Deployment and commissioning of a group of robots on a dedicated local network

Second year internship report
FISE 2023
09/05/2022 - 29/08/2022

Damien ESNAULT
ENSTA Bretagne – Autonomous Robotics
damien.esnault@ensta-bretagne.org

Supervised by:

Prof. Dr.-Ing. habil. Andreas RAUH
Carl-von-Ossietzky Universität Oldenburg – Department of Computing Science
andreas.rauh@uni-oldenburg.de

M.Sc. Friederike BRUNS
Carl-von-Ossietzky Universität Oldenburg – Department of Embedded Hardware/Software Systems
friederike.bruns@uni-oldenburg.de

Table of contents

Abstract

As part of my engineering training, I did a 16-week engineering assistant internship (from 09/05 to 29/08) at the Computer Science Department of the University of Oldenburg. Supervised by Prof. Andreas RAUH, I deployed and commissioned a group of Turtlebot3 Burger on a dedicated local network. The objective of this project was twofold:

- To offer robotics students a new way to implement control algorithms and to introduce them to multi-agent environments.
- To offer PhD students in robotics a concrete solution to implement and develop regulation of multi-agent robotic systems.

During the first 3 months, I deployed the local network and prepared the Turtlebot3 Burger. Afterwards, I thought about a new hardware configuration that would be more modular to allow users to easily adapt the robots to their needs. I finally adapted my previous work for a multi-agent's configuration. The last month was devoted to writing documentation and studying the communication between the robots and programming environments such as MATLAB Simulink and Eclipse 4diac.

Résumé

Dans le cadre de ma formation d'ingénieur, j'ai effectué un stage d'assistant ingénieur de 16 semaines (du 09/05 au 29/08) au profit du département d'Informatique de l'université d'Oldenburg. Supervisé par le professeur Andreas RAUH, j'ai déployé et mis en service un groupe de Turtlebot3 Burger sur un réseau local dédié. L'objectif de ce projet était double :

- Proposer aux étudiant(e)s en robotique un nouveau support leur permettant d'appliquer leurs connaissances tout en les introduisant aux environnements multi-agents.
- Proposer aux doctorant(e)s en robotique, une solution concrète pour mettre en place et développer des algorithmes de contrôle pour les groupes de robots.

Durant les 3 premiers mois, j'ai tout d'abord déployé le réseau local et préparé les Turtlebot3 Burger. A l'issue, j'ai pensé à une nouvelle configuration matérielle plus modulable pour permettre aux utilisateurs d'adapter facilement les robots à leurs besoins. J'ai finalement adapté mes précédents travaux pour une configuration avec plusieurs agents. Le dernier mois fut consacré à l'écriture d'une documentation et à l'étude de la communication entre les robots et des environnements de programmation comme MATLAB Simulink et Eclipse 4diac.

Keywords

Multi-agent environment, Turtlebot3 Burger, Local Area Network, MATLAB Simulink, Eclipse 4daic, Education, Internship

Acknowledgements

First, I would like to thank **Prof. Luc JAULIN** (referent professor) and **Prof. Andreas RAUH** (main tutor of this internship) for giving me the opportunity to work on this project. This internship was very enriching from a personal point of view. I was able to get interested in the theme of robot groups, a theme that is close to my heart, while living my first professional experience in an international context.

I thank **Friederike BRUNS** (doctoral student at the University of Oldenburg and co-supervisor of this internship) for welcoming me and helping me to get my bearings when I arrived at OFFIS (research centre linked to the University of Oldenburg). As the centre is geographically far from the university (around 7 minutes by bike), it was reassuring to know that I had a point of contact directly at my workplace.

I would also like to thank the doctoral students at the University of Oldenburg with whom I was able to exchange and work during this internship. To name but a few: **Mahsa MOAZEZ**, **Marit LAHME**, **Oussama BENZINANE** and **Sven MEHLHOP**.

1. Introduction

1.1 Objectives and context

The multidisciplinary engineering training at *ENSTA Bretagne*, and more particularly the *Autonomous Robotics* course, allows students to develop an attraction for innovation and research. Through projects, courses, and practical work, they are pushed to perfect their knowledge and apply it into various systems and projects.

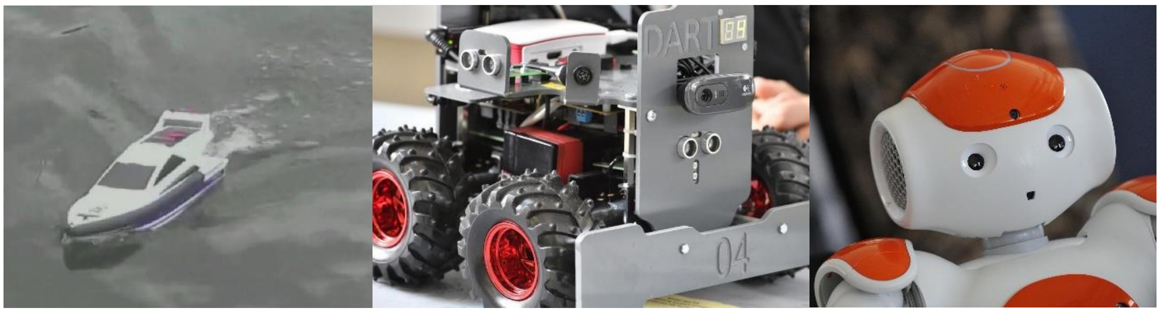


FIGURE 1 - Examples of the robots used at ENSTA Bretagne
(From left to right: DDBoat, DART and NAO)

The importance of practice in areas such as autonomous robotics justifies the need for a diverse range of robots designed for education. A university or school has several options for acquiring one: buying a dedicated robot from the market, adapting an existing system for educational use, or designing its own solution from scratch. The first solution is often expensive and restrictive, as it is limited in terms of use and therefore not versatile enough. The third solution, on the other hand, is the most versatile and least expensive, but requires too much time and resources to be deployed. The second solution is therefore the most advantageous one and this is also what **Prof. Andreas RAUH** of the *Computer Science department* at the *University of Oldenburg* concluded.

Its objective was to give university robotics students a new way to apply their knowledge. The main idea of this project was to deploy a group of small and modulable robots on a dedicated and easily deployable network, using R.O.S. to communicate and interact with the robots. This notion of modularity allows the project to be used in different ways. A first configuration can allow groups of students to work on one robot each, either in single-agent environment or in collaboration with robots from other groups. A second configuration can allow PhD students from the university to study and implement control strategies for multi-agent robotics systems. These two configurations are illustrated in Figure 2 below.

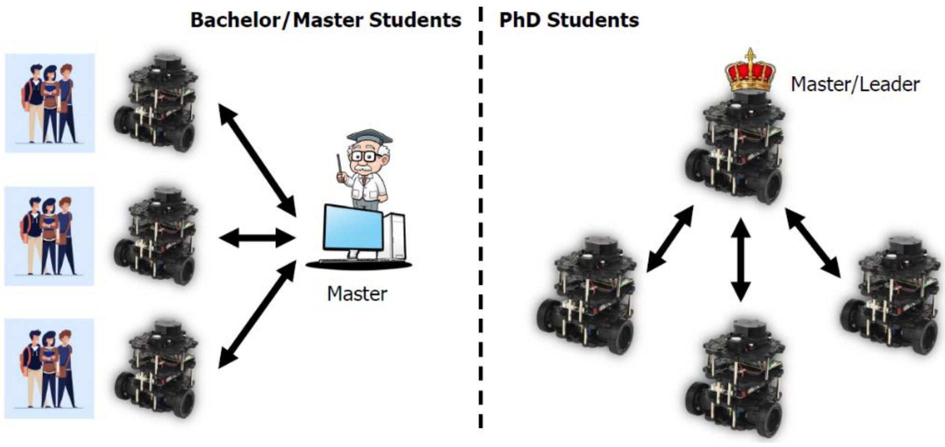


FIGURE 2 – Illustration of the use cases of the project

My job was to make this project a reality to offer a new educational support to the university's teachers. The project was initially divided into 4 main steps:

- Deploying a single-agent configuration: The first step was to build the robots and configure them, deploy the local network and check that the robots could work in their nominal configuration, i.e. with only one robot connected to the network
- Upgrading the hardware structure: The second step was to add new sensors to the robots and to think about a hardware configuration that would allow a high modularity of the sensors on the robot
- Evolving to a multi-agent configuration: The next step was to adapt my previous work so that the robots could operate while all being connected to the network at the same time
- Redacting a documentation: The last and probably most important step was to write documentation and user guides to allow teachers to use my work in the next semester.

Having completed these steps in 3 months, I then worked on a complementary step during the last month of the internship:

- Working on the communication between software and R.O.S: I checked the compatibility of the programming environments used by the university with R.O.S. In case of incompatibility, I thought of a solution allowing the communication between the software and ROS.

To carry out this project, funds were unlocked by the university to purchase four *Turtlebot3 Burger*, a mobile robotics platform designed for education and learning. After the purchase of the equipment, **Prof. RAUH** contacted **Prof. Luc JAULIN** to propose to a student from *ENSTA Bretagne* to carry out this project as part of a second-year internship. As an IETA, my career will progressively evolve from technical to project management. Therefore, the opportunities to work on such a project are very interesting for me. After accepting the internship, I was gradually able to discover the context of this internship.

First, the internship took place in Oldenburg, Germany, a country where I do not speak the language and know even less about the culture. This experience made me realise how difficult everyday life is for someone who does not speak the language of the country. Finding accommodation, shopping, or ordering in a restaurant were all challenges for me. It is precisely to simplify my daily life that I decided to learn German during the first weeks of my internship.

During these 4 months, I worked at *OFFIS*, the research centre linked to the University of Oldenburg. *OFFIS* is a private research centre with 6 research areas ranging from *Energy* to *Health* and with more than 250 employees who can be engineers, PhD students or professors coming from all possible backgrounds. Each branch is divided on average into 5 divisions or sub-divisions. In my case, I was in the *Manufacturing* branch in the *Distributed Computing and Communication* division. This experience allowed me to work in an international team with many people, using various tools to communicate and exchange. As Germany is still applying health measures to combat Covid-19, teleworking was widely used at *OFFIS*, which made it difficult for me to integrate during the first weeks. Indeed, many people did not know whether I was a newcomer or just a team member returning after a long period of teleworking.

Working at *OFFIS* was also interesting in terms of personal discipline. Indeed, *OFFIS* is located 7min by bike from the university. This meant that for most of my internship I was physically not on the same site as my tutor and therefore without direct supervision. To maintain a trusting relationship with my tutor, I had to show him my seriousness and my ability to work independently.

1.2 Stakes

This internship had different stakes, both for me and for the university and the research centre. For me this internship presented three main stakes:

1. A cultural and human stake: The opportunity to live in a foreign country for such a long time was a first time for me. This experience allowed me to discover and live the German culture but also to face the language barrier. Even though Germany is close to France, I could notice differences between my daily life in France and my daily life in Germany. To better integrate and adapt, I decided to learn German. This internship allowed me to enrich my cultural horizon and to understand the importance of an open mind for an engineer.

2. A professional stake: Knowing how to work with a multicultural and international team is, in my opinion, a necessity for an engineer. This thought led me to choose the English course on integration and team management in an international context in English during semester 4. By applying the acquired notions, I succeeded in integrating and working efficiently with this new team.

3. A technical stake: During this internship, I was able to familiarise myself with the concepts of multi-agent environments and discover the technical constraints induced by this type of system.

This project also presented stakes for the university, of which I have identified three in particular:

1. A pedagogical stake: A university can differentiate itself from another by its courses of study and its level, but above all by its pedagogy and its tools. This project has the potential to provide the *University of Oldenburg* with a new versatile and relevant pedagogical tool that will enable it to stand out from other universities. If the project offers many advantages, it could even be implemented in other schools, which would contribute to the university's influence in the country.

2. An economic stake: A major cost in this type of project is staff, even with students. Because of my military status, the university did not need to pay me financially, which meant that any work produced was produced at low cost.

3. A political stake: This project also allowed the university to evaluate the level of robotics students at *ENSTA Bretagne*. If my performance was satisfactory, the university can, in the short term, offer more internships to students and, in the long term, organise joint challenges in robotics and propose substitutions between our two schools.

2. **Description of the activities**

2.1 Organization

This internship is my second experience in development and project management. Before joining *ENSTA Bretagne*, I had the opportunity to develop an application during my military year. During this project, I underestimated the importance of a good organisation, and this impacted my performance at the end of the development and delayed the deployment of the application. In order not to repeat such a situation, I took the time to think about how to organise my work.

This reflection was even more necessary given the context of my internship. Indeed, as previously mentioned, my tutor was working in a building geographically distant from mine. To ensure supervision of the placement, **Prof. RAUH** wanted to organise two meetings a week to monitor my progress. I was afraid that this configuration would be too time-consuming for both him and me. I therefore decided to show him that I was organised, rigorous, autonomous and that I would inform him weekly of the progress of the project to develop a relationship of trust.

To enable him to follow my work, I was sending him a detailed weekly report of my activities. These reports also contained schedules (see **Appendix 2**) and diagrams explaining the concept developed during the week (see **Appendix 3**). These diagrams are, in my opinion, very useful as they can be used by the teachers to present the *Turtlebot3 Burger* to the students and more generally to present the project. By sending weekly reports and talking with my tutor during lunchtime, we quickly developed a relationship of trust, and the two weekly meetings were replaced by a monthly meeting.

To facilitate further development and information sharing, I created a dedicated *GitLab* project for my work. There I was able to save all the information and write documentation as a report for the university. During the development, I read and asked questions on various forums. So, I also created a dedicated profile on each forum for this project with the nickname: "learn2learn". The next developers will know that all the questions asked by this profile are about this project

The implementation of this organisation allowed me to maintain a fast and efficient work rhythm, while maintaining a relationship of trust with my tutor. A more detailed analysis of the results of my organisation will be made in a later section.

2.2 Technical Foundations

2.2.1 Turtlebot3 Burger

According to the *main documentation* of the *Turtlebot3 Burger:*

“TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping. The goal of TurtleBot3 is to dramatically reduce the size of the platform and lower the price without having to sacrifice its functionality and quality, while at the same time offering expandability. The TurtleBot3 can be customized into various ways depending on how you reconstruct the mechanical parts and use optional parts such as the computer and sensor.”

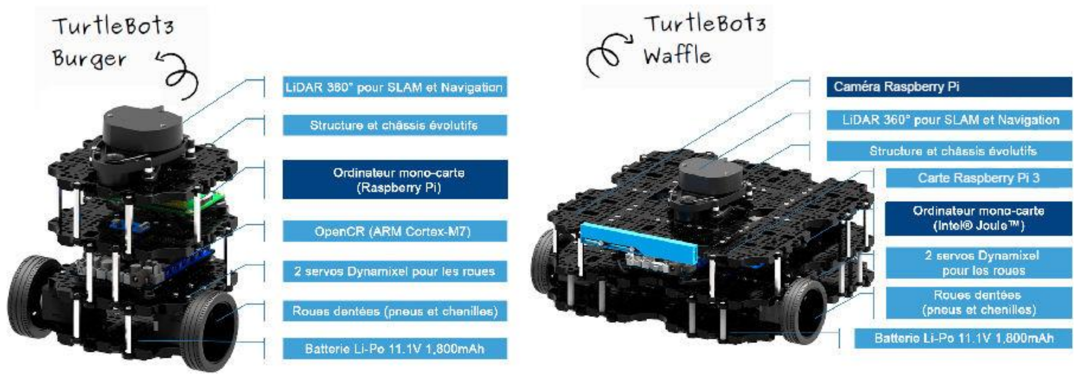

TurtleBot3 Burger
LiDAR 360° pour SLAM et Navigation
Structure et châssis évolutifs
Ordinateur mono-carte (Raspberry Pi)
OpenCR (ARM Cortex-M7)
2 servos Dynamixel pour les roues
Roues dentées (pneus et chenilles)
Batterie Li-Po 11.1V 1,800mAh
TurtleBot3 Waffle
Caméra Raspberry Pi
LiDAR 360° pour SLAM et Navigation
Structure et châssis évolutifs
Carte Raspberry Pi 3
Ordinateur mono-carte (Intel® Joule™)
2 servos Dynamixel pour les roues
Roues dentées (pneus et chenilles)
Batterie Li-Po 11.1V 1,800mAh

FIGURE 3 – Turtlebot3 Burger and Turtlebot3 Waffle features

As shown in the Turtlebot3 Burger datasheet in **Appendix 4**, the robot is initially equipped with various sensors and actuators. The available sensors are a LIDAR, two odometers, an IMU and a compass. The robot is powered by two speeds controlled *DYNAMIXEL* servomotors. The embedded intelligence includes a *Raspberry Pi 4B* and an *OpenCR 1.0 board*. The strong point of this robot is its modularity, and it is an aspect that I will use in this project

2.2.2 Robot Operating System (R.O.S.)

According to the *R.O.S. official web site*:

"The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source."

R.O.S. is therefore a key tool for roboticists and benefits from a large and active community. This software offers an autonomous and efficient management of execution threads and communication threads between the different instances. To do so, R.O.S. uses the concept of Node and Topic, a node is a program or an executable and a topic is a discussion channel on which nodes can publish or read information. A basic application in robotics is to create one node per sensor and publish sensors information in dedicated topics. The main program (which makes the decisions) then only must read the topics to estimate the state of the robot and thus determine the new action.

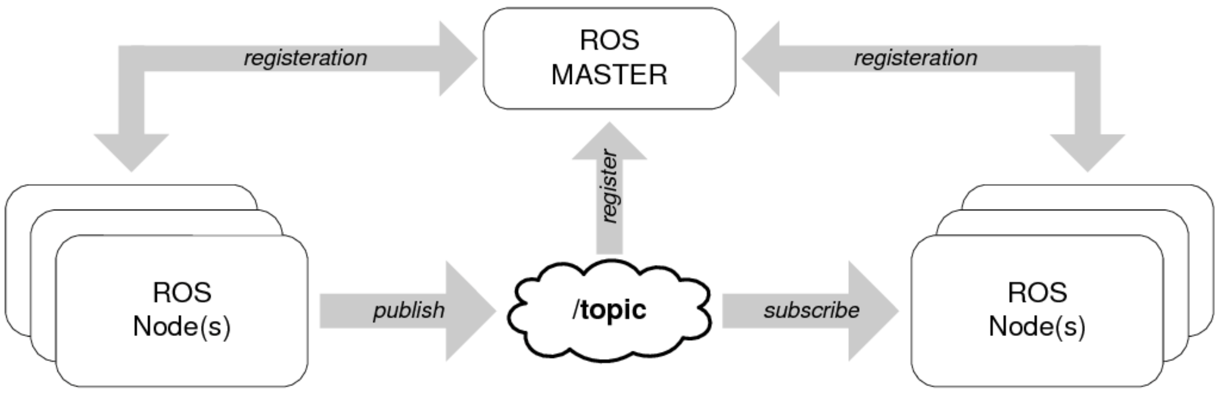


FIGURE 4 – Concept of Node and Topic in R.O.S.

