0.04	0.06	0.01	0.01	m
Translation Drift (TD)	0.65	0.97	0.16	0.16	%
Rotation Error (RE)	1.63	3.33	5.79	6.48	°
Final Rotation Error (FRE)	2.59	4.78	6.65	0.15	°

The T265 sensors are calibrated by the manufacturer during production, so the measurements should be reproducible for each sample, however the results obtained in this experiment are not consistent with this. When observing both the figure 3.3a-c and the data in the table 3.2, it can be noticed that the characteristics have repeated. The trajectory generated by camera 2 is fully adequate for experiment 1. This means that the error generated by this camera is constant and deterministic. The new sensor (camera3) also showed this behaviour, however, the error it generates is different. Unfortunately, this



(a) (b)



(c)

Figure 3.3: Experiment 2: a) trajectories on plane, b) pose over time, c) absolute error over time. Camera1 replaced with Camera3

observation can be critical for many applications, because in this case, it is impossible to predict what would be the error generated by the used camera and each camera would require separate measurements and transformations. A similar experiment was carried out several more times, both for the camera1+camera2 and camera2+camera3 setup. The results repeated, which excludes the influence of a random factor on the presented results. It is also worth noting that the results obtained with the encoder-based and RF2O method show the same behaviour as before.
For subsequent experiments the robot configuration with camera1 and camera2 was used. They show the greatest discrepancy, which may bring the most interesting research results.

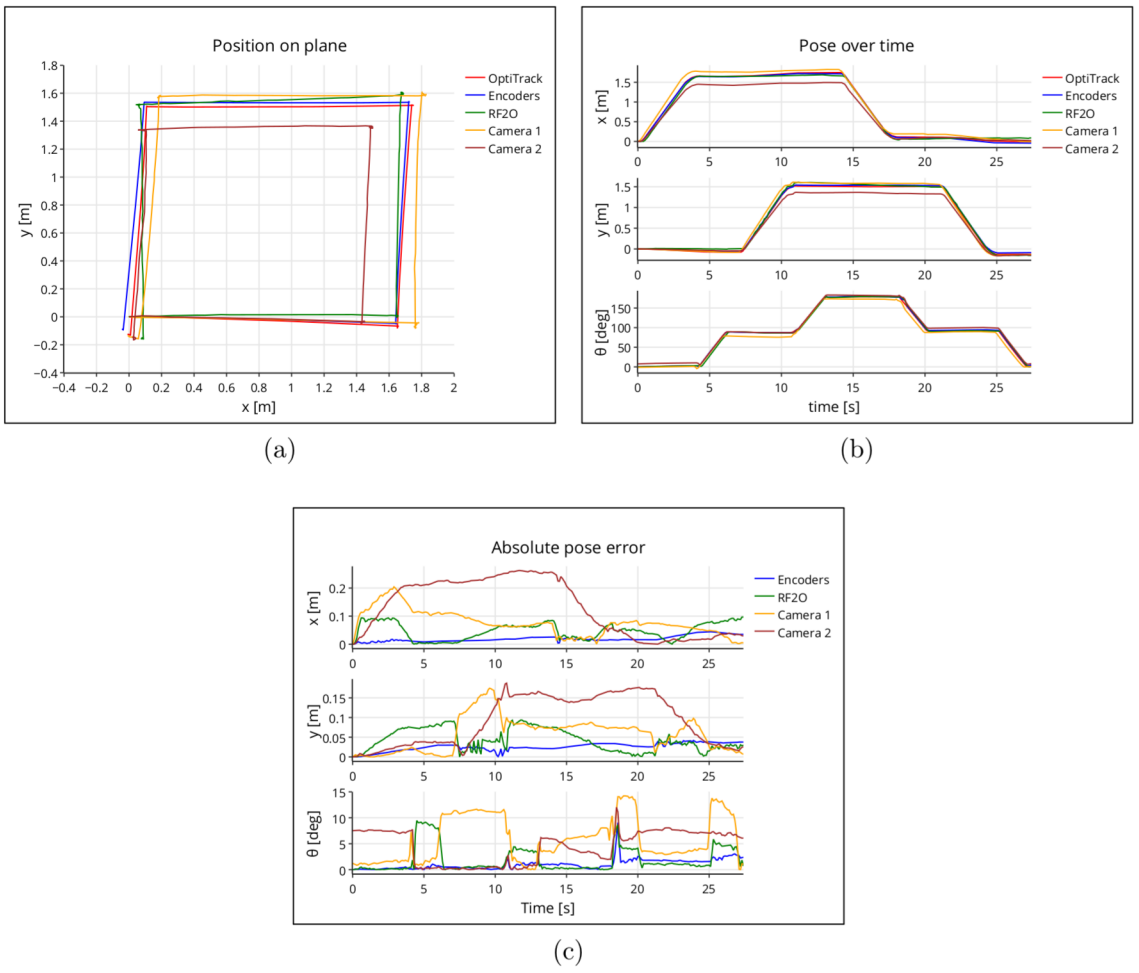
3.3 Experiment 3: robot speed

Scenario:

- Cameras used: camera1, camera2
- **Robot speed:** fast
- Obstacles: no

- Environment: stable
- Lighting: good
- External disturbances: no
- Cameras position: 45 deg

This experiment was carried out to verify the influence of the robot speed on the results generated by the compared methods. During the experiment, the linear velocity of the robot was changed from 0.2m/s to 0.5m/s (it also increases the acceleration). The remaining elements were left as in the case of experiment 1. Figures 3.4a-c and table 3.3 presents the results.



(a) (b)

(c)

Figure 3.4: Experiment 3: a) trajectories on plane, b) pose over time, c) absolute error over time. Linear velocity increased to 0.5m/s

From the graphs it can be seen that the camera1 changed its characteristics at higher speeds compared to the results from experiment 1. This time the generated trajectory is overestimated. It is also visible in the PLE meter, which informs that the length of the generated path is more than 8% longer than the reference path. Moreover, camera2 produced almost the same result but with the reverse sign. Besides, the characteristics of the cameras look very similar to the previous one. Looking at the TSE meter, it can be observed that increasing the speed of the robot had a negative impact on the quality

Table 3.3: Experiment 3: performance metrics. Linear velocity increased to 0.5m/s

Metric	Encoders	RF2O	Cam1	Cam2	Unit
Path Length Error (PLE)	-1.06	4.99	8.14	-8.44	%
Translation Speed Error (TSE)	0.04	0.11	0.08	0.05	m/s
Translation Error (TE)	0.04	0.08	0.11	0.2	m
Final Translation Error (FTE)	0.05	0.1	0.01	0.04	m
Translation Drift (TD)	0.75	1.51	0.15	0.6	%
Rotation Error (RE)	1.45	3.07	7.29	5.45	°
Final Rotation Error (FRE)	2.39	0.7	1.06	6.1	°

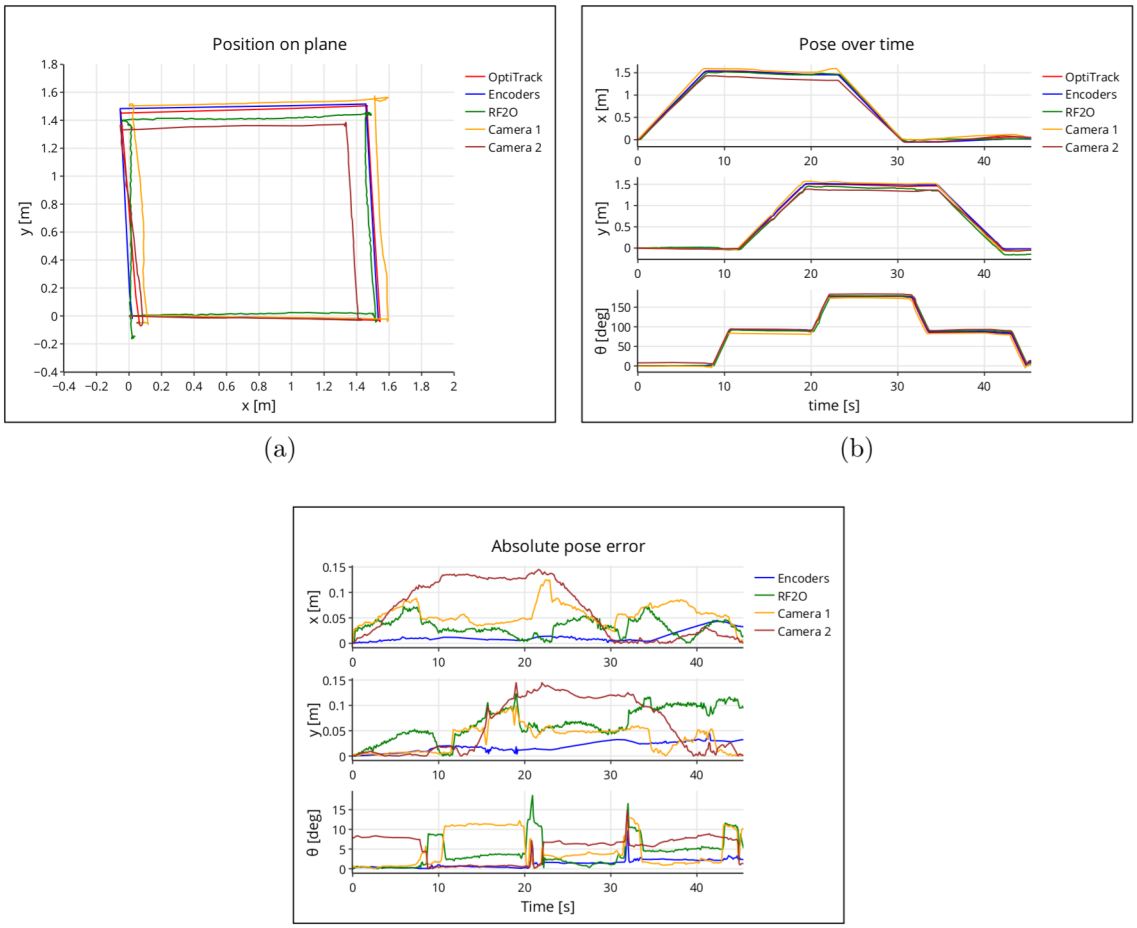
of speed measurements by all the methods. The error of each method has increased. The same situation is in the case of RMSE translation. It is also worth noting that increasing the speed negatively affected the results obtained from the encoder odometry. Greater drift is visible, which may be caused by wheel slippage at accelerations higher than before. Summarising this experiment, it can be concluded that the increase in the speed of the robot's movement negatively affects the quality of the location obtained from the considered methods. Between experiment 1 and this experiment, the difference in speed is only 0.3m/s, and there is already a difference that can increase at even higher speeds. 0.5m/s (1.8km/h) is commonly not a high speed for mobile robots, especially for aerial robots (e.g. drones). Increasing the measurement error of T265 cameras with the increasing speed of the object on which they are mounted may disqualify them from use in some applications.