2.2.3 MATLAB Simulink

MATLAB Simulink is, according to the dedicated *MathWorks web site*, *"a block diagram environment used to design systems with multidomain models, simulate before moving to hardware, and deploy without writing code"*. Simulink is a part of the MATLAB environment which is, according to the dedicated *MathWorks web site*, *"a programming and numeric computing platform used by millions of engineers and scientists to analyse data, develop algorithms, and create models"*.

We decided to use MATLAB Simulink for this project because, by using blocks, any user can create a control algorithm. Indeed, MATLAB Simulink integrates different toolboxes and libraries that can, for example, handle communications with R.O.S. topics. The user can then read the information from a sensor without worrying about the means to send the information to MATLAB Simulink.

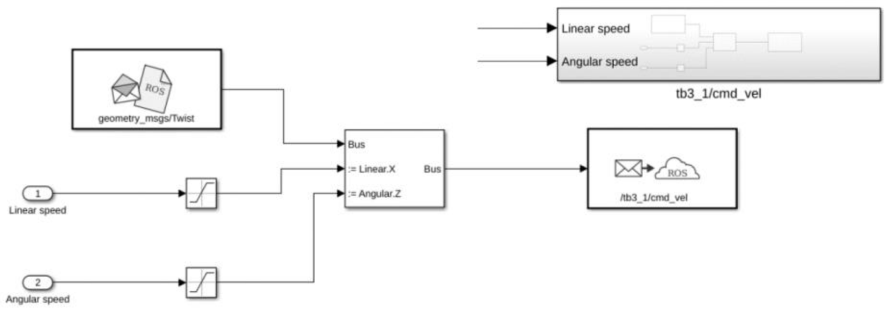


FIGURE 5 - Example of application for the R.O.S. toolbox: publish speed commands on a topic

2.2.4 Eclipse 4diac

Eclipse 4diac is, according to the *Eclipse web site*, *"an open-source infrastructure for distributed industrial process measurement and control systems based on the IEC 61499 standard. [...] IEC 61499 defines a domain specific modelling language for developing distributed industrial control solutions"*.

Like MATLAB Simulink, Eclipse 4diac is a graphical programming environment that uses block sequences to create programs. Compared to MATLAB Simulink, Eclipse 4diac is dedicated to industrial applications and its blocks are therefore adapted to this need. The blocks are usually simple functions such as loops, inverters, or comparators. They use two types of signals to communicate: event signals and data signals. An event signal allows one block to trigger another, and a data signal allows information to be shared between blocks. It is also possible to create your own blocks, which increases the capabilities of this software.

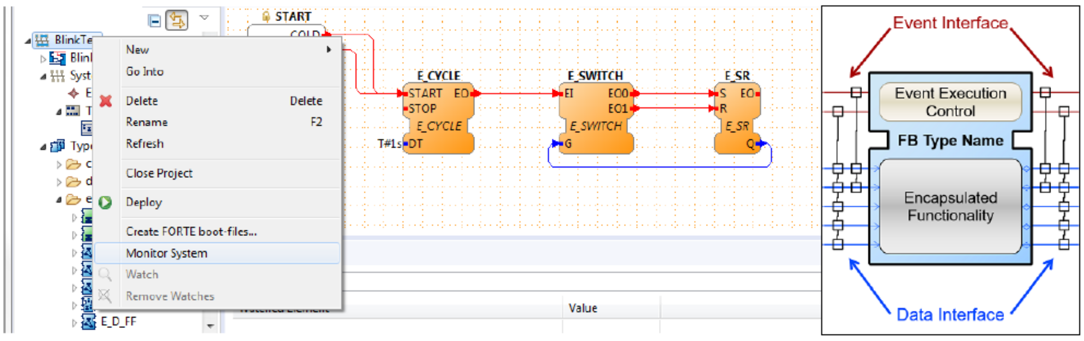


FIGURE 6 – Illustration of the IEC 61499 standard

2.3 Deploying a single-agent configuration

2.3.1 Building the robots

The first step of the project was to deploy the two *Turtlebot3 Burger* that I had at my disposal. Indeed, the robots are delivered in kit and therefore need to be built, configured, and tested. To do this I relied on the official documentation provided by *ROBOTIS* [1]. The procedure

to follow is not complex but quite long. It took me about two and a half days to be able to fully deploy a single robot.

The complete deployment takes place in 5 steps: building the robot, setting up my computer, commissioning the *Raspberry Pi*, commissioning the *OpenCR* board and testing the built-in functions such as SLAM or the Autonomous Navigation program. I will not detail the commands used in the deployment, but they are available in the documentation [1].

The construction of the robot is quite simple, just follow the instructions in the assembly manual. Nevertheless, you must be careful on some steps. You must be careful when installing the servo motors because it is easy to invert the two motors. The error will only be noticed during the test phase, when the robot will do the opposite of what it is asked to do. As the motors are on the first layer, the whole robot will have to be dismantled to correct the error. It is also necessary to take care to leave the different connectors of the boards (*Raspberry Pi* and *OpenCR*) accessible to connect a keyboard and a screen during the commissioning.

Setting up the computer mainly consists of installing ROS1 and the *Turtlebot3 Burger* ROS packages, so I won't go into detail about this part. My hardware configuration was a Ubuntu 20.04 LTS computer with ROS1 Noetic. I also had to modify my bashrc file to allow my ROS master to communicate over the network.

To get the *Raspberry Pi 4B* up and running, I had to burn the OS image provided by *ROBOTIS* to the SD card. The image provided was an Ubuntu 20.04 Server with a compiled version of ROS1 Noetic. I then did the network setup to allow the robot to connect to the *OFFIS* WIFI network (while waiting to deploy a dedicated network).

Setting up the *OpenCR* card was quite simple as I just had to connect it to the *Raspberry Pi* and install the dedicated firmware prepared by *ROBOTIS*.

The final step was to test the robot by running built-in functions. Indeed, among the packages of the *Turtlebot3 Burger* are three interesting packages: teleoperate, SLAM and Autonomous Navigation. These packages use all the elements on the robot and thus allow us to check the integrity of the robot. To perform the test, I first teleoperated the robot to take it into an empty *OFFIS* room, then I used the SLAM program to map the room and used the Autonomous Navigation program to allow the robot to navigate in the room while avoiding obstacles. The test was successful, and I concluded that both *Turtlebot3 Burger* were now ready to be used.

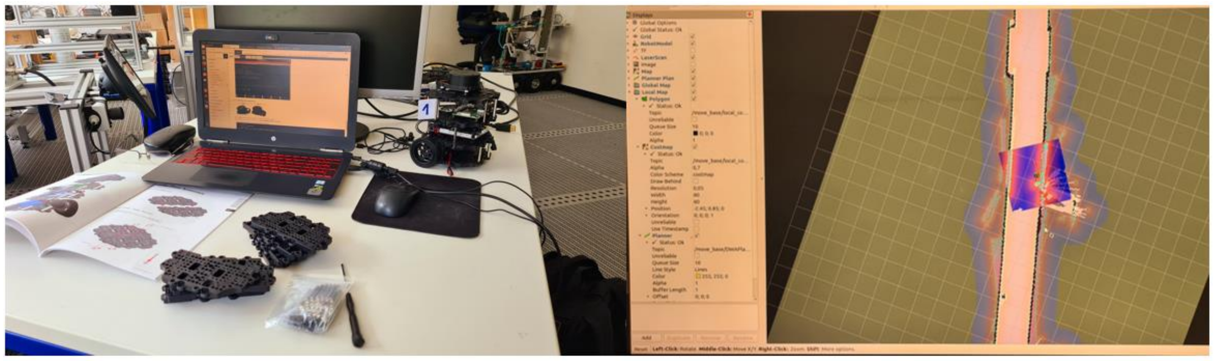


FIGURE 7 - Building the robot (left) and using the autonomous navigation program (right)

2.3.2 Deploying the network in a basic configuration

The *Turtlebot3 Burger* will be used in various contexts (courses, tutorials, projects, demonstrations, etc.) and in various locations (universities, *OFFIS*, conference room, outdoors, etc.). Access to a secure, stable WIFI network with an Internet connection will therefore not always be certain. Even if we assume that this is possible, it will mean changing the network configuration of each *Turtlebot3 Burger* each time the WIFI network is changed. This configuration is not complex, it is enough to modify the file *"50-cloud-init.yaml"* by adding the SSID and the password of the new network before restarting the Raspberry Pi, but it is especially time consuming because it is necessary to repeat the operation for each agent. Moreover, a connection to a new network can lead to various problems (frequency problems, various incompatibilities, poorly configured network firewall, etc.). This led us to deploy a dedicated WIFI network using a router. A router is easily transportable, secure and retains its configuration over time, so no incompatibility problems could appear.

For this project, I proposed to use the *NETGEAR AC1200* router (see data sheet in **Appendix 5**). This router is compact, offers a good throughput and allows to deploy a 2.4Ghz network and a 5Ghz network thanks to its two antennas. I've already had the opportunity to work on *NETGEAR* routers and I am therefore used to their firmware. For this project, I will only use the 2.4Ghz network for a question of range. The 5Ghz network can be used for another project or used for the *Turtlebot3 Burger* project if a higher throughput than the 2.4Ghz network one is required. The need for high throughput was the main parameter for the design. Indeed, we are going to use ROS1 and integrate many sensors on each robot, such as cameras that require a high throughput to broadcast the images acquired in real time.

Deploying the router was straightforward, all I had to do was change the SSID to *"TURTLEBOT"* and change the base password by accessing the router configuration page (http://192.168.1.1). However, problems occurred when I tried to connect the *TurtleBot3 Burger* to the new network. The robots refused to connect to the router, but all my other devices could connect to it. By doing some research [2], I realised that the image provided by *ROBOTIS* configures the *Raspberry Pi* to American standards, especially the network standards. As the American WIFI channels are different from the European WIFI channels used by the router, it was impossible for the *Raspberry Pi* to connect to the router. I then simply modified the network parameters of the boards by changing the *REGDOMAIN* variable to "GE" (Germany) in *"/etc/default/crda"*. Following this modification, the robots could now connect to the router without any problem.

However, it should be kept in mind that the router is not connected to the Internet. It is therefore sufficient to operate the *Turtlebot3 Burger* but not to update it or to do any development. The loss of Internet access did, however, lead to a problem: clock synchronisation via NTP was now impossible. Indeed, the *Turtlebots3 Burger* are not equipped with RTC. To set their clock, they need to synchronise with an NTP server at each start-up. When they lost access to the Internet, they no longer had a server to refer to, so I had to deploy my own NTP server.

2.3.3 Setting up a NTP server

When using ROS1 in a multi-agent environment, it is essential that each agent is in the same time frame, especially with a remote and centralized ROS master. Indeed, most of the messages that will pass through the network will contain timestamps to timestamp the information. In the event of a time synchronisation problem, messages may be read at incorrect dates or even never be read if the message is dated in the future. In such a context, regulations or collaborations are therefore impossible. To allow the *Turtlebot3 Burger* to place themselves in the same time frame, they need a common time reference, i.e., an NTP server.

To deploy an NTP server, you must first define which system will host the server. There are several options: deploying the NTP server on one of the *Turtlebot3 Burger*, deploying the NTP server on the Master PC or deploying a physical NTP server. The first solution is not possible because the robots' time reference is unreliable and random. Moreover, in case of a breakdown or maintenance of the robot, the server will be inoperative. The third solution would be the most optimal, but it would be very expensive and would create extra bulk. Deploying the NTP server on the Master PC is therefore the most suitable solution for this project. This is especially true since it is the PC Master, more precisely the Master ROS, that will centralise the data of all the *Turtlebot3 Burger*. Synchronising to its clock is therefore the most practical and efficient choice to avoid any problems.

The deployment of the NTP server was done in two phases: the configuration of the server on the Master PC and the setup of the clients on the *Turtlebot3 Burger*. To create and configure a local NTP server, I used one of the many tutorials available online [3] and a technical report produced by students at the University of Toledo, Toledo, USA [4]. I then verified the correct operation of my server by connecting to it with an NTP client that I coded in Python. The client was able to communicate perfectly with the server with a latency of about 1ms, which was more than acceptable. Indeed, my goal was to obtain a time offset between my agents lower than 500ms. This value corresponds to the maximum offset tolerated by the *Turtlebot3 Burger* ROS packages. My NTP server was therefore operational.

The most complex step was to set up the NTP client used by the *Turtlebot3 Burger*. Indeed, I never managed to make it communicate with my NTP server despite many attempts and the opening of a thread on the *Ubuntu* forum. To overcome this problem, I decided to use the NTP client I had previously coded in *Python*, optimize it and run it in the background using *Ubuntu CRON*. With this method, I managed to get an offset of about 50ms between the different agents. However, this should only be a temporary solution, as it is sometimes necessary to manually run the program to force the synchronisation. The final solution will be to make the real NTP client of the *Turtebot3 Burger* communicate with the server and thus achieve an offset of about 1ms.

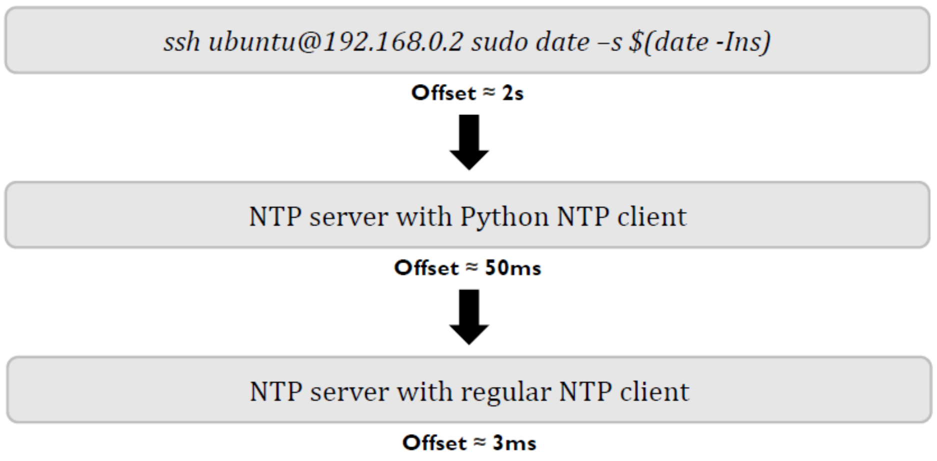

ssh ubuntu@192.168.0.2 sudo date –s $(date -Ins)
Offset ≈ 2s
NTP server with Python NTP client
Offset ≈ 50ms
NTP server with regular NTP client
Offset ≈ 3ms

FIGURE 8 – The different steps of my work to deploy the NTP server

2.4 Upgrading the hardware structure

2.4.1 Adding a Raspberry Pi camera

The second step of the project was to improve the hardware structure of the robots, or at least to make it more modular. Indeed, the *Turtlebot3 Burger* is only a mobile base that must be modified to fit our goals. In the case of the university, the aim was to obtain a versatile and modular hardware architecture so that professors and students could adapt it to their project needs. However, before working on how to achieve such a hardware architecture, I first had to install sensors to demonstrate the project.

Indeed, during this period, *OFFIS* organised a major event called *"Long Night of Digitization"* in partnership with *BTC Business Technology Consulting AG*, an IT consulting company based in Oldenburg, Germany. The aim of the event was to reconnect the Public and the Industry with the Research after the COVID19 crisis by demonstrating the importance of digital technologies in our lives and in the future. On this occasion, all the departments had a stand to explain their work and I was no exception. My objective was to capture the attention of young students and of the public with the *Turtlebot3 Burger*, allowing them to control the robots with a game controller and camera feedback. I had therefore to add a camera on the robot and create a program to teleoperate the robots with a game controller. I will concentrate here on the addition of the camera, but I will detail the results of this event in a dedicated part (see 3.3 Long Night of Digitization).

My tutor, **Prof. RAUH**, had already planned to participate in this event and had therefore previously purchased *Raspberry Pi cameras*. This camera offers several advantages: it is compact, cheap, and easily integrable on a *Raspberry Pi*. However, due to an error at the time of the purchase, we received *Raspberry Pi NoIR* cameras (see data sheet in **Appendix 6**). These cameras are like the normal cameras except that they do not have an infrared filter (NoIR). This difference causes a colour distortion in the camera images, as if a pink filter was applied (see Figure 9), but it allows night vision to be performed with an infrared lamp. The problem is that the LIDAR also uses infrared light to estimate the distance around the robot, so incorporating an infrared lamp would interfere with the LIDAR and even render it inoperative. However, we decided to continue with these cameras because even if the colour is altered, it is still possible to distinguish the shapes and therefore to teleoperate the robot by camera feedback.

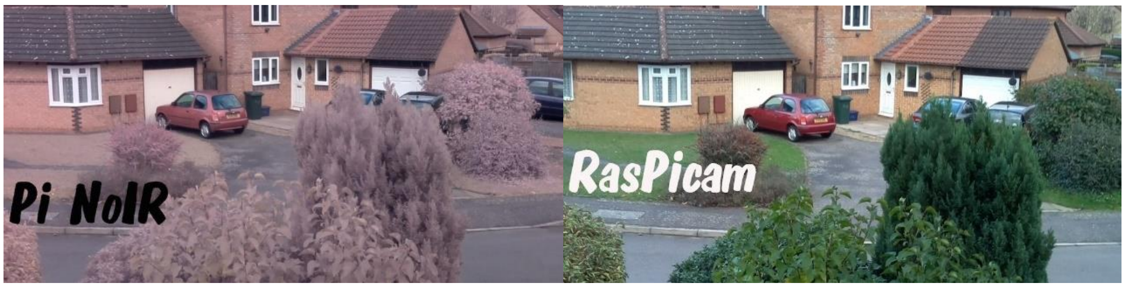


FIGURE 9 – Difference between Raspberry Pi NoIR camera (left) and regular Raspberry Pi camera (right)

Getting the camera up and running was a straightforward step, firstly I just plugged the camera into the dedicated connector on the *Raspberry Pi*. The next step was to find the necessary drivers to retrieve the video stream and publish it in a ROS topic. The advantage of using a *Raspberry Pi camera* is that it is a very common model and therefore finding working drivers and programs is easy. Searching in the main ROS package of the *Turtlebot3 Burger* called *"turtlebot3_bringup"*, I found the launch file *"turtlebot3_rpicamera.launch"* which allows to get the video stream and publish it in a ROS topic. While reading the documentation I noticed that the image was published in uncompressed format which could consume a lot of bandwidth, especially with 4 *Turtlebot3 Burger*. To avoid this, I installed the ROS package called *"compressed_image_transport"* which allows the *turtlebot3_rpicamera.launch* file to publish the image in a compressed format, thus saving bandwidth. I then simply modified the main launch file of the Turtlebot3 burger by adding code to launch the *turtlebot3_rpicamera.launch* file and publish the compressed video stream of the camera in a dedicated topic.

The last step was to integrate the camera on the robot. To do this, I simply modelled a support using *Autodesk Inventor*, a CAD software. I then printed it using the 3D printers available at *OFFIS* and assembled it on the robot. I designed the support in such a way that it is easy to change its orientation to make it as versatile as possible. Indeed, future users will be able to choose: to orientate the support downwards to do line tracking, to orientate the support vertically to do remote controlling and obstacle detection or to orientate it upwards to do face detection and person tracking. After these manipulations, the *Raspberry Pi cameras* were fully operational and integrated on the robots.

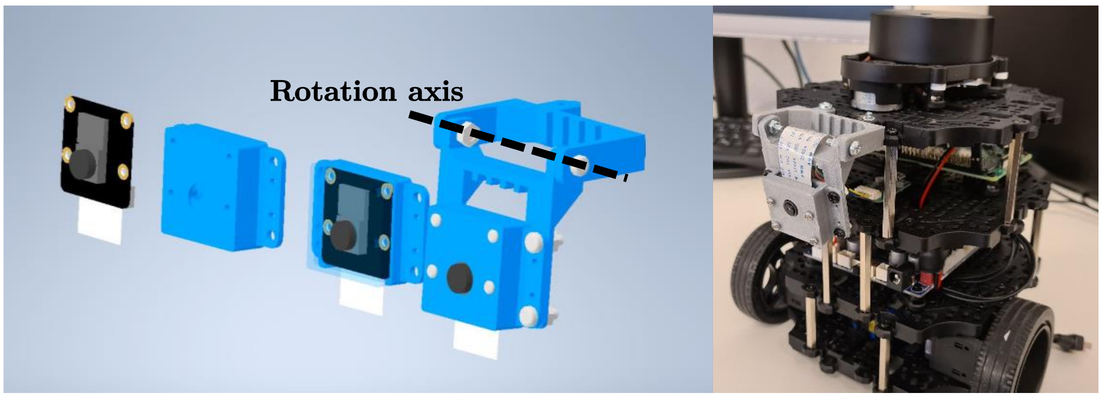


FIGURE 10 – 3D model of the Raspberry Pi camera support (left) and one integrated on the robot (right)

2.4.2 Designing a modular hardware structure

Let's go back to the initial objective which was to modify the hardware structure of the *Turtlebot3 Burger* to make it more modular. The notion of modularity I am talking about corresponds to the ease with which it will be possible to add, modify or remove a sensor on the robot. So, we should not only limit ourselves to physical modularity but also to software modularity, because if connecting a new sensor is simple but configuring it is complex then the solution is not modular. So ideally, we need to find a versatile and ergonomic physical solution that integrates drivers that are easy to use and adapt. It is this motivation that led **Prof. RAUH** to turn to the *HAT Brick* from *Tinkerforge* (see datasheet in **Appendix 7**).

Raspberry Pi HATs are add-on boards that can be plugged into a *Raspberry Pi* to extend its capabilities. The *HAT Brick* from *Tinkerforge*, for example, offers a stabilised power supply, an RTC, a sleep mode and, above all, 8 standardised connectors allowing a wide range of sensors to be connected. With this solution, it is therefore possible to buy different sensors that all use the same connector and that can be connected on the *Raspberry Pi* by simply plugging them in. Like many *Raspberry Pi HATs*, the *HAT brick* offers a set of drivers to read sensor data in different programming languages such as Python or C++. All these reasons made us believe that the *HAT brick* was the best solution for this project

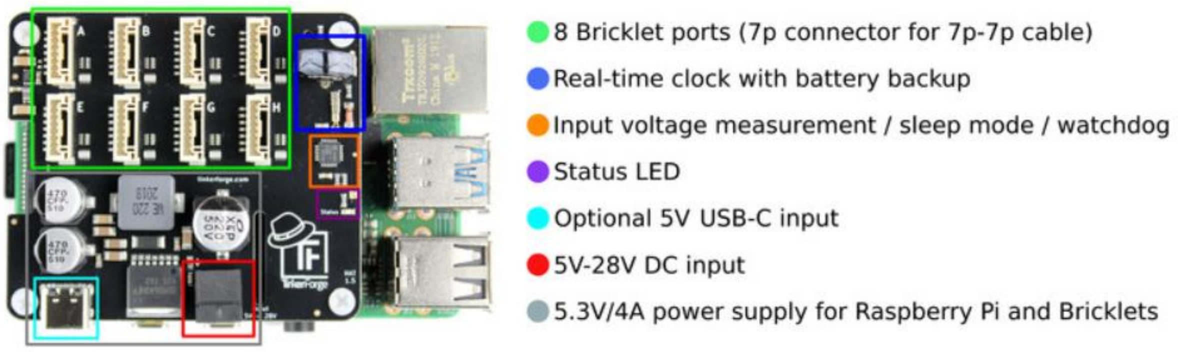
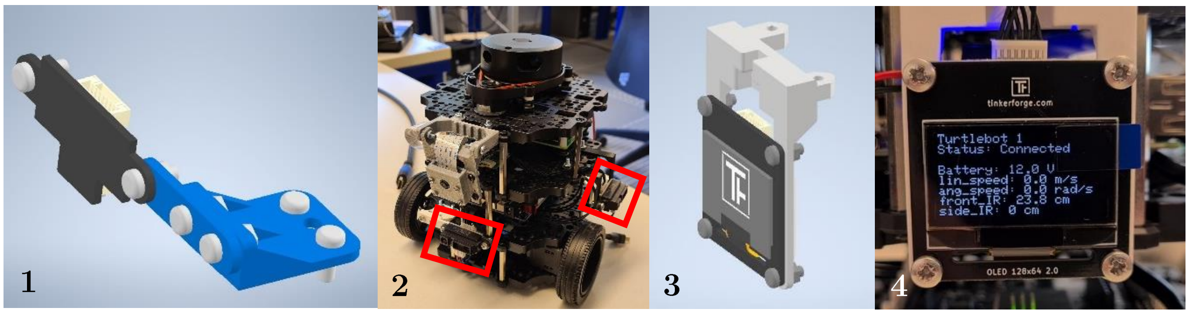

8 Bricklet ports (7p connector for 7p-7p cable)
Real-time clock with battery backup
Input voltage measurement / sleep mode / watchdog
Status LED
Optional 5V USB-C input
5V-28V DC input
5.3V/4A power supply for Raspberry Pi and Bricklets

FIGURE 11 – Tinkerforge HAT brick

To obtain the new hardware structure of the robot, it is therefore sufficient to add a *HAT Brick* on the *Raspberry Pi 4B* as shown in **Appendix 8**. However, this new configuration required a modification of the robot's electrical structure. Indeed, the *Raspberry Pi 4B* was supplied with 5V stabilised on pins 3 and 4 by the *OpenCR* (see **Appendix 3**) but by adding the *HAT Brick*, pins 3 and 4 were no longer available. It is now necessary to supply the *HAT Brick* directly, which will then manage the power supply of the sensors and of the *Raspberry Pi* in stabilised 5V.

According to the documentation, the best way to power this device is to use the 5V-28V supply described in Figure 11. As the battery is connected to the *OpenCR* for powering the motors, I had to find a way to power the *HAT Brick* with the battery voltage from the *OpenCR*. Reading the documentation of the board (see **Appendix 9**), I noticed that 3 power outputs are available: one in 3.3V, one in 5V stabilized (used previously to power the *Raspberry Pi*) and one in 12V 4A. I chose to use the 12V 4A output to power the *HAT Brick* because it offers a similar voltage to the battery. For that I tried to find the corresponding power cable, but without success. So, I started with a raw wire and ordered the necessary connectors to make my own power cable. Once this step was done, the new hardware structure was functional, and I just had to install the *HAT Brick* drivers and daemons on the *Raspberry Pi* to make it operational.

During this project I had access to 3 sensors: the *10-80cm IR distance Bricklet 2.0*, the *4-30cm IR distance Bricklet 2.0* and the *OLED 128x64 Bricklet 2.0*. In short, two infrared distance sensors with different ranges and a touch screen. To be able to integrate them on the robot and demonstrate the modularity of the structure I had to create two dedicated supports as shown in Figure 12.



```
Turtlebot 1
Status: Connected
Battery: 12.0 V
lin_speed: 0.0 m/s
ang_speed: 0.0 rad/s
front_IR: 23.8 cm
side_IR: 0 cm
```

FIGURE 12 – 1. IR distance sensor support 2. Turtlebot3 Burger with IR distance sensors 3. Touch screen support 4. Screen integrated on the robot

2.4.3 Creating a dedicated HAT Brick R.O.S. package

Before looking for the most modulable software architecture for this project, we need to understand how to acquire data from the sensors connected to the *HAT Brick*. The main element of the acquisition system is the *Brick Daemon*, a daemon that acts as a bridge between the *HAT Brick* and the drivers of the different programming languages. The drivers allow communication with the daemon and the access to sensors information in two different ways: by specifying the connector to be monitored (designated by a letter between A and H, see Figure 11) or by specifying the identifier of the sensor to be monitored (called *UID* in the drivers). It is therefore sufficient to install the drivers corresponding to our programming environment and to choose a method for reading the information coming from the *HAT Brick*. For modularity reasons, we prefer to read the information by specifying the *UID* of the sensor and not by specifying the connector. This method avoids many problems such as data type management and connection errors. Now that we know how to read the information from a sensor, we need to think about the software structure to adopt to easily add, modify or remove a sensor from the robot.

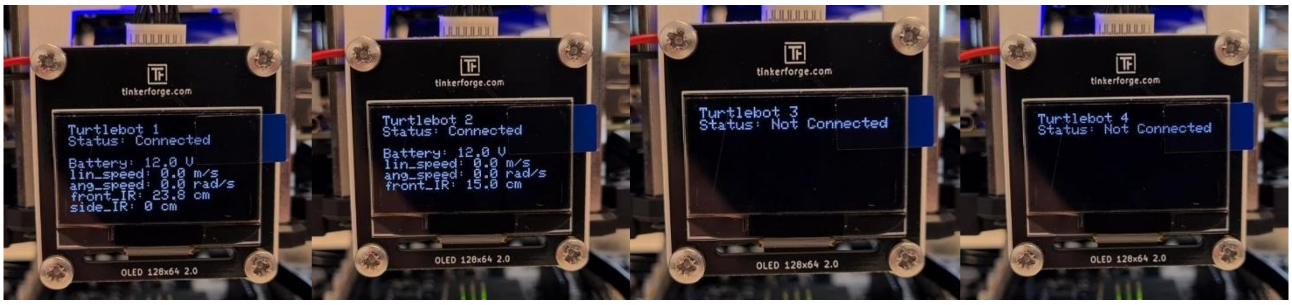
As a reminder, the *Turtlebot3 Burger* are powered by ROS1 and launched by a main launch file called *"turtlebot3_robot.launch"* in the ROS package called *"turtlebot3_bringup"*. The aim would be to apply a similar method to the one used to read and publish information from a *Raspberry Pi camera*. Indeed, using the *"turtlebot3_rpicamera.launch"* file, it is possible to read the video stream of a camera and to publish it in a topic specified by the user. To integrate a new camera on the robot, it is thus enough to modify the file *"turtlebot3_robot.launch"* to call the launch file *"turtlebot3_rpicamera.launch"* and to specify the id of the monitored camera and the topic in which to publish the information. That's why I decided to create a dedicated ROS package that would contain a python program for each type of sensor. This package is called *"hat_brick_pkg"*. To add a sensor, it would be enough to modify the file *"turtlebot3_robot.launch"* to call the program corresponding to the type of sensor and to specify the UID of the sensor and the topic in which to publish the information

To have a better understanding, let's take a simple example: I want to add an IR distance sensor to the front of the robot and publish the data in the topic named *"front_distance_ir"*. In this example, the UID of the sensor will be *"TEw"*. You just have to connect the sensor and integrate it on the robot using the support mentioned in the previous section. Then you have to modify the file *"turtlebot3_robot.launch"* by creating a new node which calls the program named *"distance.py"*, program which corresponds to the distance sensors. It only remains to specify in parameter the UID and the name of the topic to have a functional sensor. This example can be summarized by Figure 13, which corresponds to the lines added in the main launch file.

```
<node pkg="hat_brick_pkg" type="distance.py" name="front_distance_pub" output="screen">
   <param name="topic_name" value="front_distance_ir"/>
   <param name="UID" value="TEw"/>
</node>
```

FIGURE 13 – Modifying the launch file to add a new distance sensor

I also created a program for the screen called *"screen.py"*. The program takes only one parameter which is the UID of the screen. The user can choose to display an image, by publishing the corresponding binary matrix in the topic *"display/image"*, or to publish text on the screen lines, by publishing the strings in the topics *"display/lineX"*, where X is the number of the line. In the case that the user does not display any information, the screen will show the status of the different *Turtlebot3 Burger*. If a robot is not connected, the screen will display *"Not Connected"* and if the robot is connected, the screen will display *"Connected"* as well as various information from the robot such as battery voltage and sensor measurements. In the case of Figure 14, we can see for example that the *Turtlebot3 Burger* 1 and 2 are connected and therefore we can read their status but *Turtlebot3 Burger* 3 and 4 are disconnected and therefore the screen shows *"Not Connected"*.



```
Turtlebot 1
Status: Connected

Battery: 12.0 V
lin_speed: 0.0 m/s
ang_speed: 0.0 rad/s
front_IR: 23.8 cm
side_IR: 0 cm
```

```
Turtlebot 2
Status: Connected

Battery: 12.0 V
lin_speed: 0.0 m/s
ang_speed: 0.0 rad/s
front_IR: 15.0 cm
```

```
Turtlebot 3
Status: Not Connected
```

```
Turtlebot 4
Status: Not Connected
```

FIGURE 14 – OLED display showing the state of the different agents of the fleet

2.5 Evolving to a multi-agent configuration

2.5.1 Choosing a multi-agent hardware configuration

After having deployed the *Turtlebot3 Burger* in a single agent configuration and after having improved the hardware and software structure of the robots to make them more scalable, it was time to evolve this previous work towards a multi-agent configuration. Evolving towards such a configuration implies many changes, both from a network and a software point of view. Before starting this work, **Prof. RAUH** and I thought about the hardware structure that we would give to each *Turtlebot3 Burger*. Indeed, the objective is also that the robots can collaborate to carry out a mission, which implies that they are complementary and not redundant.

In its nominal configuration, the group will consist of 4 *Turtlebot3 Burger*. Each robot has 1 LIDAR, 2 odometers, 1 IMU and 1 *Raspberry Pi NoIR camera* but the question is which sensors to add on each robot to make them complementary. We have divided the robots into 3 classes: supervisor, operator, and support agents. The supervisor will be the leader of the group and should be able to perform all the missions, understand its environment and eventually interact with the user through a screen. The operator is a more basic robot that will be specialised in a specific task and will receive its instructions from the supervisor. The support agent is a robot that will behave most of the time like an operator but will be able to support or replace temporarily the supervisor if required. In the case of the project, the*Turtlebot3 Burger* number 1 will be the supervisor, the*Turtlebot3 Burger* number 2 and 3 will be operators and the*Turtlebot3 Burger* number 4 will act as a support agent.

To carry out its missions, robot number 1 will be equipped with two distance IR sensors. The sensor with the longer range will be placed on one lateral side of the robot for wall tracking and the sensor with the shorter range will be placed in front of the robot for frontal obstacle detection. It will also be equipped with a screen to be able to interact with a user or simply display information about the robots and the status of the mission. Robot number 2 will be equipped with a short-range distance IR sensor that will be placed in front of the robot. Robot number 3, to be complementary to robot number 2, will be equipped with a long-range distance IR sensor that will be placed in front of the robot. Robot number 4 will not be equipped with distance IR sensors but with a webcam. Indeed, as mentioned in the previous sections, due to an error, the camera on each robot does not have an infrared filter, which gives the impression of having a pink filter on the images. It is still possible to distinguish shapes but not colours. For this reason, we have equipped robot number 4 with a high-resolution webcam that will allow it to recognise colours in the short term and that can be used for stereovision in the long term. Robot number 4 is also equipped with a screen to allow it to interact with users if robot 1 is not available. The **Appendix 10** is an illustration of this material structure.

2.5.2 Upgrading the network configuration

In the case of a multi-agent configuration, the management and the allocation of IP addresses are very important steps. Indeed, in a nominal operation, the network will host several *Turtlebot3 Burger*, the computer hosting the ROS master (called master computer or just Master) and the computers of the different students. To operate the robots will need to know the IP address of the master computer and to launch the robots, the teacher will need to know location of each robot. The students will also need to know the IP address of the master to retrieve information from the robots. Therefore, if no network structure is defined, the allocation of IP addresses will be random, and this will prevent a simple and intuitive use of the project. Therefore, I had to think and propose a network structure adapted to this project, which would allow to easily associate an IP address to a type of agent and thus to propose a simple and intuitive use.

First, I identified the needs and functioning of each type of system. The *Turtlebot3 Burger* and the Master must be easily located on the network. Therefore, it is necessary to assign them a fixed IP address which will be reserved for them at each network deployment. On the contrary, students and other users do not need to be easily locatable on the network. They only need to be able to locate the Master to send requests to it, not the contrary. Therefore, assigning a range of IP addresses to which users can connect is sufficient. This configuration leads us to Figure 15, which is a representation of the desired network organisation for this project.

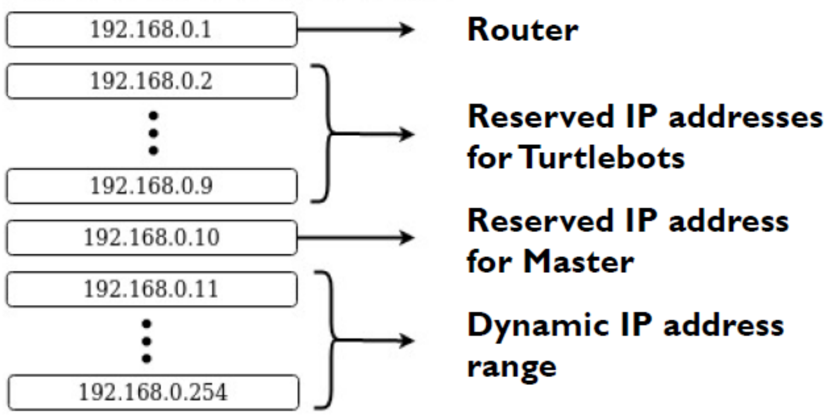


FIGURE 15 – Structure of the network

In this configuration, 192.168.0.1 is the default address to access the router. I then reserved 8 IP addresses (192.168.0.2 to 192.168.0.9) for the *Turtlebot3 Burger*. When I was writing this report, the university only had 4 *Turtlebot3 Burger*. The 4 unused IP addresses will allow the university to add new robots if the project shows interesting results. To assign and reserve a unique IP address for each robot, I decided to apply an "n+1" relationship between the robot number and its IP address. For example, *Turtlebot3 Burger* number 1 will receive the IP address 192.168.0.2 and conversely, the robot connected to the IP address 192.168.0.5 will be the *Turtlebot3 Burger* number 4. By applying this rule, it is therefore very easy to locate a particular robot on the network. The address 192.168.0.10 will host the Master. The remaining IP addresses will be the IP addresses available for the users. To apply this configuration, I just had to modify the router parameters, notably by modifying the dynamic range of IP addresses. An illustration of this network structure (excluding users) can be found in **Appendix 11**.