3.4 Experiment 4: close obstacles

Scenario:

- Cameras used: camera1, camera2
- Robot speed: slow
- Environment: stable
- **Obstacles:** yes
- Lighting: good
- External disturbances: no
- Cameras position: 45 deg

This experiment aimed to check whether the presence of nearby obstacles influenced the results generated by the VIO and RF2O methods. During the experiment, several cardboard boxes were used that were placed along the path of the robot's movement(as shown in figure 2.1c). These cardboard boxes were higher than the robot, so they were visible to both the cameras and the lidar. The rest of the scenario was the same as in experiment 1. The results are presented in the figures 3.5a-c and in table 3.4.
Based on the obtained results, it can be concluded that the presence of nearby, stationary obstacles has a greater impact on the RF2O method than on the results obtained from the Visual-Inertial Odometry algorithm. The graphs show a deterioration in the quality



(a) (b)

(c)

Figure 3.5: Experiment 4: a) trajectories on plane, b) pose over time, c) absolute error over time. Added nearby obstacles

Table 3.4: Experiment 4: performance metrics. Added nearby obstacles

Metric	Encoders	RF2O	Cam1	Cam2	Unit
Path Length Error (PLE)	-1.32	8.13	8.3	-4.09	%
Translation Speed Error (TSE)	0.02	0.05	0.04	0.03	m/s
Translation Error (TE)	0.03	0.08	0.07	0.12	m
Final Translation Error (FTE)	0.05	0.1	0.01	0.0	m
Translation Drift (TD)	0.81	1.61	0.16	0.0	%
Rotation Error (RE)	1.68	5.24	6.38	6.07	°
Final Rotation Error (FRE)	2.33	5.24	10.0	1.35	°

of the RF2O method both on straight lines and while rotating. The generated paths are much less stable than before. The translation errors read from the meters do not indicate much difference, while the rotation errors have increased. This behaviour may indicate that the presence of nearby elements noises the results of the lidar-based algorithm. This conclusion may be disqualifying for this method in the case of work, for example, in narrow corridors. A small difference is also seen in the case of camera 1. As in experiment 3, an over-estimation of the distance can be observed. Moreover, in this case, large discrepancies are visible at the time of rotating. This means that the cameras may behave unpredictably

under certain conditions. The presence of nearby obstructions does not affect the encoder odometry.

3.5 Experiment 5: environment characteristic

Scenario:

- Cameras used: camera1, camera2
- Robot speed: slow
- **Obstacles:** yes
- **Environment:** dynamic
- Lighting: good
- External disturbances: no
- Cameras position: 45 deg

This experiment was devoted to checking the impact of the behaviour of the environment in which the robot operates on the quality of the data from localization algorithms. The environment has been made dynamic by using obstacles like in experiment 4 and additional moving objects in the form of a person walking around the robot and waving cardboard at the height of the cameras and lidar. This scenario was designed to simulate environments in which there is continual motion around the robot (for example, multi-robot systems or collaborative robots). The results of this experiment are presented in the figures 3.6a-c charts and table 3.5.