2.5.3 Running multiple robots with R.O.S.

To deploy a multi-agent configuration, it is necessary to know how to make several identical robots run in parallel under ROS without creating conflicts. Indeed, in its initial configuration, the *Turtlebot3 Burger* does not allow several robots to run at the same time without creating conflicts. The *Turtlebot3 Burger* are in fact instances of the same robot, so they have the same software structure. In the case of ROS, this means that the bots will try to publish their information in the same topics, thus creating conflicts because when reading a piece of information at a given time, it will be impossible to identify which *Turtlebot3 Burger* published the information. This principle is illustrated in Figure 16, using the example of the topic */scan* in which the LIDAR data is published.

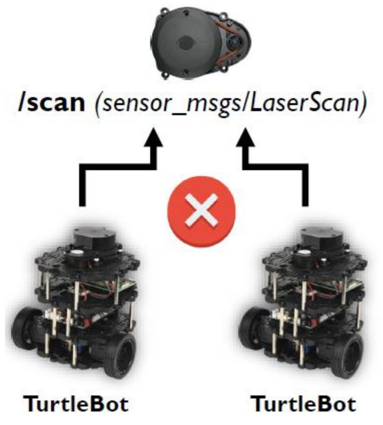


FIGURE 16 – 2 Turtlebot3 Burger trying to publish data on the same topic

The most obvious and simple solution to avoid this problem is to use the concept of namespace in ROS. A namespace allows you to differentiate one instance of an object from another by giving it a name that will be used as a prefix in front of its topics. In the case of this project, each *Turtlebot3 Burger* will receive a name depending on its number in the form *"tb3_X"* with X the number of the robot. The different robots will add their prefixes to the name of their topic and allow to distinguish, for example, the topic */scan* of *Turtlebot3 Burger* number 1 from the one of the *Turtlebot3 Burger* number 2 and thus avoid conflicts. In the case of the */scan* topic, the new configuration is illustrated in Figure 17.

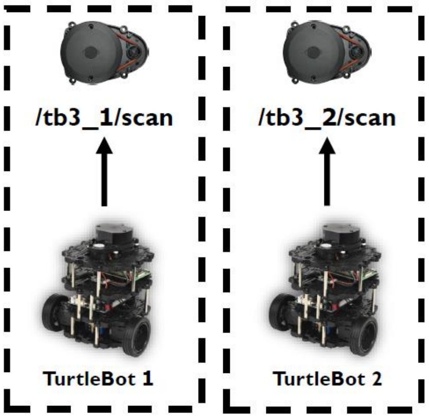


FIGURE 17 – Using ROS namespace to avoid conflicts between instances of the same robots

To use namespaces under ROS, it only required to specify the name you want to give to the robot in the *ROS_NAMESPACE* system variable and to specify the suffix used I, the *multi_robot* parameter of the *"turtlebot3_robot.launch"* file. However, after launching the robot, all topics had a suffix except for the */scan* topic. Indeed, all robots tried to publish the information from their LIDARs in the same topic, which led to errors. Since the problem was only in the LIDAR topic, I immediately looked in the LIDAR drivers, and more precisely in the source codes, to find the origin. The problem came from the driver, the main program was not using the *multi_robot* argument to define the topic name. After searching online and on forums, I learned that the problem could be caused by the LIDAR because it was changed at the beginning of 2022 and that the drivers would not have been fully updated yet. The fault was reported to the developers, and I only had to modify the source code slightly to correct the error. After this correction, the multi-agent configuration was functional, and several Turtlebot3 Burger could run at the same time on the network without creating conflicts. So, I was done with this part, or so I thought.

2.5.4 Adapting the built-in programs for multi-agent configuration

The last step to validate the deployment of the multi-agent configuration was to verify the correct functioning of the main programs of the ROS *Turtlebot3 Burger* package (teleop, SLAM and Autonomous Navigation). To do this I wanted to apply a protocol similar to the one in the part 2.3.1 Building the robots: realize an autonomous navigation in an OFFIS room. This involves using the teleop program to navigate to the room, the SLAM program to map the environment and the Autonomous Navigation program to navigate autonomously in the room. However, the test did not go as planned because the programs in the ROS package did not seem to support multi-agent configuration and more specifically the use of namespace under ROS. To correct this, I tried to modify the source code of each program to make them work in a multi-agent's configuration.

The first program I wanted to fix was the teleoperation program. The source code of this program is the file *turtlebot3_teleop_key.py* which is in the ROS package named *turtlebot3_teleop*, more precisely in the folder *nodes*. To understand, modify and test the different source codes, without impacting my installation, I preferred to create a dedicated ROS package in which I could copy the unworking programs to modify them. So, I copied the *turtlebot3_teleop_key.py* file into my ROS package named *turtlebot3_modified_package*. A quick read of the program allowed me to identify line 138 as the problematic line. This line corresponds to the creation of the ROS publisher and does not consider any parameter to add a suffix to the topic. So, I had to modify this part to ask the user about the suffix to use. The old and new versions of this program can be found in **Appendix 12**. After this simple modification the teleoperation program was now compatible with a multi-agent configuration.

The second program I modified was the SLAM program. Fixing this program was much more complex and time consuming than the first one. Indeed, the SLAM program uses 3 subprograms to run: *turtlebot3_remote.launch* (used to create the different coordinate systems of the problem, see Figure 18.), *turtlebot3_$(arg slam_methods).launch* (used to read the LIDAR data and apply a user-defined mapping algorithm. The argument slam_methods is the name of the algorithm chosen by the user) and *turtlebot3_$(arg slam_methods).rviz* (used to display the map in real time in a RVIZ window). So, I had to analyse these programs and understand how they are used by the SLAM algorithm to find the part(s) to correct. For a matter of time and page limit, I will not detail my research but immediately talk about the results. I was able to conclude that it was sufficient to add parameters to the SLAM program to allow the user to specify the suffix used and the names of the different coordinate systems used for the projections. Indeed, the problem seemed to be that the sub-programs could not find the various coordinate systems of the problem because they had a suffix. The old and new versions of the launch file can be found in **Appendix 13**. After this modification the SLAM program was now compatible with a multi-agent configuration.

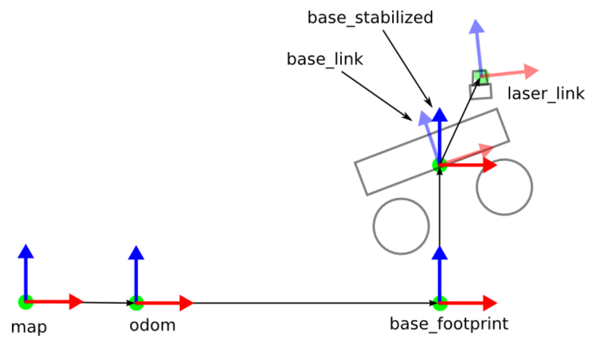


FIGURE 18 – Coordinate systems used by the SLAM algorithm

The last program I wanted to fix was the Autonomous Navigation program. This program corresponds to the file *turtlebot3_navigation.launch* in the ROS package *turtlebot3_navigation*. It uses 5 subprograms: *turtlebot3_remote.launch* (used to create the different coordinate systems of the problem, see Figure 18.), *map_server* (used to read the map produced by the SLAM program), *amcl.launch* (used to compute the best path to follow), *move_base.launch* (used to follow the computed path) and *rviz* (used to display the map, the robot and the path to follow). As with the SLAM program, I had to read and understand the dependencies of each of these sub-programs to fix the package. I tried to apply similar fixes that the SLAM program by adding parameters to specify the suffixes and the different coordinate systems used. I also modified the different parameter files (.yaml), but I finally couldn't get the package to work in a multi-agent configuration. I think the problem comes from the *move_base* package which is compiled and so I can't make any changes. Therefore, we must wait for a correction from the developers before the autonomous navigation program can work in a multi-agent configuration.

2.6 Working on the communication between software and R.O.S.

The last step of my internship was to work on the communication between ROS and programming software commonly used at the university. Indeed, the goal is that a maximum of students, of all levels, can use the *Turtlebot3 Burger*. As ROS is usually learned in the last years of the training cycle, it is necessary to provide a way for students to interact with the *Turtlebot3 Burger* without having to manipulate ROS directly. The aim is to integrate ROS communication modules into software used by students during the early years of their training cycle, allowing them to easily read and send information to the *Turtlebot3 Burger*. These modules will have to be intuitive enough for any student to use them without knowing all the mechanics of ROS but only its main foundations. For this internship, I will focus on two graphical programming environments: *MATLAB Simulink* and *Eclipse 4diac*.

2.6.1 MATLAB Simulink and R.O.S.

To begin with I decided to work on the communication between ROS and *MATLAB Simulink*. Indeed, knowing that *MATLAB* is a paying solution that is widely used in education and industry, it seemed obvious to me that a ROS communication module should already exist. After some research, I quickly found the existence of a *MATLAB* library called *ROS Toolbox* which allowed interaction with ROS topics in a rather intuitive way [5]. For example, to publish information on a ROS topic, you simply create an empty ROS message of the desired type (e.g. *geometry_msgs/Point*) and assign values to it using the assignment block. The modified message can then be published in the desired ROS topic using a *Publish block*. This example is illustrated in the Figure 19.

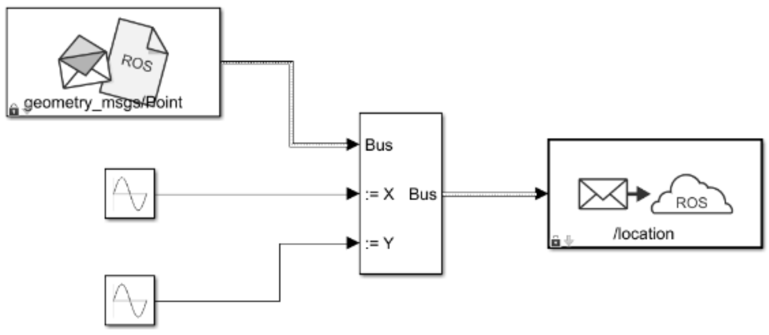


FIGURE 19 – Publishing data in ROS topic using MATLAB Simulink

Reading information from ROS topics is also possible but requires a slight subtlety. A notion of exchange frequency is introduced here. When *MATLAB Simulink* publishes information on a ROS topic, the information is sent at the frequency at which the simulation is running. However, in the case of reading information, the data may be published at a different frequency than the simulation one. Therefore, to avoid over-reading errors or a delay phenomenon of the information, it is necessary to use a particular block which is the *Enabled Subsystem block*. This block will allow the information to be read at the publication frequency and thus avoid possible errors. To continue with the previous example, to read the information published in the topic */location*, we need to create a *Subscribe bloc*k to read the information in the topic and couple it with the *Enabled Subsystem block* to synchronise the reading frequency with the publication frequency. This example is illustrated in Figure 20.

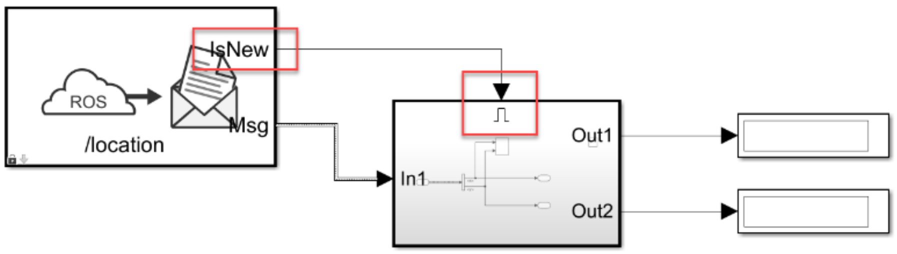


FIGURE 20 - Reading data from a ROS topic using MATLAB Simulink

The advantage of *MATLAB Simulink* is also the ability to create sub-blocks. Indeed, the aim is still to hide all the blocks related to ROS to make the process intuitive for students. By using sub-blocks, it is possible to transform Figure 19 into a block with two inputs, called for example X and Y, and to transform Figure 20 into a block with two outputs, which can also be called X and Y. Thus, by using the sub-blocks, the students do not know that the *ROS Toolbox* library is used because they only see blocks with inputs (which will be the commands sent to the robot) and outputs (which will be the information from the robot's sensors). To return to the *Turtlebot3 Burger* case, there is only one command to send: the desired linear and angular speed in the */cmd_vel* topic. So, to control a *Turtlebot3 Burger*, I just need to create a ROS publisher like the one in Figure 19 and hide it in a sub-block with two entries: Linear speed and Angular speed.

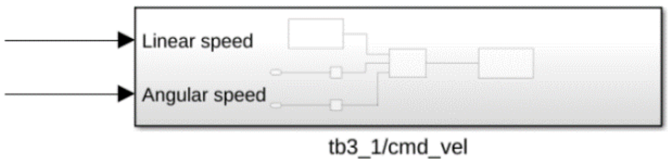


FIGURE 21 – Sending speed command to the Turtlebot3 Burger number 1 using MATLAB Simulink

2.6.2 Eclipse 4diac and R.O.S.

The second programming environment I worked on was *Eclipse 4diac*. *Eclipse 4diac* is not a charged solution but an open-source software, so it was not certain that a ROS communication module would be integrated in the software. However, as *Eclipse 4diac* is quite popular in Germany and used more and more in robotics, there was a chance that a contributor, or an official developer, had developed such a module. In fact, one of the developers did it but there was a trouble. The problem is that the module has not been updated since 2014 and is therefore only compatible with ROS Indigo and Ubuntu 14.04 (as a reminder, the nominal configuration for students would be Ubuntu 20.04 and ROS Noetic). As a result, several solutions were available to me: the first was to update the existing ROS module and the second was to think of another way to set up a communication with ROS. The first solution would be the most efficient but might be difficult and long to deploy. In the worst case, I might not be able to implement it before the end of my internship. The second solution offers more certainty. Indeed, several other protocols are supported by *Eclipse 4diac* and updated communication modules are available. It would therefore be possible to use one of these modules and couple it with a bridge to indirectly create a communication with ROS. This solution would be the easiest to implement in the remaining time but not the most optimal as setting up a bridge could create a slight delay of information which will have to be quantified.

In order to know which solution I should choose, I decided to talk to **Prof. RAUH** to find out the short-term needs of the university. Indeed, if the short-term goal is to use the project intensively, then a powerful ROS communication module is required, but if the goal is to have a demonstrator to show the capabilities of the project, then the second option is the most adapted. Following our discussion, we decided to develop the second option. Among the available protocols, I have isolated one in particular: the MQTT protocol.

The MQTT protocol is a TCP/IP communication protocol based on the publisher/subscriber model. Like ROS, it is possible to create topics in which publishers can share information and subscribers can read information. Unlike ROS, MQTT topics only carry information in the form of strings, so sending matrices or images is more complex. The fact that MQTT is built on the publisher/subscriber model means that MQTT clients incorporate the concept of call-back. When an MQTT client subscribes to a topic, it is possible to call a function each time new information is available, this principle is called call-back and it is also available with ROS. My idea is to use the call-back concept to realize the bridge I mentioned before. Indeed, if we wish to transmit information between *Eclipse 4diac* and ROS, we will have to set up a bridge which will allow the transfer of information between the MQTT topics and the ROS topics. For that, the bridge will have to use an MQTT client to exchange with the MQTT topics and a ROS client to exchange with the ROS topics. My idea is to use the call-back concept so that when a new information is available on an MQTT topic (or an ROS topic), the MQTT subscriber (respectively the ROS subscriber) calls the ROS publisher (respectively the MQTT publisher) thanks to the call-back concept to transmit the information to it so that it can publish it.

Let's take a simple example, a user wants to process information from one of the *Turtlebot3 Burger* sensors with *Eclipse 4diac*. Let's assume that the information is published in the ROS */sensor* topic, so the goal is for the bridge to transfer the information to the MQTT */sensor* topic. By doing so, the user will only have to use the MQTT module of *Eclipse 4diac* to read the information, which creates a communication between ROS and *Eclipse 4diac*. The bridge will therefore instantiate an ROS client that will subscribe to the ROS */sensor* topic and an MQTT client that will publish to the MQTT */sensor* topic. By using the call-back function of the ROS subscriber, it is possible, at each acquisition of the sensor, to read the new published value and to call a function which will take as argument this new value, and which will use the MQTT publisher to publish the information in the MQTT */sensor* topic. This principle can also be applied in the other direction (MQTT to ROS) and allows to create an indirect communication between ROS and *Eclipse 4diac* and thus offers users to control the *Turtlebot3 Burger* through *Eclipse 4diac*.

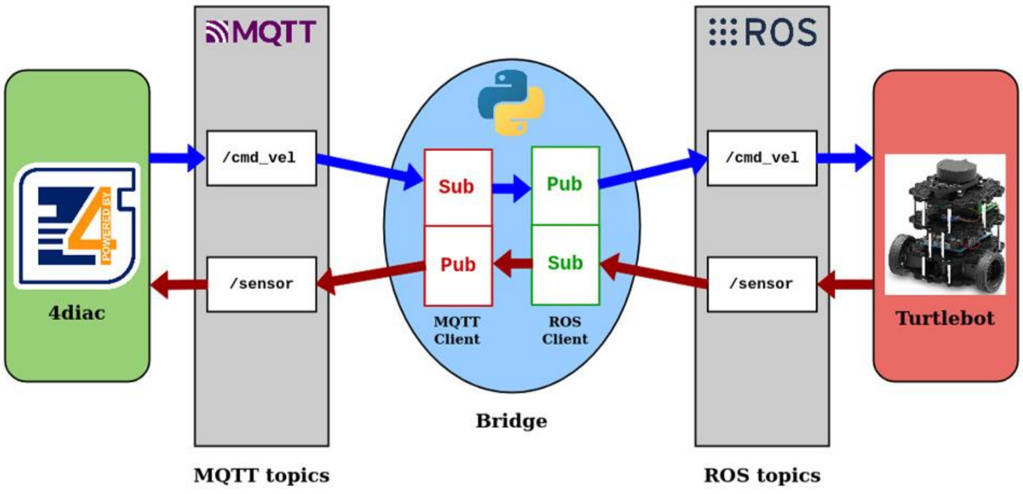


FIGURE 22 – Concept of the bridge between MQTT and ROS to create a communication between Eclipse 4diac and the Turtlebot3 Burger

3. Results and lessons learned

3.1 Culture

These 16 weeks spent in Germany were very formative and enriching for me from a cultural point of view. The fact that I had a job and fixed hours allowed me to really immerse myself in the German culture by living the daily life of the population. Indeed, visiting a country and living in a country are two very different things. In order to succeed in living in Germany, I quickly decided to learn German using alternative methods. Standard language courses are usually very restrictive for me as they involve fixed durations, at sometimes restrictive times, and with a continuous workload. For this reason, I decided to learn German using *Duolingo* in order to fit my lessons to my schedule. By working on pronunciation with my German colleagues, I was able to master the basics of everyday life quite quickly which changed my daily life.

Working in an international research centre allowed me to meet people from all walks of life during coffee breaks, even French people. I was also lucky enough to have very open and friendly colleagues who helped me to get to know the German way of life. So, this experience was really enriching and rich in experiences and could encourage me to work in Germany in the future of my career.

3.2 Long Night of Digitization

As told before in this report, during my internship, *OFFIS* organised a major event called *"Long Night of Digitization"* in partnership with *BTC Business Technology Consulting AG*, an IT consulting company based in Oldenburg, Germany. The aim of the event was to reconnect the Public and the Industry with the Research after the COVID19 crisis by demonstrating the importance of digital technologies in our lives and in the future. On this occasion, all the departments had a stand to explain their work and I was no exception. My objective was to capture the attention of young students and of the public with the *Turtlebot3 Burger*, allowing them to control the robots with a game controller and camera feedback. I had therefore to add a camera on the robot and create a program to teleoperate the robots with a game controller.

During this event, I had to present my work to a very diverse audience and make them understand the interest of such a project for education. The language barrier was a difficulty that I tried to overcome to transmit my passion. A large audience was interested in what I had to say and some of them even asked me for advice on how they could discover robotics themselves. The director of the university also came to the stand and showed a strong interest in the project, which for him will be an excellent pedagogical support for the students. This evening was a success and will be engraved in my memory for many years to come.

FIGURE 23 – Presenting the project to a father and his son

3.3 Work Done

By the end of August 2022, all the objectives of my internship had been met. Indeed, the *Turtlebot3 Burger* were deployed and operational in a multi-agent configuration. Equipped with a modular and versatile hardware structure, they could now adapt to the needs of different users with very little deployment time. The network was also fully operational and, as promised, easily transportable and deployable. The documentation and user guides made it easy for someone unfamiliar with the project to pick it up and deploy it. Students can already work with the *Turtlebot3 Burger* using different software such as *MATLAB Simulink* or *Eclipse 4diac*. After presentation and validation of the results to **Prof. RAUH** and the different members of the *Computer Science department* of the Oldenburg university, I concluded that the project had been successfully completed.

3.4 Future work

Despite the results produced during these 4 months, there are still elements to improve or even develop:

- First, it will be necessary to understand why the NTP client on the *Turtlebot3 Burger* refuses to connect to the NTP server. Ideally, the problem should be identified and solved to allow a more accurate time synchronisation between the different agents.

- It will also be necessary to monitor possible updates to the ROS package for autonomous navigation so that it can support a multi-agent configuration and more precisely the use of namespace under ROS

- During the different uses of the project, it will be necessary to enrich the ROS package *hat_brick_pkg* with programs allowing to easily integrate new types of sensors such as temperature sensors, pressure sensors, ... etc.

- To facilitate the interaction between ROS and *Eclipse 4diac*, it would be necessary to improve the bridge and to adapt it especially to the project. Indeed, it is currently necessary to manually specify the name of each topic that we wish to monitor, whereas we already know the name of the topics used by the *Turtlebot3 Burger*. The best solution is still to update the ROS communication module to ROS Noetic.

- As ROS1 is nearing the end of its development in favour of ROS2, it would be interesting to adapt my work by upgrading the *Turtlebot3 Burger* to ROS2. *ROBOTIS* already offers a ROS2 version of the *Turtlebot3 Burger* drivers on its website.

4. Conclusion

During this 16-week internship in Germany, I was able to experience working in an international and professional context. It is hard, even with hindsight, to know if this change has impacted negatively or positively my ability to work and to focus. However, I must admit that the change in my habits and daily life was quite complicated during the first weeks, thus probably impacting my productivity. However, I have pleasant memories of this internship, which I associate with a strong cultural opening, a discovery of the German language and a new life experience. This internship gave me a lot on a technical level, as well as on a human level, and this will have an impact on the engineer I will become.

This internship also allowed me to discover the problems and issues related to multi-agent environments, while applying the technical and human knowledge that I acquired during my training at *ENSTA Bretagne*. I was able to learn how to deploy a network, to build a robot, to use different ROS packages, to design 3D supports and to work on communications between different software. This internship was therefore very complete and allowed me to develop all the technical skills that one can expect from a robotics engineer and that will probably be useful in the future.

Glossary

ENSTA	Ecole Nationale Supérieure de Techniques Avancées
R.O.S.	Robot Operating System (see 2.2.2 Robot Operating System (R.O.S.) for more info)
IETA	Ingénieur des Etudes et Techniques de l'Armement
LIDAR	Light Detection and Ranging or Laser Imaging, Detection, and Ranging
IMU	Inertial Measurement Unit
SLAM	Simultaneously Localisation and Mapping
SSID	Service Set Identifier
NTP	Network Time Protocol
RTC	Real Time Clock
CAD	Computer Aided Design

Bibliography

[1] ROBOTIS, "Turtlebot3 Burger E-manual," 2022.

[2] C. Reusch, "Basic facts about WLAN standards in North America and Europe," in *crnetpackets.com*, 2015.

[3] K. Buzdar, "How to Install NTP Server and Client(s) on Ubuntu 20.04 LTS," in *VITUX - Linux Compendium*, 2021.

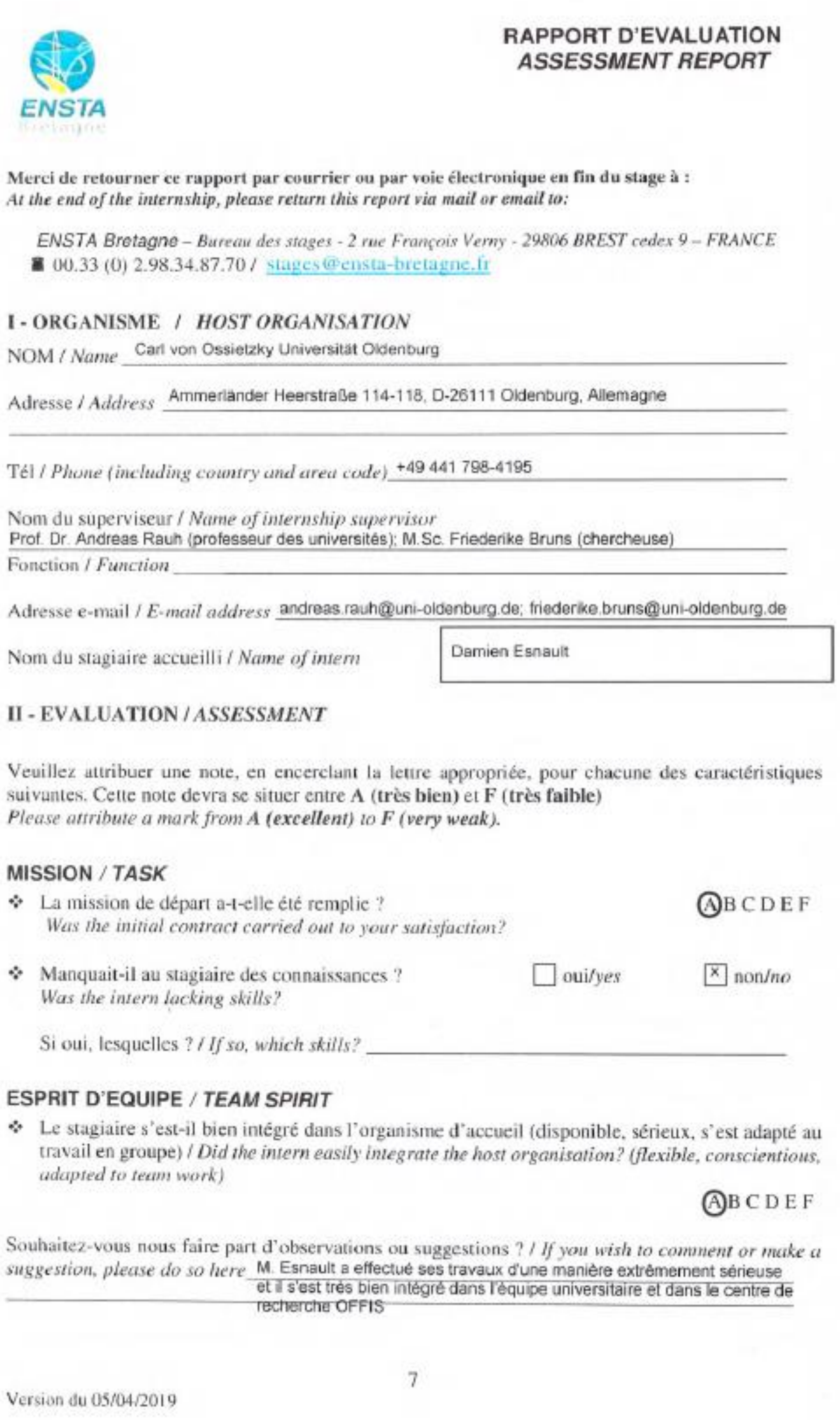
[4] C. WILLIAMS and A. SCHROEDER, "Utilizing ROS 1 and the Turtlebot3 in a Multi-Robot System," The University of Toledo, Toledo, OH USA, 2020.

[5] MathWorks, "Get Started with ROS in Simulink," in *MathWorks documentation*.

Table of figures

Appendices

Appendix 1: Assessment report



ENSTA Bretagne

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport par courrier ou par voie électronique en fin du stage à :
At the end of the internship, please return this report via mail or email to:

ENSTA Bretagne – *Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE*
00.33 (0) 2.98.34.87.70 / stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* Carl von Ossietzky Universität Oldenburg

Adresse / *Address* Ammerländer Heerstraße 114-118, D-26111 Oldenburg, Allemagne

Tél / *Phone (including country and area code)* +49 441 798-4195

Nom du superviseur / *Name of internship supervisor*
Prof. Dr. Andreas Rauh (professeur des universités); M.Sc. Friederike Bruns (chercheuse)

Fonction / *Function*

Adresse e-mail / *E-mail address* andreas.rauh@uni-oldenburg.de; friederike.bruns@uni-oldenburg.de

Nom du stagiaire accueilli / *Name of intern* Damien Esnault

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)**
*Please attribute a mark from **A (excellent)** to **F (very weak)**.*

MISSION / *TASK*

- La mission de départ a-t-elle été remplie ? *Was the initial contract carried out to your satisfaction?* [✓ A] B C D E F
- Manquait-il au stagiaire des connaissances ? *Was the intern lacking skills?* ☐ oui/*yes* [✓ non/*no*]

  Si oui, lesquelles ? / *If so, which skills?*

ESPRIT D'EQUIPE / *TEAM SPIRIT*

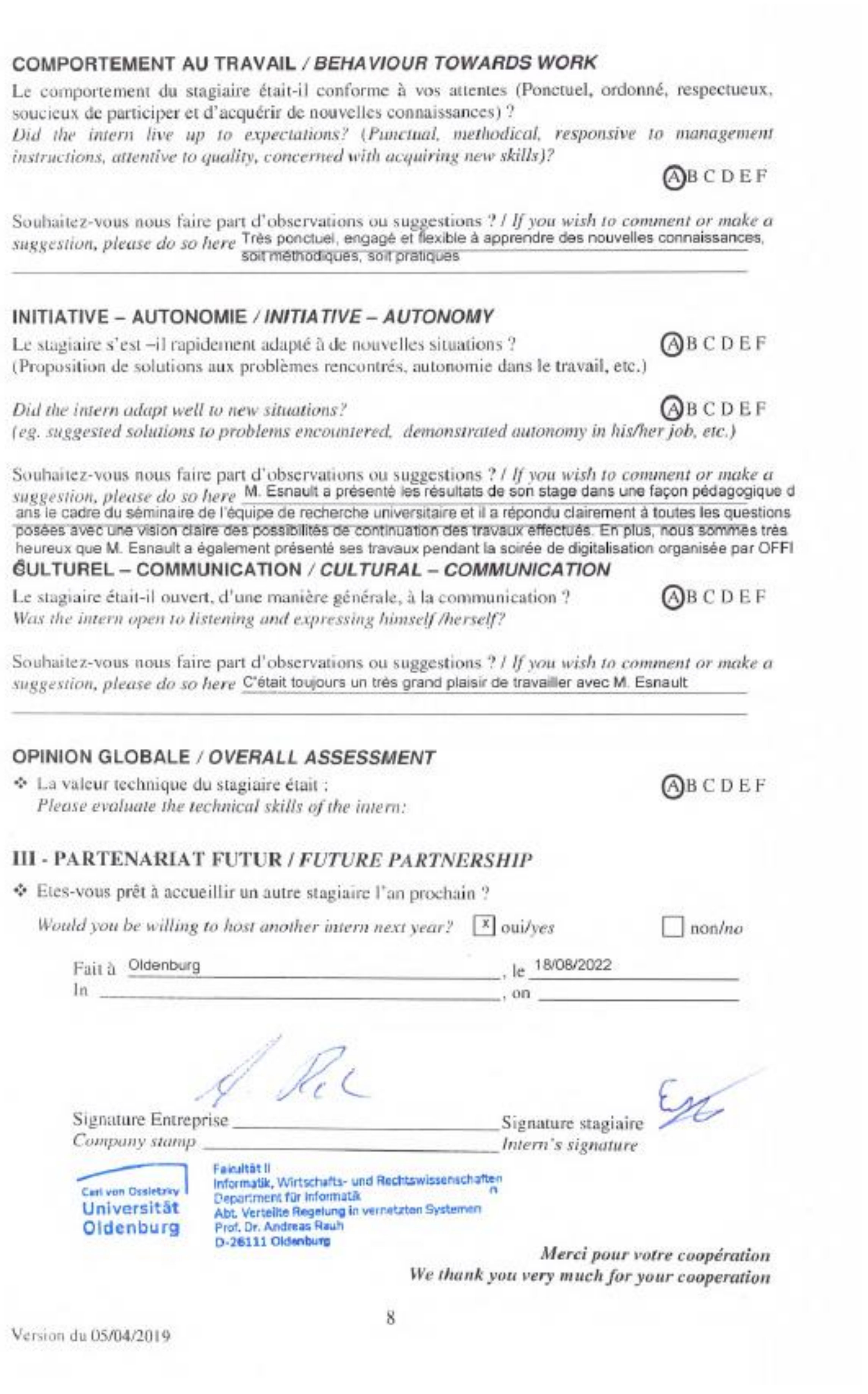
- Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the intern easily integrate the host organisation? (flexible, conscientious, adapted to team work)* [✓ A] B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* M. Esnault a effectué ses travaux d'une manière extrêmement sérieuse et il s'est très bien intégré dans l'équipe universitaire et dans le centre de recherche OFFIS

7

Version du 05/04/2019

ENSTA BRETAGNE



COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

Le comportement du stagiaire était-il conforme à vos attentes (Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances) ?
Did the intern live up to expectations? (Punctual, methodical, responsive to management instructions, attentive to quality, concerned with acquiring new skills)?

[✓ A] B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Très ponctuel, engagé et flexible à apprendre des nouvelles connaissances, soit méthodiques, soit pratiques

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? (Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.) [✓ A] B C D E F

Did the intern adapt well to new situations? (eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.) [✓ A] B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* M. Esnault a présenté les résultats de son stage dans une façon pédagogique d
ans le cadre du séminaire de l'équipe de recherche universitaire et il a répondu clairement à toutes les questions
posées avec une vision claire des possibilités de continuation des travaux effectués. En plus, nous sommes très
heureux que M. Esnault a également présenté ses travaux pendant la soirée de digitalisation organisée par OFFI

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ?
Was the intern open to listening and expressing himself /herself? [✓ A] B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* C'était toujours un très grand plaisir de travailler avec M. Esnault

OPINION GLOBALE / *OVERALL ASSESSMENT*

- La valeur technique du stagiaire était :
  Please evaluate the technical skills of the intern: [✓ A] B C D E F

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

- Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?
  Would you be willing to host another intern next year? [✓ oui/*yes*] non/*no*

Fait à Oldenburg, le 18/08/2022
In ______, *on* ______

[illegible]

Signature Entreprise
Company stamp

Signature stagiaire
Intern's signature

[illegible]

Carl von Ossietzky Universität Oldenburg
Fakultät II
Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik
Abt. Verteilte Regelung in vernetzten Systemen
Prof. Dr. Andreas Rauh
D-26111 Oldenburg