Table 3.5: Experiment 5: performance metrics. Dynamic environment

Metric	Encoders	RF2O	Cam1	Cam2	Unit
Path Length Error (PLE)	-1.23	5.64	6.57	-9.42	%
Translation Speed Error (TSE)	0.02	0.07	0.06	0.05	m/s
Translation Error (TE)	0.02	0.18	0.16	0.2	m
Final Translation Error (FTE)	0.04	0.09	0.02	0.14	m
Translation Drift (TD)	0.64	1.45	0.32	2.26	%
Rotation Error (RE)	1.45	8.25	6.92	5.4	°
Final Rotation Error (FRE)	2.38	10.87	8.55	1.86	°

The results obtained during this experiment shows that the dynamic environment in which the robot moves negatively affects the VIO and RF2O methods. Each of these trajectories indicates a loss of quality. This is especially visible for the RF2O and camera1 results. This is also confirmed by the RMSE measurement, which increased significantly for both trajectories comparing to experiment 1. Additionally, during this experiment, the effect of the re-localization algorithm of the camera1 was observed. It is visible in the last phase of the robot movement. The moment of re-localization is marked with a red arrow in the 3.6d figure. This enables the VI-SLAM algorithm to recognise a place where the robot was previously and update the position to the correct one if the measurement error has accumulated since its last stay there.

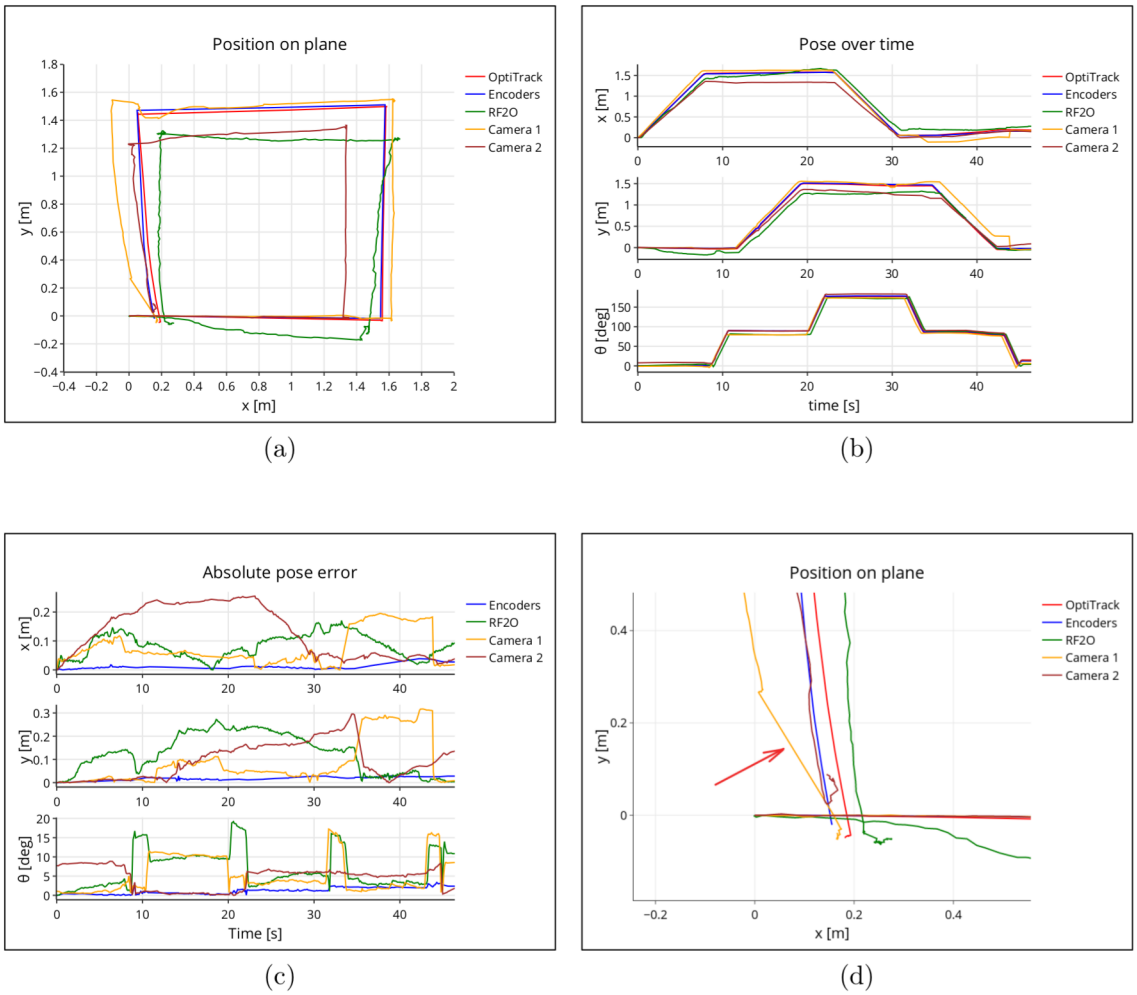


Figure 3.6: Experiment 5: a) trajectories on plane, b) pose over time, c) absolute error over time. Dynamic environment

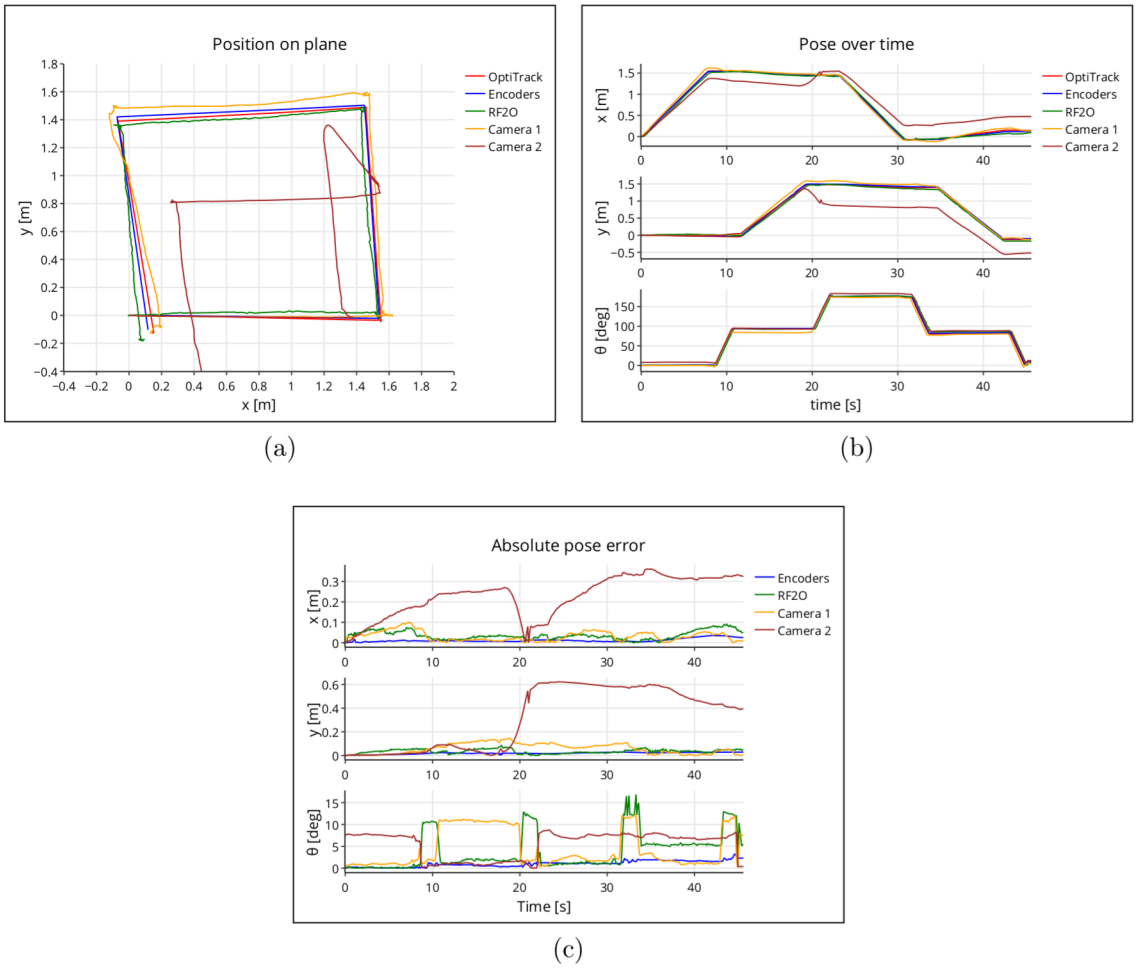
3.6 Experiment 6: lighting conditions

Scenario:

- Cameras used: camera1, camera2
- Robot speed: slow
- **Obstacles:** yes
- Environment: static
- **Lighting:** poor
- External disturbances: no
- Cameras position: 45 deg

This experiment investigated the effect of lighting quality on location quality. The idea behind this experiment was to simulate the changing light conditions during the robot's operation, which may occur when the robot, for example, travels from one room to another, with different conditions in both. Such conditions were simulated by turning off the

lights 2 seconds after the robots started to move, so the methods could initially localise the robot in perfect light conditions and then the conditions changed for the rest time of robot movement. There were only a few light-emitting elements in the surroundings of the robot: LEDs on OptiTrack cameras, one monitor and several gaps from behind the window shades and the doors. The obstacles have been left in positions as in experiments 4 and 5. The results are shown in the figure 3.7 and in table 3.7