Merci pour votre coopération
We thank you very much for your cooperation

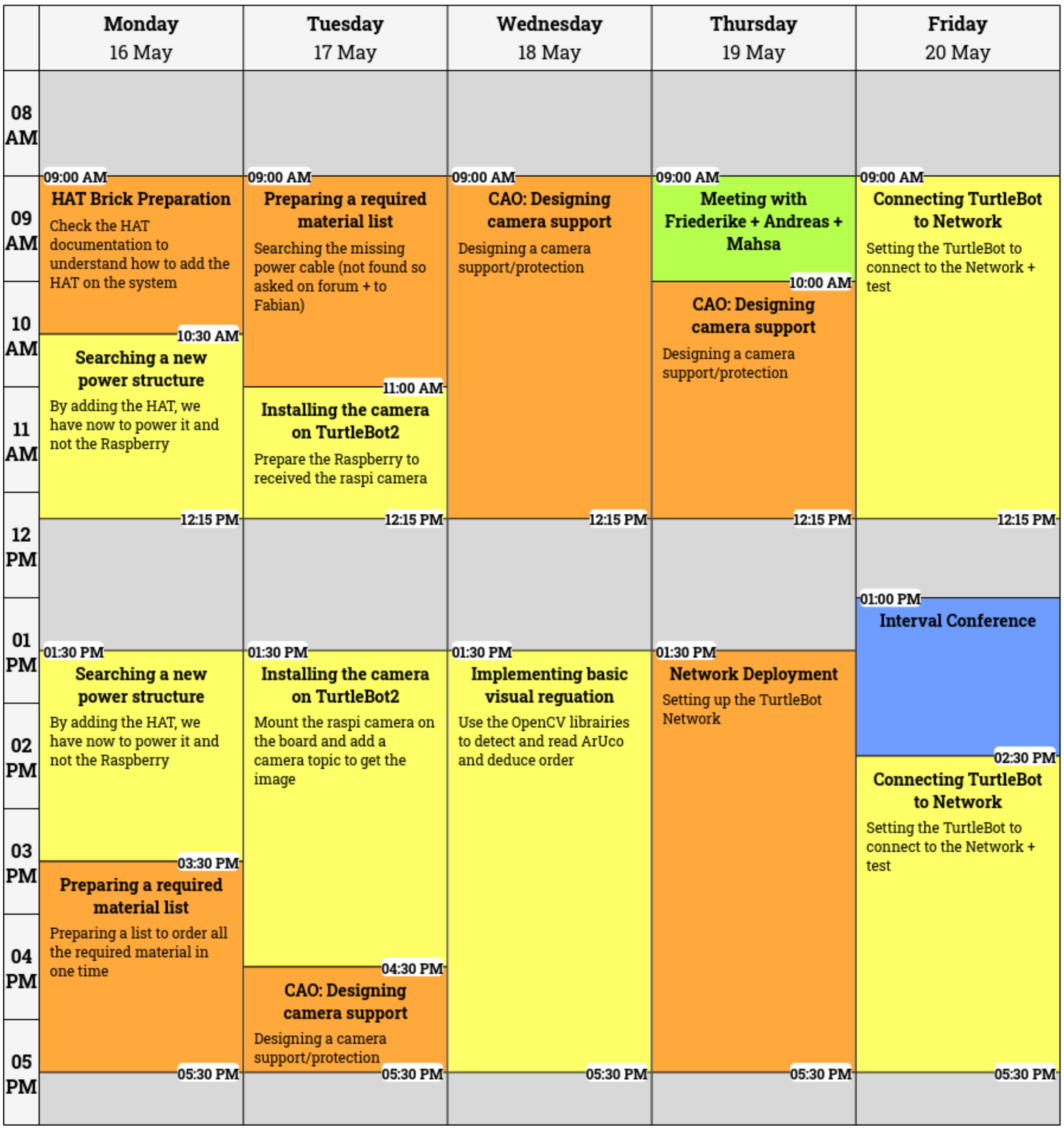
8

Version du 05/04/2019

ENSTA BRETAGNE

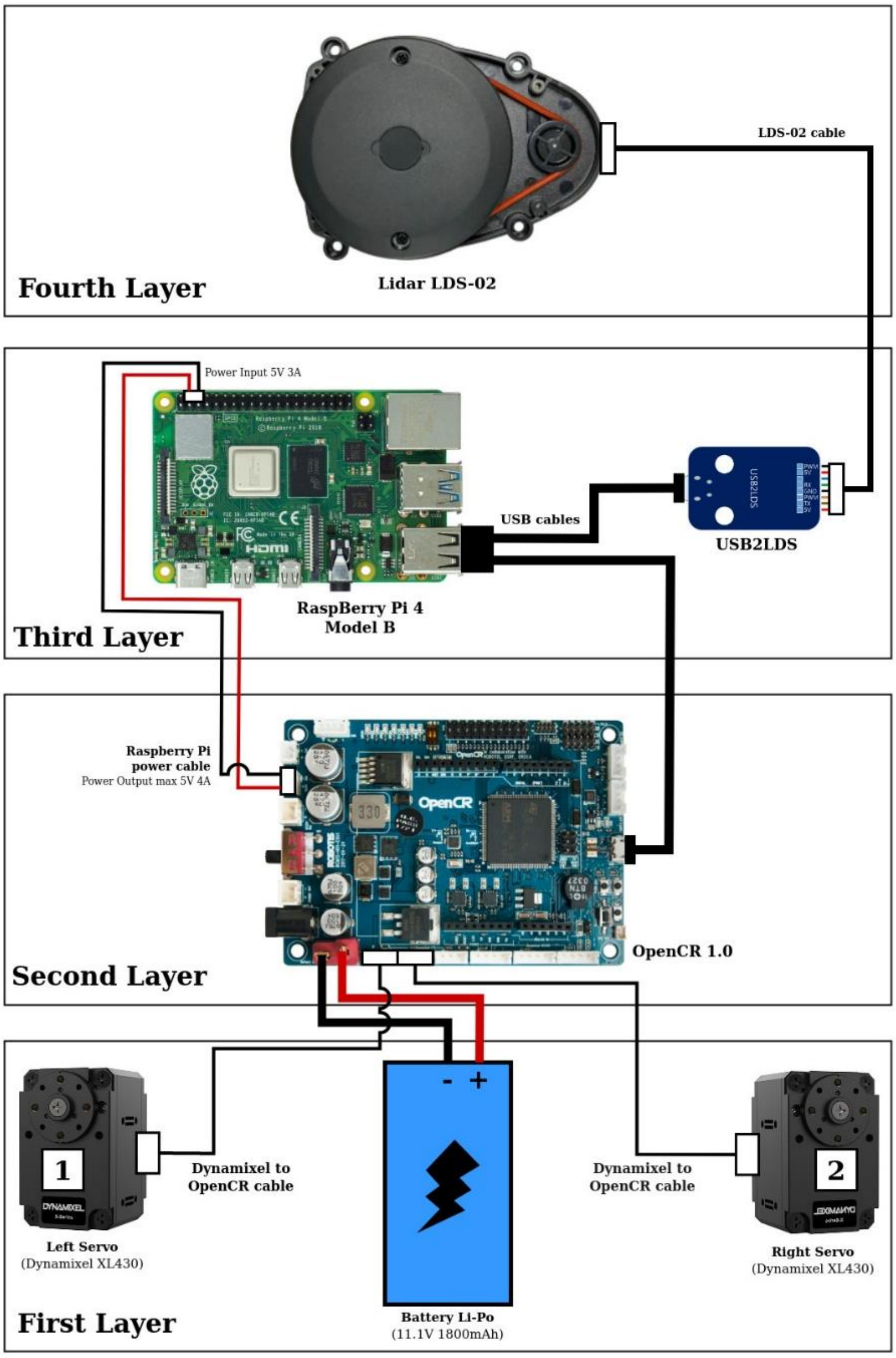
Appendix 2: Example of a weekly schedule

16/05/2020 - 20/05/2022 : Work Schedule

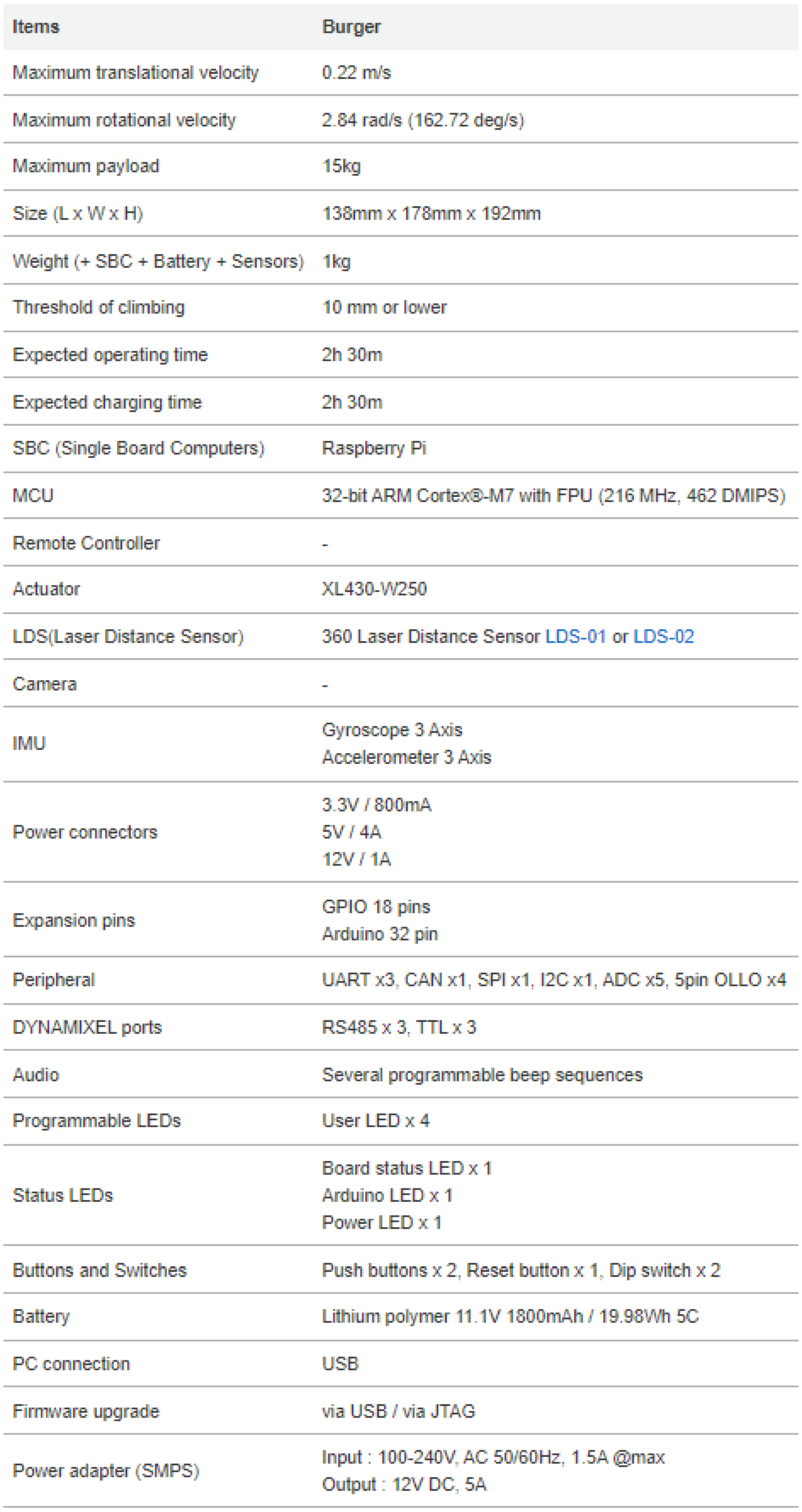


	Monday 16 May	**Tuesday** 17 May	**Wednesday** 18 May	**Thursday** 19 May	**Friday** 20 May
08 AM					
09 AM	09:00 AM **HAT Brick Preparation** Check the HAT documentation to understand how to add the HAT on the system	09:00 AM **Preparing a required material list** Searching the missing power cable (not found so asked on forum + to Fabian)	09:00 AM **CAO: Designing camera support** Designing a camera support/protection	09:00 AM **Meeting with Friederike + Andreas + Mahsa**	09:00 AM **Connecting TurtleBot to Network** Setting the TurtleBot to connect to the Network + test
10 AM	10:30 AM **Searching a new power structure** By adding the HAT, we have now to power it and not the Raspberry			10:00 AM **CAO: Designing camera support** Designing a camera support/protection	
11 AM		11:00 AM **Installing the camera on TurtleBot2** Prepare the Raspberry to received the raspi camera			
12 PM	12:15 PM	12:15 PM	12:15 PM	12:15 PM	12:15 PM
01 PM	01:30 PM **Searching a new power structure** By adding the HAT, we have now to power it and not the Raspberry	01:30 PM **Installing the camera on TurtleBot2** Mount the raspi camera on the board and add a camera topic to get the image	01:30 PM **Implementing basic visual reguation** Use the OpenCV librairies to detect and read ArUco and deduce order	01:30 PM **Network Deployment** Setting up the TurtleBot Network	01:00 PM **Interval Conference**
02 PM					02:30 PM **Connecting TurtleBot to Network** Setting the TurtleBot to connect to the Network + test
03 PM	03:30 PM **Preparing a required material list** Preparing a list to order all the required material in one time				
04 PM		04:30 PM **CAO: Designing camera support** Designing a camera support/protection			
05 PM	05:30 PM	05:30 PM	05:30 PM	05:30 PM	05:30 PM

Appendix 3: Example of a visual support (initial hardware structure)

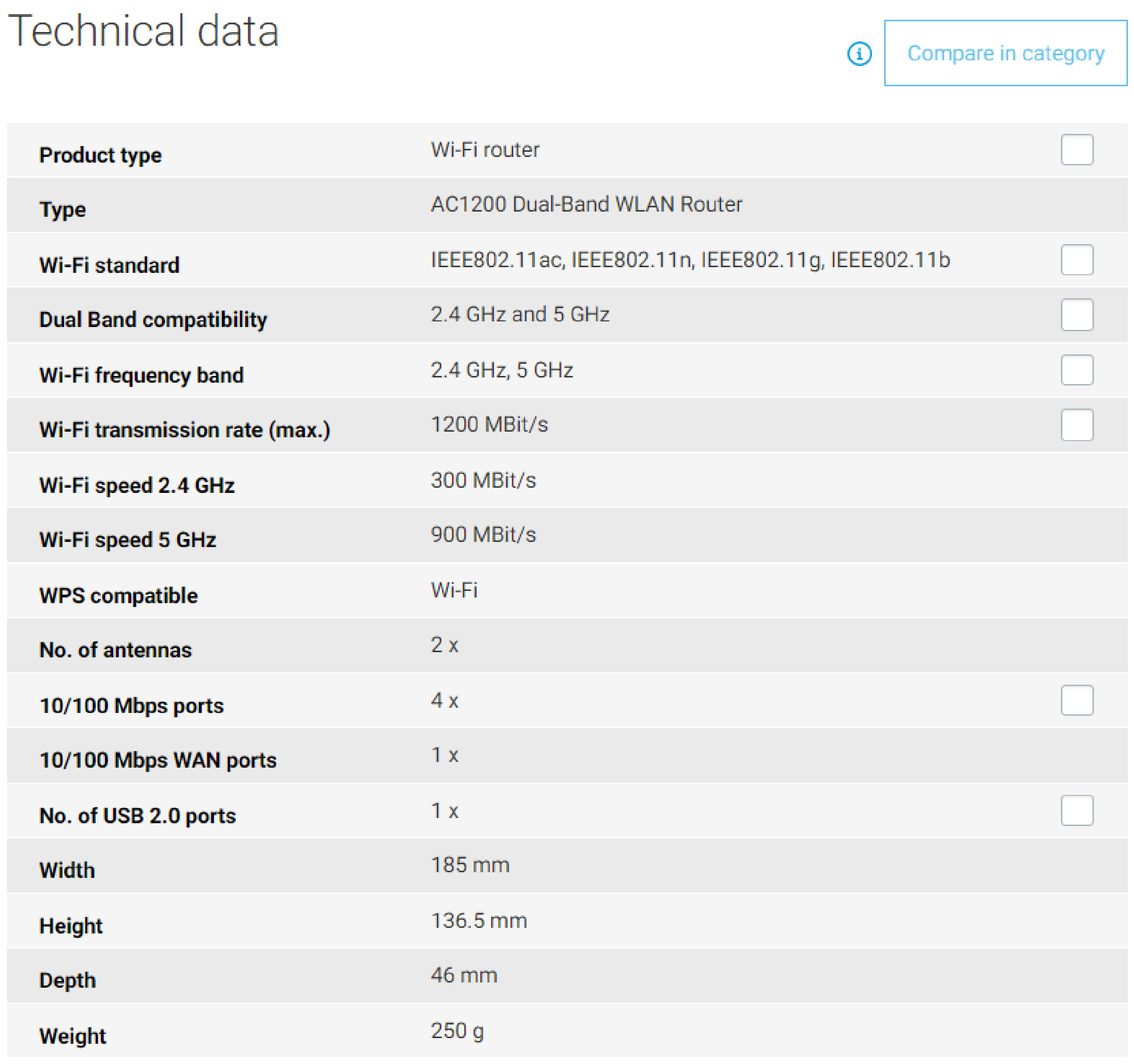


Appendix 4: Turtlebot3 Burger specifications



Items	Burger
Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)
Maximum payload	15kg
Size (L x W x H)	138mm x 178mm x 192mm
Weight (+ SBC + Battery + Sensors)	1kg
Threshold of climbing	10 mm or lower
Expected operating time	2h 30m
Expected charging time	2h 30m
SBC (Single Board Computers)	Raspberry Pi
MCU	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)
Remote Controller	-
Actuator	XL430-W250
LDS(Laser Distance Sensor)	360 Laser Distance Sensor LDS-01 or LDS-02
Camera	-
IMU	Gyroscope 3 Axis Accelerometer 3 Axis
Power connectors	3.3V / 800mA 5V / 4A 12V / 1A
Expansion pins	GPIO 18 pins Arduino 32 pin
Peripheral	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4
DYNAMIXEL ports	RS485 x 3, TTL x 3
Audio	Several programmable beep sequences
Programmable LEDs	User LED x 4
Status LEDs	Board status LED x 1 Arduino LED x 1 Power LED x 1
Buttons and Switches	Push buttons x 2, Reset button x 1, Dip switch x 2
Battery	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C
PC connection	USB
Firmware upgrade	via USB / via JTAG
Power adapter (SMPS)	Input : 100-240V, AC 50/60Hz, 1.5A @max Output : 12V DC, 5A

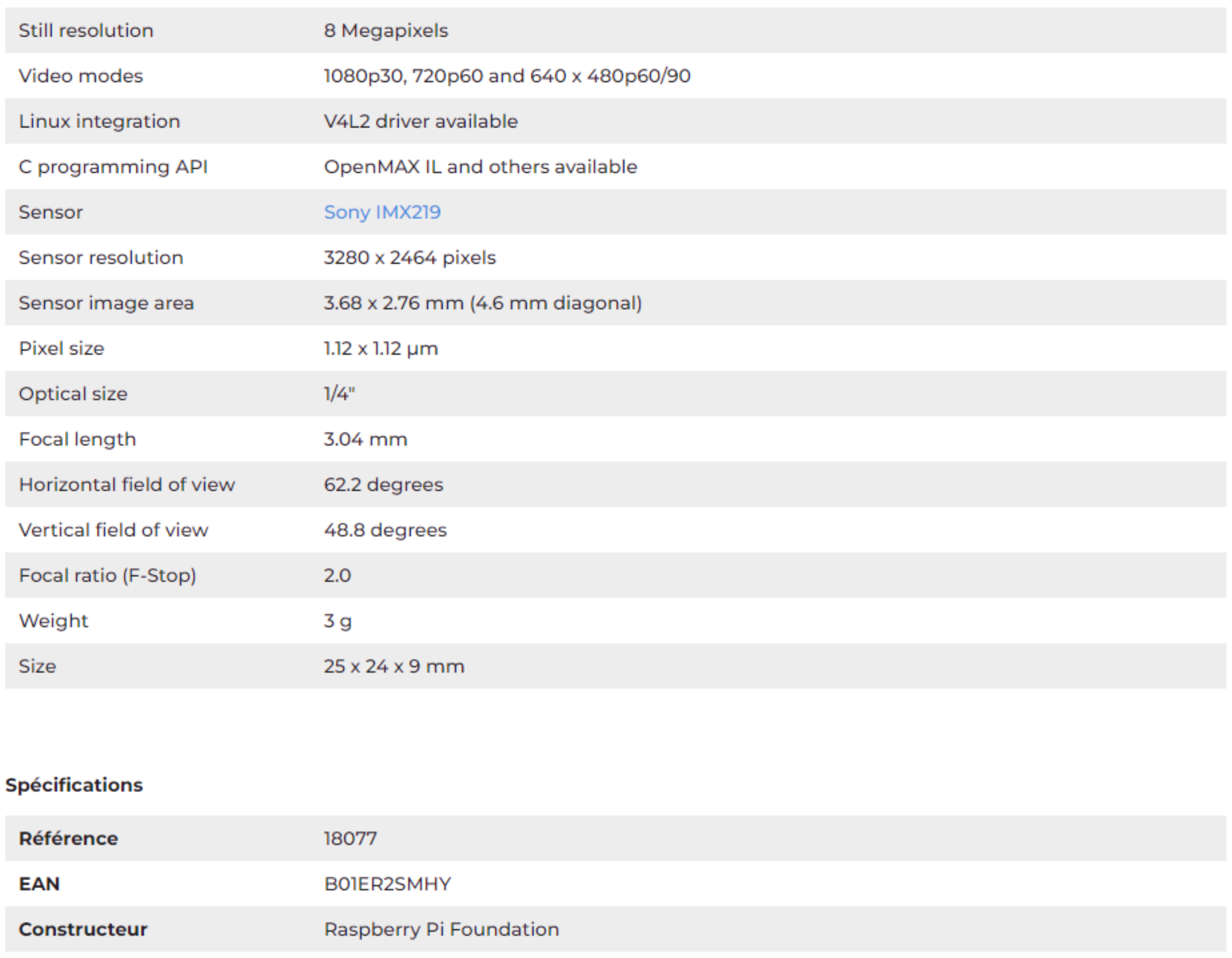
Appendix 5: Router datasheet



Technical data

Product type	Wi-Fi router	☐
Type	AC1200 Dual-Band WLAN Router	
Wi-Fi standard	IEEE802.11ac, IEEE802.11n, IEEE802.11g, IEEE802.11b	☐
Dual Band compatibility	2.4 GHz and 5 GHz	☐
Wi-Fi frequency band	2.4 GHz, 5 GHz	☐
Wi-Fi transmission rate (max.)	1200 MBit/s	☐
Wi-Fi speed 2.4 GHz	300 MBit/s	
Wi-Fi speed 5 GHz	900 MBit/s	
WPS compatible	Wi-Fi	
No. of antennas	2 x	
10/100 Mbps ports	4 x	☐
10/100 Mbps WAN ports	1 x	
No. of USB 2.0 ports	1 x	☐
Width	185 mm	
Height	136.5 mm	
Depth	46 mm	
Weight	250 g	