(a) (b)

(c)

Figure 3.7: Experiment 6: a) trajectories on plane, b) pose over time, c) absolute error over time. Changed light conditions

Table 3.6: Experiment 6: performance metrics. Changed light conditions

Metric	Encoders	RF2O	Cam1	Cam2	Unit
Path Length Error (PLE)	-1.17	9.23	10.47	3.47	%
Translation Speed Error (TSE)	0.02	0.06	0.04	0.09	m/s
Translation Error (TE)	0.03	0.06	0.08	0.48	m
Final Translation Error (FTE)	0.04	0.07	0.01	0.51	m
Translation Drift (TD)	0.64	1.12	0.16	8.19	%
Rotation Error (RE)	1.21	5.59	6.08	6.17	°
Final Rotation Error (FRE)	2.31	5.43	7.51	0.38	°

The obtained results indicate that the quality of lighting has a huge impact on the location determined by RealSense cameras. It is related to the characteristics of the VIO

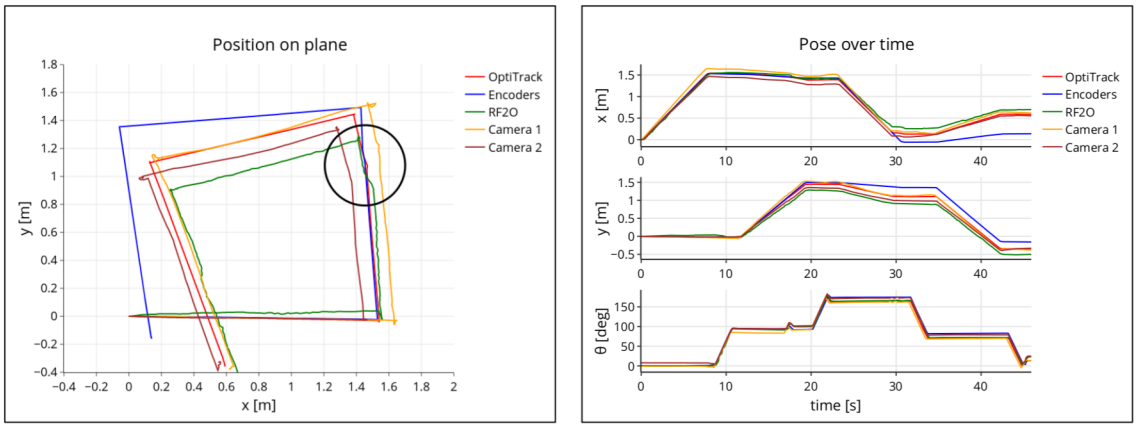
algorithm, which is based on image processing. It is visible in the graphs, as well as in metrics, that the results from both cameras deteriorated. Moreover, camera2 got lost completely its localisation. At the furthest point, the camera2 estimate differed from the reference method by almost 63 centimetres. The RMSE showed an error of 0.48m and the translation drift was 8.19%. Additionally, one can observe that the rotation errors between both cameras do not differ much. It would seem that camera2 should accumulate a much greater error than camera1, which generated a better trajectory. This behaviour is because the camera rotation is generated mainly based on the results obtained from the IMU device and the lighting has no major impact on the rotation results. Moreover, it can be observed that the PLE and TSE meters do not show any significant deviation. This means that the camera correctly detected that it was moving, however, it misjudged where it was located. Again, the reason is the fusion of the Visual Odometry algorithm with data from the Inertial Measurement Unit. Another conclusion is that the cameras behave unpredictably, which confirms the conclusions from previous experiments. Only one of the cameras lost its location while the other camera did well under the same conditions. The obtained results indicate that the cameras can not be used in environments with poor or no light. The situation is different in the case of the RF2O. As the lidar is an active device (it emits a light signal itself), the lack of light in the environment does not disturb its operation.

3.7 Experiment 7: external disturbances

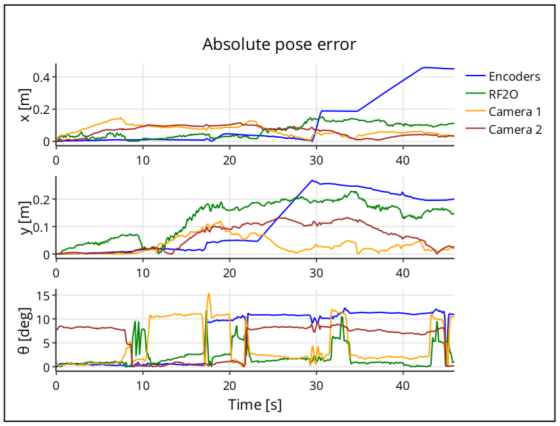
Scenario:

- Cameras used: camera1, camera2
- Robot speed: slow
- **Obstacles:** yes
- Environment: static
- Lighting: good
- **External disturbances:** yes
- Cameras position: 45 deg

During all the previous experiments, the robot was moving on at an almost perfectly flat surface with quite good adhesion to the wheels. This resulted in very good results generated by the odometry algorithm based on encoder data. In a large number of cases, robots must move to different surfaces (not always perfectly flat). For example, almost all wheeled outdoor robots have to deal with unevenness (grass, stones, sand) that cause slippage. This experiment aimed to simulate a slippage while the robot is moving to be able to observe how it affects the location outputted by considered methods. The skid was simulated by forcefully pushing the robot while it was moving. The obstacles were still present around the robot. The pictures 3.8a-c shows the results obtained. The slip moment is marked in the figure 3.8a with a black circle. The table 3.7 shows the results of the metrics.



(a) (b)



(c)

Figure 3.8: Experiment 7: a) trajectories on plane, b) pose over time, c) absolute error over time. Added external disturbances

Table 3.7: Experiment 7: performance metrics. Added external disturbances

Metric	Encoders	RF2O	Cam1	Cam2	Unit
Path Length Error (PLE)	2.36	2.3	11.32	-0.36	%
Translation Speed Error (TSE)	0.03	0.06	0.04	0.03	m/s
Translation Error (TE)	0.25	0.17	0.09	0.11	m
Final Translation Error (FTE)	0.49	0.18	0.04	0.04	m
Translation Drift (TD)	8.22	3.02	0.67	0.67	%
Rotation Error (RE)	8.47	2.96	6.12	6.54	°
Final Rotation Error (FRE)	10.92	1.0	10.43	0.22	°

From the obtained results, it can be seen that the slippage was not detected by the encoder-based method, while all other methods adequately showed this disturbance. It is also very well visible in the values generated by almost all metrics: TE at 0.25m, the final shift of almost half a meter, drift of 8.22%, RMSE of rotation of 8.47 degrees and final rotation error at almost 11 degrees. The main conclusion from this experiment is to not use the encoder-based odometry localisation method for robots that are exposed to slippage or to fuse the obtained data with data from different method (as VIO) to compensate for the error, which is the most popular case in nowadays robotics.

3.8 Experiment 8: cameras position

Scenario:

- Cameras used: camera1, camera2
- Robot speed: slow
- Obstacles: no
- Environment: stable
- Lighting: good
- External disturbances: no
- **Cameras position:** 90 deg

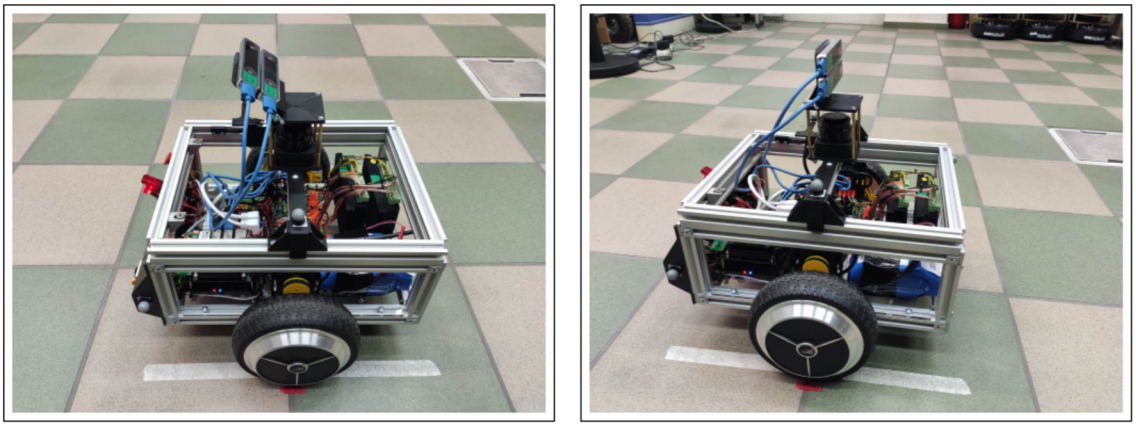
The purpose of this experiment was to find out how the camera angle affects the results generated by the VIO algorithm. In previous experiments, the cameras were positioned at an angle of 45 degrees to the ground, which prevented the cameras from seeing the front of the robot during their work. During this experiment, both cameras were rotated to a 90-degree position, which causes that the robot was a large part of each of the images captured by the cameras. In such a setup, the robot is the stationary element to the cameras. Figures 3.9a-d shows how the cameras were mounted and the image directly seen by the cameras for each of these cases (45 and 90 degrees).

This experiment was carried out under the same conditions as in the case of experiment 1. The results are shown in the figure 3.10a-c and the table 3.8.

Table 3.8: Experiment 8: performance metrics. Cameras rotated to 90 degrees

Metric	Encoders	RF2O	Cam1	Cam2	Unit
Path Length Error (PLE)	-1.24	6.09	-21.67	-26.55	%
Translation Speed Error (TSE)	0.01	0.04	0.14	0.1	m/s
Translation Error (TE)	0.03	0.06	0.71	0.53	m
Final Translation Error (FTE)	0.05	0.09	0.51	1.14	m
Translation Drift (TD)	0.8	1.44	8.18	18.29	%
Rotation Error (RE)	1.43	2.93	4.61	1.17	°
Final Rotation Error (FRE)	2.28	0.03	0.83	0.69	°

The obtained results indicate that the cameras do not work properly in this configuration. The result of this experiment was unexpected and the observed behaviour was a big surprise. In the documentation of the RealSense T265 cameras, there is no mention of the method of mounting the cameras on the robot, which suggests that the mounting is free to choose. As the experiment shows, assembly is very important for the quality of data generated by cameras. To eliminate the possibility that this behaviour is a one-time result, this test was performed with many different environment configurations. The result was the same in all cases. In addition, to eliminate the possibility that the behaviour was related to the room where the experiments took place, the results were consulted with other people who use T265 cameras for professional purposes. These people confirmed that the observed result is common. After additional research, it was discovered that the

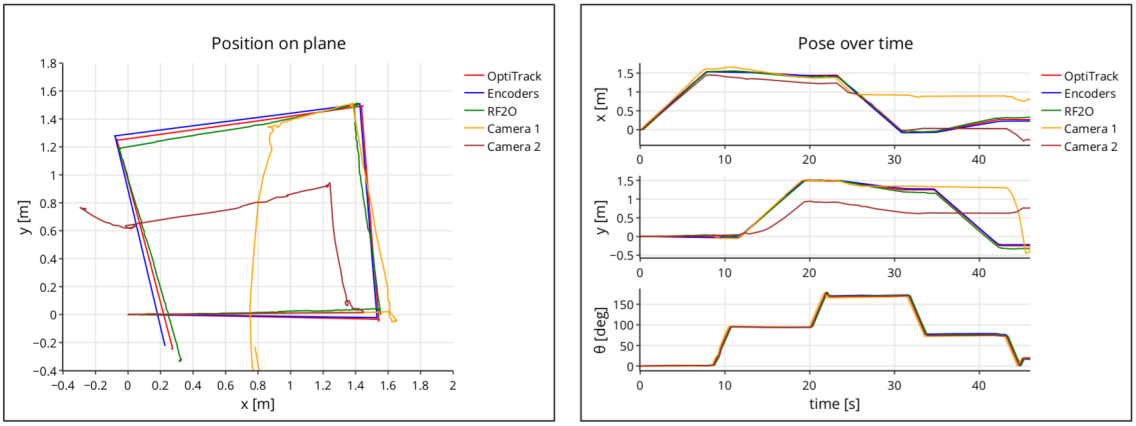


(a) (b)

(c) (d)

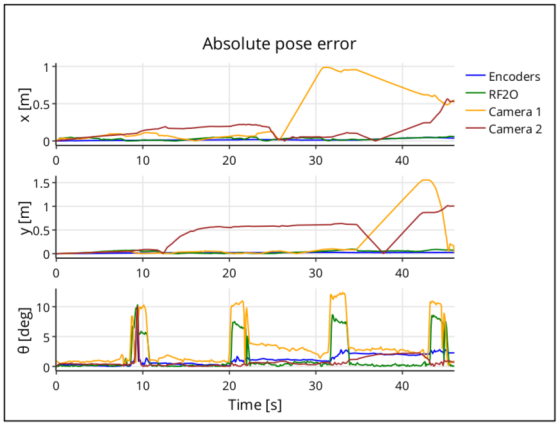
Figure 3.9: Experiment 8: a) robot with cameras rotated 45 degrees , b) robot iwth cameras rotated 90 degrees, c) view from RealSense camera when is rotated 45 degree, d) view from camera when is rotated 90 degree

optimal installation of the cameras used in the Visual Odometry algorithm (not only the RealSense case) is a place where none of the elements of the robot is directly visible. The errors obtained are the result of the normal work of the Visual Odometry algorithm. As described in section **??** the algorithm searches for features, which in the case of camera rotated 90 degrees, were chosen also from the image area occupied by the robot. During the feature tracking (or matching) phase, it generates errors because some points do not move according to others.



(a)

(b)



(c)

Figure 3.10: Experiment 8: a) trajectories on plane, b) pose over time, c) absolute error over time. Cameras rotated to 90 degrees