Appendix 6: Raspberry Pi NoIR camera datasheet

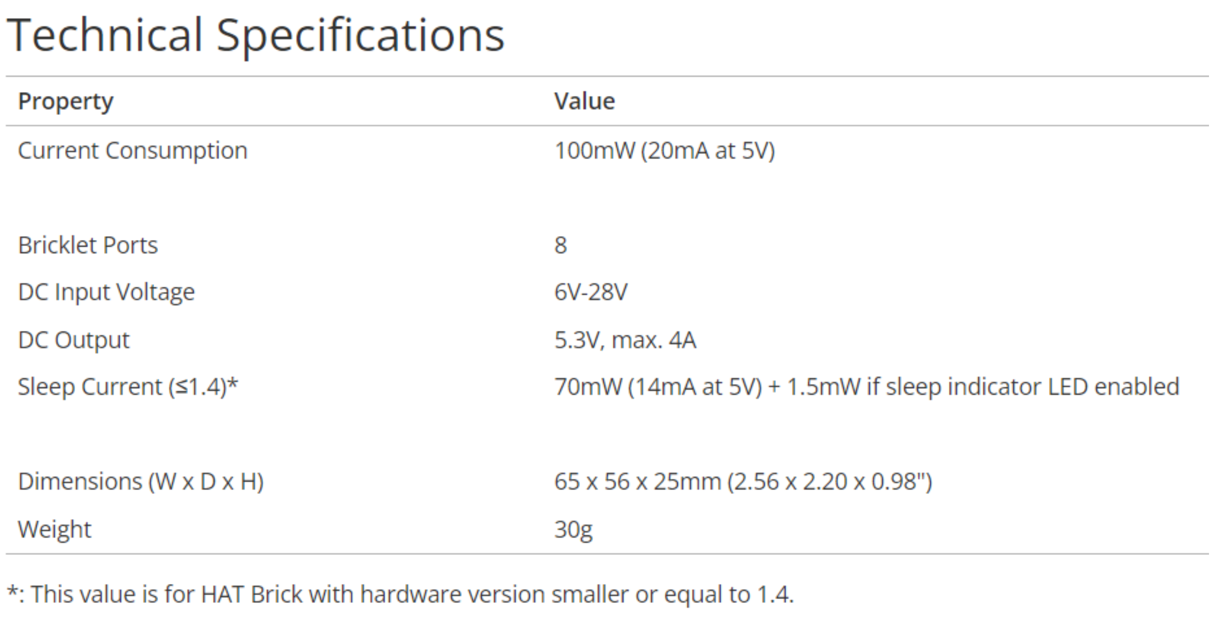


Still resolution	8 Megapixels
Video modes	1080p30, 720p60 and 640 x 480p60/90
Linux integration	V4L2 driver available
C programming API	OpenMAX IL and others available
Sensor	Sony IMX219
Sensor resolution	3280 x 2464 pixels
Sensor image area	3.68 x 2.76 mm (4.6 mm diagonal)
Pixel size	1.12 x 1.12 µm
Optical size	1/4"
Focal length	3.04 mm
Horizontal field of view	62.2 degrees
Vertical field of view	48.8 degrees
Focal ratio (F-Stop)	2.0
Weight	3 g
Size	25 x 24 x 9 mm

Spécifications

Référence	18077
EAN	B01ER2SMHY
Constructeur	Raspberry Pi Foundation

Appendix 7: Tinker Forge HAT Brick datasheet

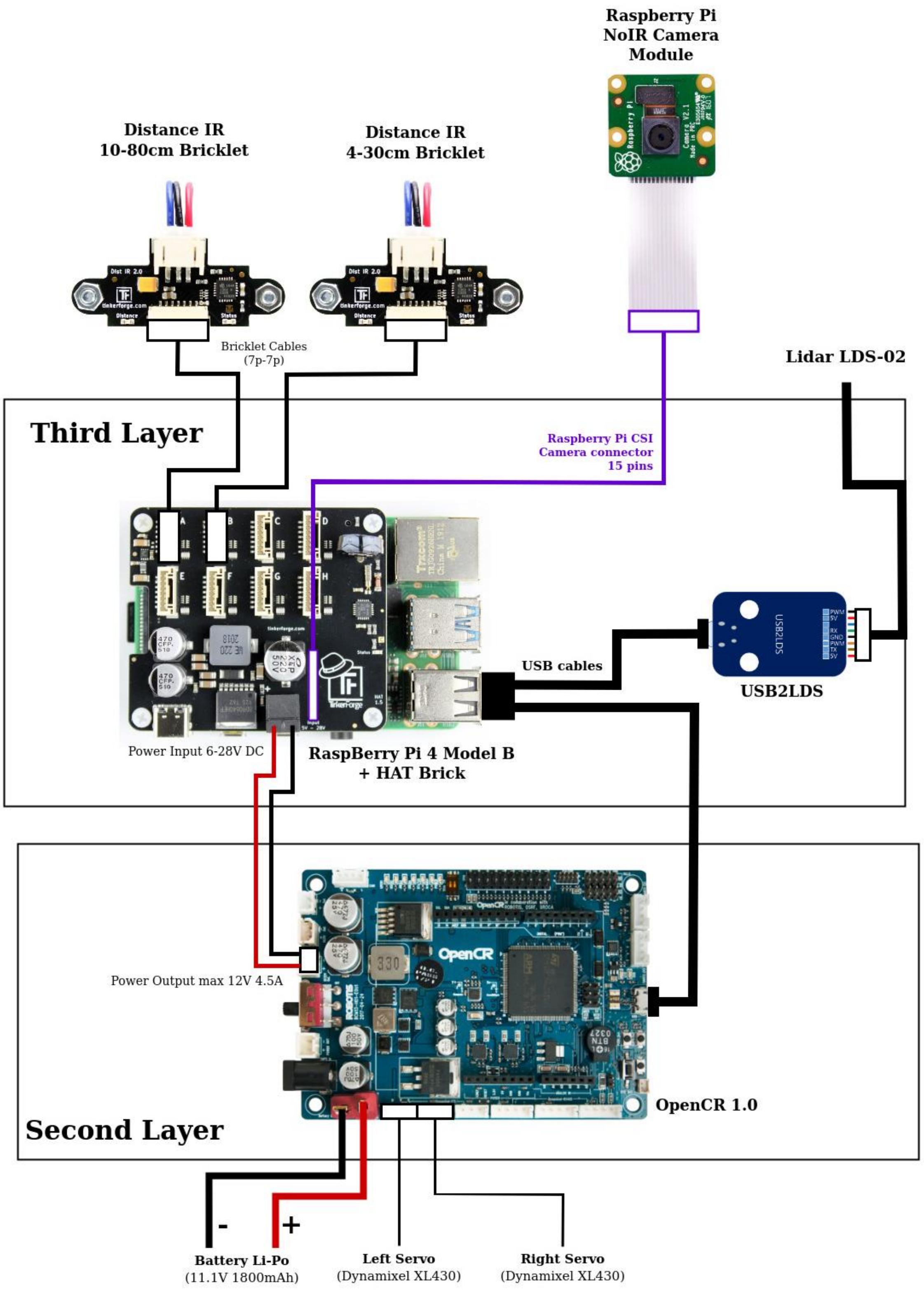


Technical Specifications

Property	Value
Current Consumption	100mW (20mA at 5V)
Bricklet Ports	8
DC Input Voltage	6V-28V
DC Output	5.3V, max. 4A
Sleep Current (≤1.4)*	70mW (14mA at 5V) + 1.5mW if sleep indicator LED enabled
Dimensions (W x D x H)	65 x 56 x 25mm (2.56 x 2.20 x 0.98")
Weight	30g

*: This value is for HAT Brick with hardware version smaller or equal to 1.4.

Appendix 8: New Turtlebot3 Burger hardware structure


Raspberry Pi NoIR Camera Module
Distance IR 10-80cm Bricklet
Distance IR 4-30cm Bricklet
Bricklet Cables (7p-7p)
Lidar LDS-02
Third Layer
Raspberry Pi CSI Camera connector 15 pins
USB cables
USB2LDS
Power Input 6-28V DC
RaspBerry Pi 4 Model B + HAT Brick
Power Output max 12V 4.5A
OpenCR 1.0
Second Layer
-
+
Battery Li-Po (11.1V 1800mAh)
Left Servo (Dynamixel XL430)
Right Servo (Dynamixel XL430)

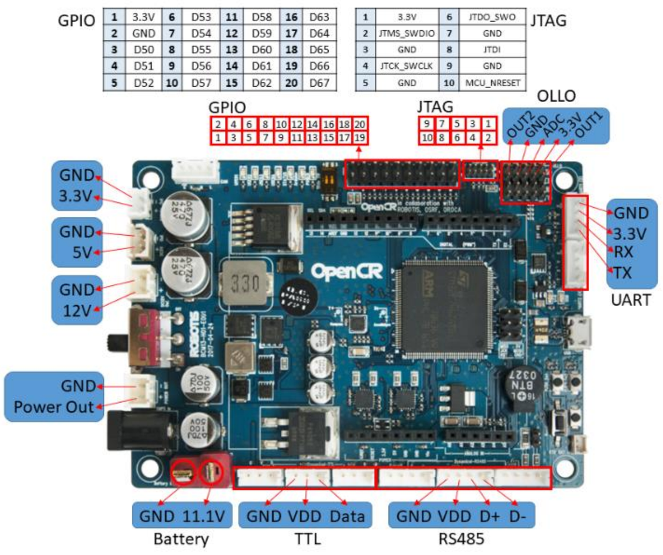
Appendix 9: OpenCR 1.0 datasheet

2. Specifications

Items	Specifications
Microcontroller	STM32F746ZGT6 / 32-bit ARM Cortex®-M7 with FPU (216MHz, 462DMIPS) Reference Manual, Datasheet
Sensors	(**Discontinued**) Gyroscope 3Axis, Accelerometer 3Axis, Magnetometer 3Axis (MPU9250) (**New**) 3-axis Gyroscope, 3-Axis Accelerometer, A Digital Motion Processor™ (ICM-20648)
Programmer	ARM Cortex 10pin JTAG/SWD connector USB Device Firmware Upgrade (DFU) Serial
Digital I/O	32 pins (L 14, R 18) *Arduino connectivity 5Pin OLLO x 4 GPIO x 18 pins PWM x 6 I2C x 1 SPI x 1
Analog INPUT	ADC Channels (Max 12bit) x 6
Communication Ports	USB x 1 (Micro-B USB connector/USB 2.0/Host/Peripheral/OTG) TTL x 3 (B3B-EH-A / DYNAMIXEL) RS485 x 3 (B4B-EH-A / DYNAMIXEL) UART x 2 (20010WS-04) CAN x 1 (20010WS-04)
LEDs and buttons	LD2 (red/green) : USB communication User LED x 4 : LD3 (red), LD4 (green), LD5 (blue) User button x 2 Power LED : LD1 (red, 3.3 V power on) Reset button x 1 (for power reset of board) Power on/off switch x 1
Input Power Sources	5 V (USB VBUS), 5-24 V (Battery or SMPS) Default battery : LI-PO 11.1V 1,800mAh 19.98Wh Default SMPS : 12V 4.5A External battery Port for RTC (Real Time Clock) (Molex 53047-0210)
Input Power Fuse	125V 10A LittleFuse 0453010
Output Power Sources	*12V max 4.5A(SMW250-02) *5V max 4A(5267-02A), 3.3V@800mA(20010WS-02)
Dimensions	105(W) X 75(D) mm
Weight	60g

* 5V power source is supplied from regulated 12V output. Total power consumption on 12V and 5V ports should not exceed 55W.

3. Layout/Pin Map



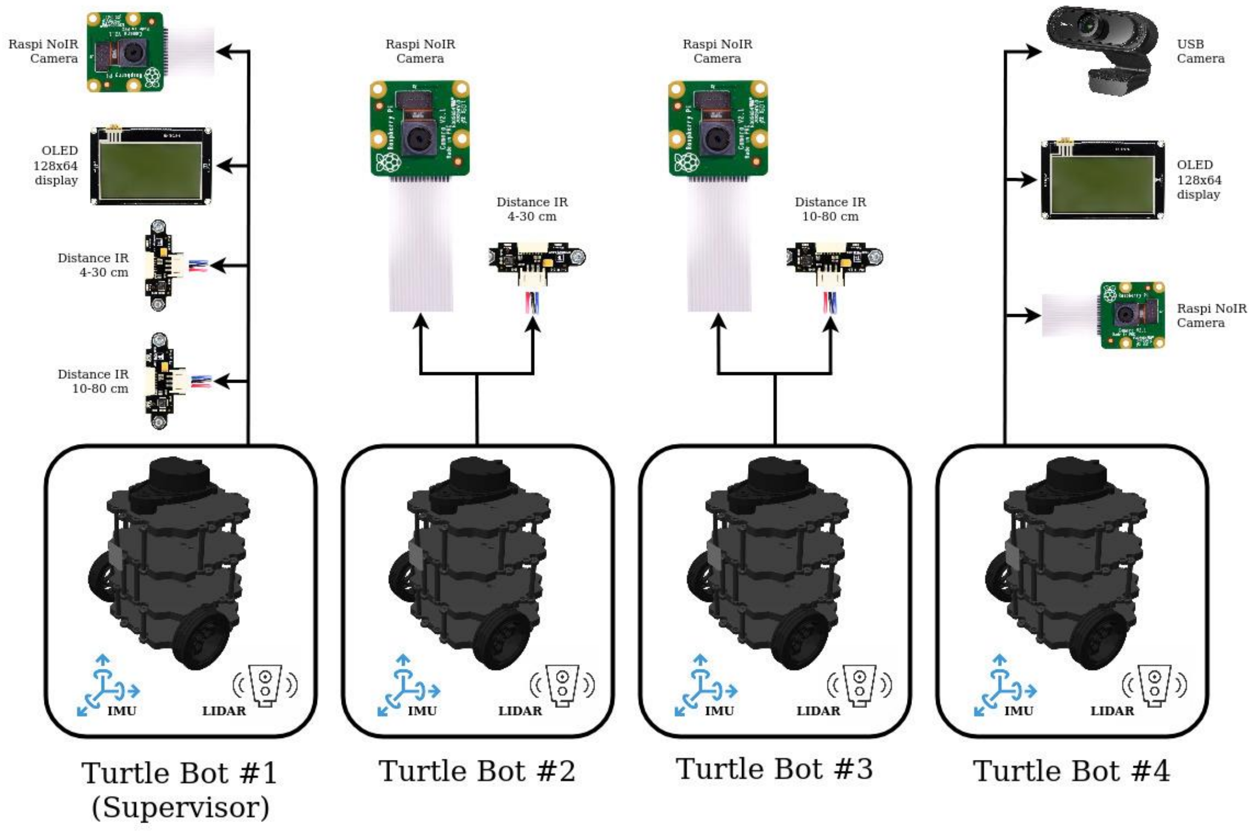
GPIO

1	3.3V	6	D53	11	D58	16	D63
2	GND	7	D54	12	D59	17	D64
3	D50	8	D55	13	D60	18	D65
4	D51	9	D56	14	D61	19	D66
5	D52	10	D57	15	D62	20	D67

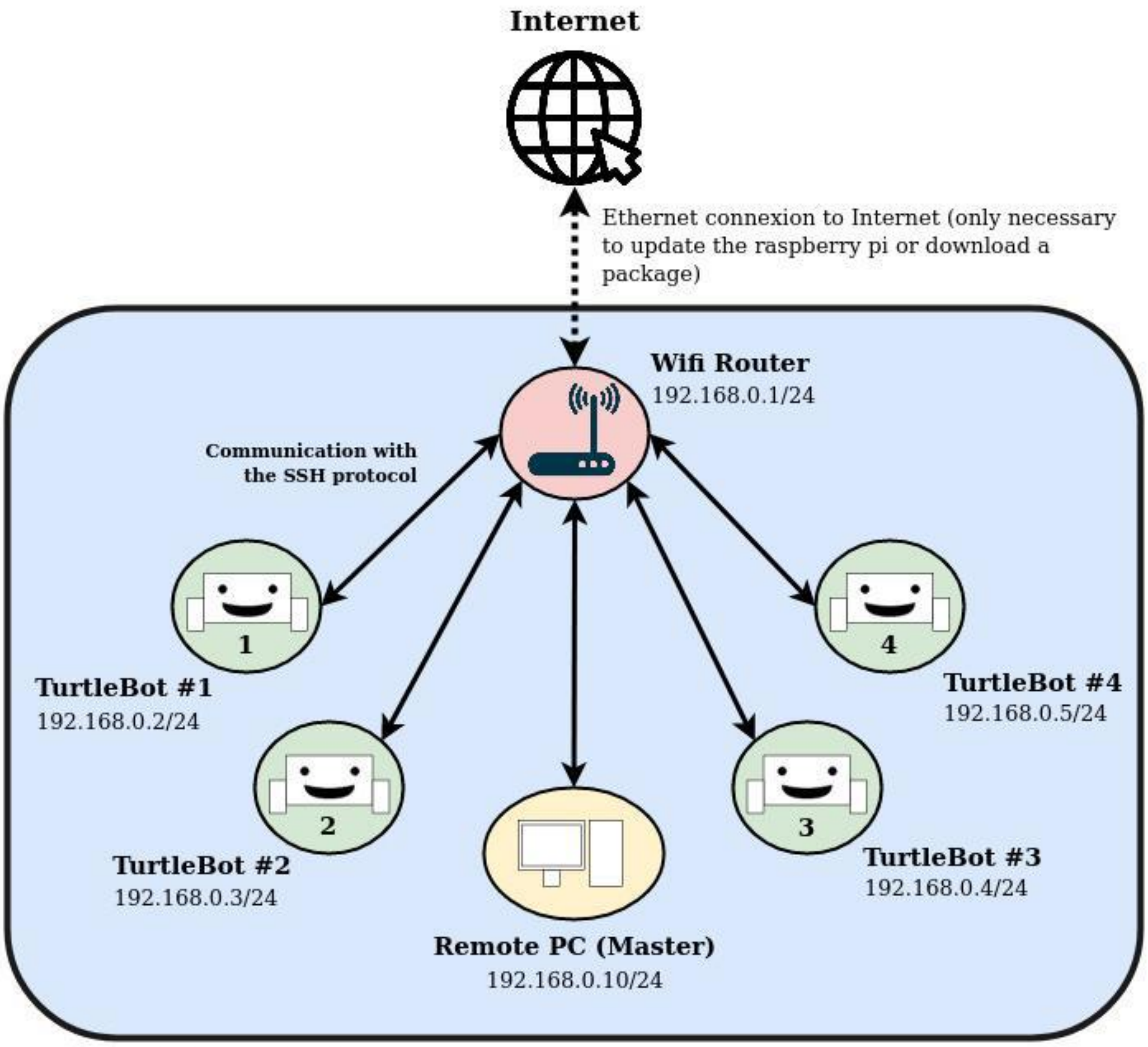
JTAG

1	3.3V	6	JTDO_SWO
2	JTMS_SWDIO	7	GND
3	GND	8	JTDI
4	JTCK_SWCLK	9	GND
5	GND	10	MCU_NRESET

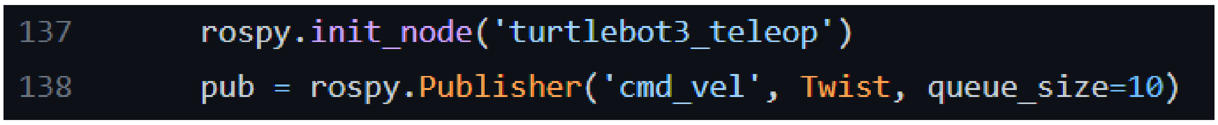
Appendix 10: Turtlebot3 Burger fleet configuration



Appendix 11: Representation of the network

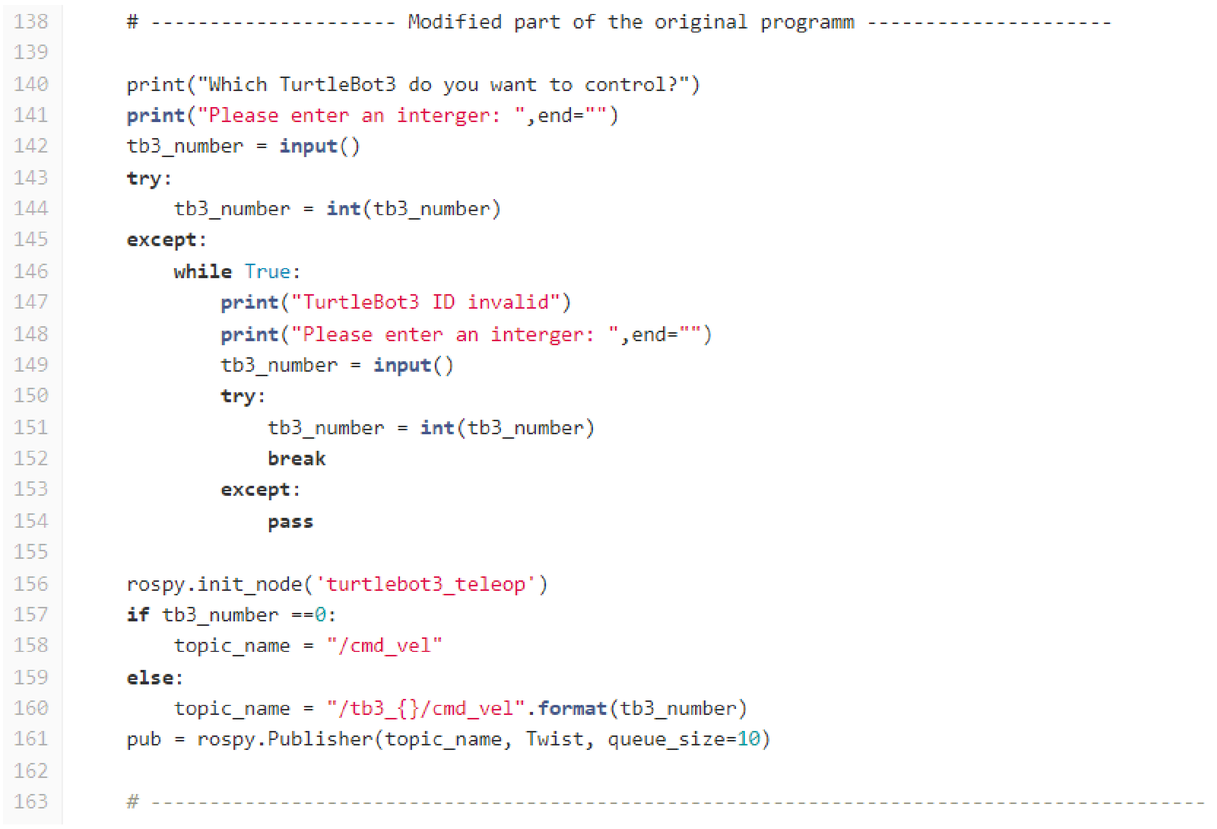


Appendix 12: Modified Turtlebot3 Burger teleop program



```
        rospy.init_node('turtlebot3_teleop')
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
```

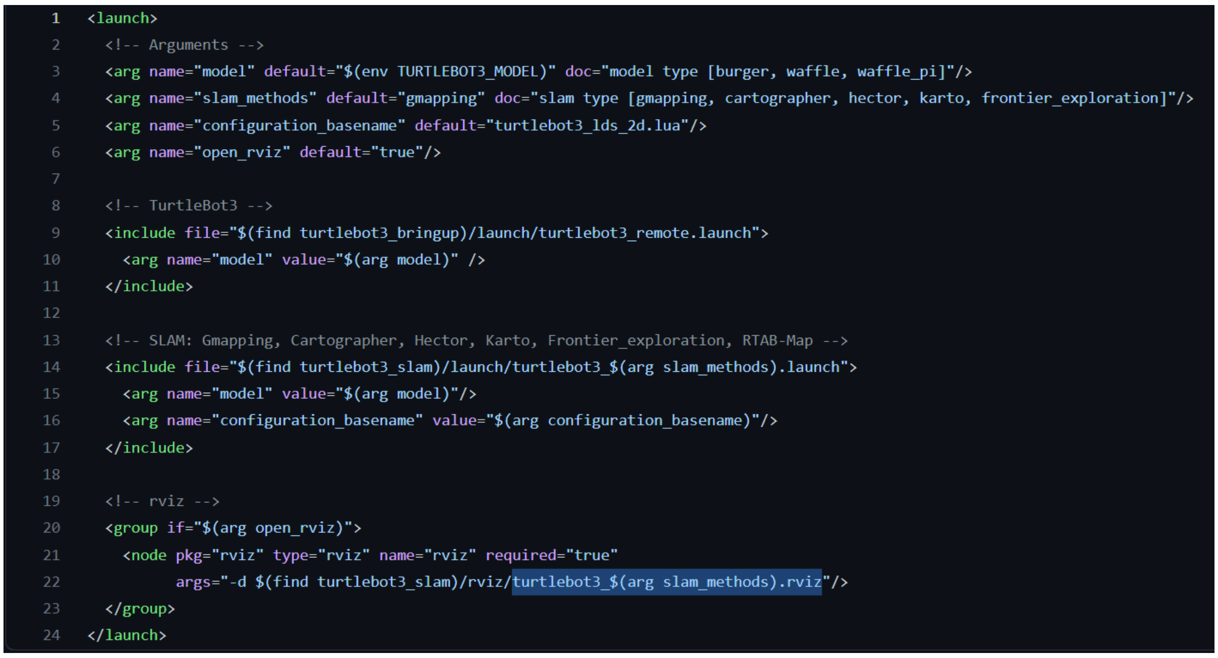
Old version



```
    # -------------------- Modified part of the original programm --------------------

    print("Which TurtleBot3 do you want to control?")
    print("Please enter an interger: ",end="")
    tb3_number = input()
    try:
        tb3_number = int(tb3_number)
    except:
        while True:
            print("TurtleBot3 ID invalid")
            print("Please enter an interger: ",end="")
            tb3_number = input()
            try:
                tb3_number = int(tb3_number)
                break
            except:
                pass

    rospy.init_node('turtlebot3_teleop')
    if tb3_number ==0:
        topic_name = "/cmd_vel"
    else:
        topic_name = "/tb3_{}/cmd_vel".format(tb3_number)
    pub = rospy.Publisher(topic_name, Twist, queue_size=10)

    # --------------------------------------------------------------------------------------
```

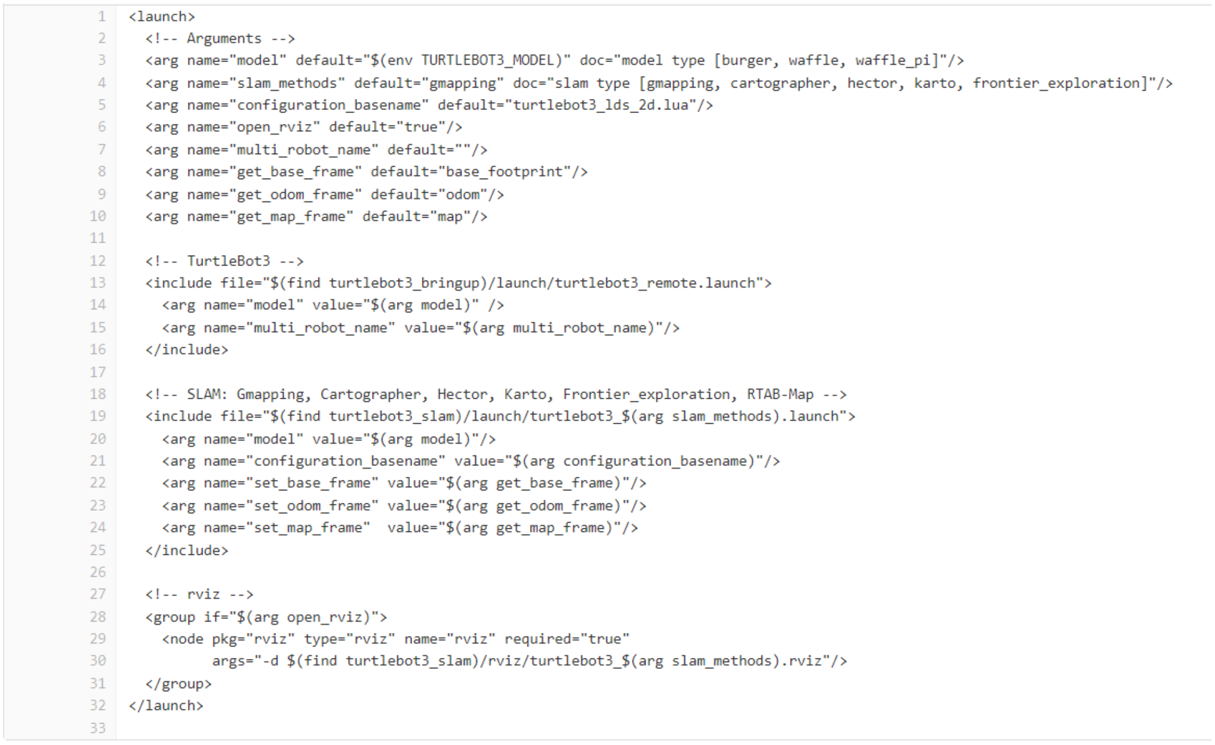
New version

Appendix 13: Modified Turtlebot3 Burger SLAM program



```xml
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="slam_methods" default="gmapping" doc="slam type [gmapping, cartographer, hector, karto, frontier_exploration]"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="open_rviz" default="true"/>

  <!-- TurtleBot3 -->
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)" />
  </include>

  <!-- SLAM: Gmapping, Cartographer, Hector, Karto, Frontier_exploration, RTAB-Map -->
  <include file="$(find turtlebot3_slam)/launch/turtlebot3_$(arg slam_methods).launch">
    <arg name="model" value="$(arg model)"/>
    <arg name="configuration_basename" value="$(arg configuration_basename)"/>
  </include>

  <!-- rviz -->
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find turtlebot3_slam)/rviz/turtlebot3_$(arg slam_methods).rviz"/>
  </group>
</launch>
```

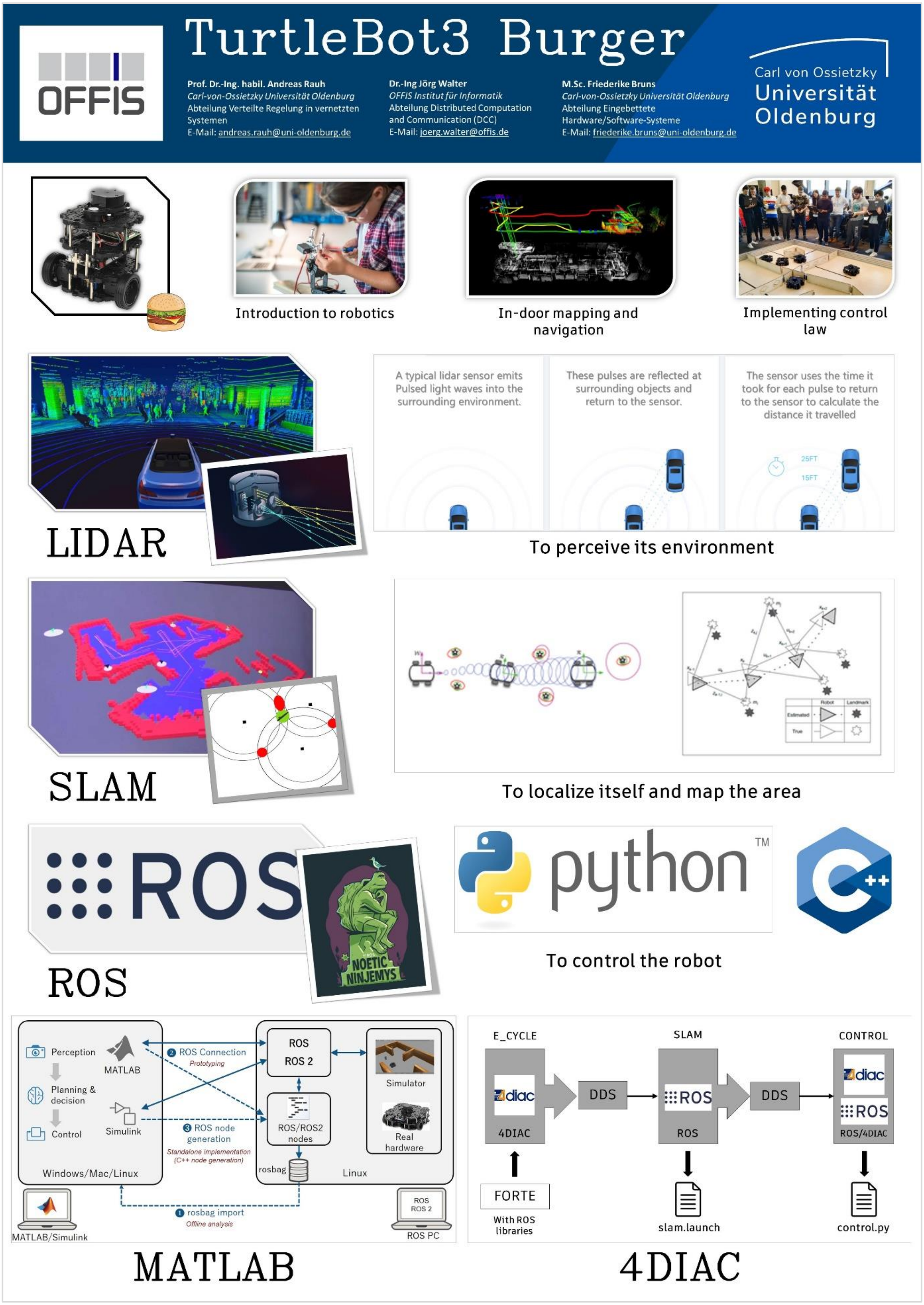
Old version



```xml
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="slam_methods" default="gmapping" doc="slam type [gmapping, cartographer, hector, karto, frontier_exploration]"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="open_rviz" default="true"/>
  <arg name="multi_robot_name" default=""/>
  <arg name="get_base_frame" default="base_footprint"/>
  <arg name="get_odom_frame" default="odom"/>
  <arg name="get_map_frame" default="map"/>

  <!-- TurtleBot3 -->
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)" />
    <arg name="multi_robot_name" value="$(arg multi_robot_name)"/>
  </include>

  <!-- SLAM: Gmapping, Cartographer, Hector, Karto, Frontier_exploration, RTAB-Map -->
  <include file="$(find turtlebot3_slam)/launch/turtlebot3_$(arg slam_methods).launch">
    <arg name="model" value="$(arg model)"/>
    <arg name="configuration_basename" value="$(arg configuration_basename)"/>
    <arg name="set_base_frame" value="$(arg get_base_frame)"/>
    <arg name="set_odom_frame" value="$(arg get_odom_frame)"/>
    <arg name="set_map_frame"  value="$(arg get_map_frame)"/>
  </include>

  <!-- rviz -->
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find turtlebot3_slam)/rviz/turtlebot3_$(arg slam_methods).rviz"/>
  </group>
</launch>
```

New version

Appendix 14: Long night of digitalization poster


TurtleBot3 Burger
OFFIS
Prof. Dr.-Ing. habil. Andreas Rauh
Carl-von-Ossietzky Universität Oldenburg
Abteilung Verteilte Regelung in vernetzten Systemen
E-Mail: andreas.rauh@uni-oldenburg.de
Dr.-Ing Jörg Walter
OFFIS Institut für Informatik
Abteilung Distributed Computation and Communication (DCC)
E-Mail: joerg.walter@offis.de
M.Sc. Friederike Bruns
Carl-von-Ossietzky Universität Oldenburg
Abteilung Eingebettete Hardware/Software-Systeme
E-Mail: friederike.bruns@uni-oldenburg.de
Carl von Ossietzky Universität Oldenburg
Introduction to robotics
In-door mapping and navigation
Implementing control law
LIDAR
A typical lidar sensor emits Pulsed light waves into the surrounding environment.
These pulses are reflected at surrounding objects and return to the sensor.
The sensor uses the time it took for each pulse to return to the sensor to calculate the distance it travelled
25FT
15FT
To perceive its environment
SLAM
To localize itself and map the area
ROS
NOETIC NINJEMYS
python™
To control the robot
Perception
MATLAB
Planning & decision
Control
Simulink
Windows/Mac/Linux
2 ROS Connection
Prototyping
3 ROS node generation
Standalone implementation (C++ node generation)
ROS
ROS 2
ROS/ROS2 nodes
Simulator
Real hardware
rosbag
Linux
1 rosbag import
Offline analysis
MATLAB/Simulink
ROS PC
MATLAB
E_CYCLE
SLAM
CONTROL
4diac
4DIAC
DDS
ROS
ROS/4DIAC
FORTE
With ROS libraries
slam.launch
control.py
4DIAC