Université Toulouse III – Paul Sabatier
UPSSITECH
118 Route de Narbonne
31062 Toulouse CEDEX 9

2nd year Robotic and Interactive Systems

Ostfalia, University of Applied Sciences
Faculty of Computer Science
Salzdahlumer Str. 46/48
38302 Wolfenbüttel, Germany

INTERNSHIP REPORT

Prototyping a mobile 3D-printer with a KUKA youBot robot

12 April 2021 – 16 July 2021

Presented and supported by

Clément TRUILLET

Internship tutor
Julien VANDERSTRAETEN

Internship supervisor
Reinhard GERNDT

Acknowledgement

First, I would like to thank Prof. Dr. Karger, president of the University of Ostfalia and Mr. Gutenschwager, dean of the faculty of computer science of Ostfalia for their welcome at Ostfalia university in Wolfenbüttel. I would also like to thank my internship supervisor and my internship tutor, Prof. Dr-Ing. Gerndt and Mr. Vanderstraeten for allowing me to carry out this internship and for having supervised me throughout the project.

Next, I would like to thank the people who made it possible for me to go to Wolfenbüttel. All the Erasmus offices of University Toulouse III Paul Sabatier, Arielle Noirot, the secretary of the SRI formation and Patrick Danès, the Erasmus coordinator of UPSSITECH.

Also, it would be a mistake to forget to thank Mr. Kelm who accompanied me brilliantly with all his knowledge available during the internship as well as Mrs. Bizien without whom we would have been quickly lost in the meanders of the German administration as well as many situations of everyday life very specific to the Germans.

Finally, I would like to thank Prof. Lerasle, head of the Robotic and Interactive Systems (SRI) course, Julien Pinquier, head of the second year SRI course, as well as the entire UPSSITECH pedagogical team for all the teaching they gave me, often with passion, and which enabled me to fully live this international experience.

I would also like to thank my parents for supporting me during this period and, of course, Raphaël Bizet without whom these three months wouldn't have been the same.

And finally, I would like to thank all the people I've met during this period for their energy and the advice they gave me.

Contents

Figure List

Annexes List

Glossary

In alphabetical order

CSV	Comma Separated Values is a text format representing a table with values separated by commas
LTS	Long Term Support designates a version of a software program that has support assured for a long time
OSR	The Open Source Robotics Foundation is an organisation whose objective is to support the development of software in the field of robotics
ROS	Robot Operating System is a collection of software frameworks for robot software development
SDF	Simulation Description Format is a format representing robot for the Gazebo simulator
SRI	Robotics and Interactive Systems
URDF	Unified Robot Description Format is a format representing robot for ROS
WSL2	Windows Subsystem Linux 2 is a Linux environment available under Windows 10
XML	Extensible Markup Language is a markup language

Introduction

As part of my Robotic and Interactive Systems training at the Paul Sabatier University of Toulouse, I had the opportunity to do an internship from April 12 to July 16, 2021 at the Institute of Distributed Systems of the Faculty of Computer Science in Ostfalia, University of Applied Sciences, in Wolfenbüttel (Lower Saxony, Germany).

Although it has a significant robotic connotation, this internship is part of a research context and aims to answer the question of how robotics can take its place in the field of construction and public works.

The objective of this internship was to carry out a proof of concept of a mobile robot capable of printing small but possibly very long 3D structures.

In order to do so, the internship will be divided into two distinct parts: a simulation of a chosen robot, the KUKA youBot, under Ubuntu, ROS 2 and Ignition Gazebo, and a realization with the same robot remotely controlled and commanded with ROS2.

After presenting the University of Ostfalia, the Faculty of Informatics and the Institute of Distributed Systems, we will focus on the background of this problematic through a state of the art and then describe the technologies used during this internship and its organisation as well as the chosen approaches to the subject, their developments, and results.

This document is not a technical note, but only presents the methods and working tools used without going into details of implementation.

The appendices at the end of this document illustrate this report and are referenced in many points.

I. Presentation of the Ostfalia university

The University of Applied Sciences(www.ostfalia.de), chaired by Prof. Dr. Karger (cf. Figure 1), has been operating under the name of „Ostfalia “since 2009 but the history of Ostfalia and its predecessor institutions dates to 1853 (Suderburg), 1905 (Braunschweig) and 1928 (Wolfenbüttel).

With about 13,000 students, four campuses (Salzgitter, Wolfenbüttel, Wolfsburg and Suderburg), twelve faculties and 21 departments, Ostfalia counts among the biggest universities of applied sciences in Lower Saxony.

The Faculty of Computer Science, chaired by Prof. Dr. Gutenschwager, based on the Wolfenbüttel campus is divided into 4 institutes covering different areas of computer science, the institute of information engineering, software engineering, media informatics and e-learning as well as distributed systems.

This institute for embedded systems, headed by Prof. Dr. Detlef and Prof. Dr-Ing. Gerndt and with a staff of about 20 people, covers the fields of robotics, modelling, embedded systems but also vehicle computing, circuit technology and data transfer.

The robotics group was founded in order to bundle activities in the field of robotics at Ostfalia and to promote joint research projects. It tackles questions about the use of robots in industry and in health and care sectors. The topic of robot safety and the use of humanoid robots is addressed as well.

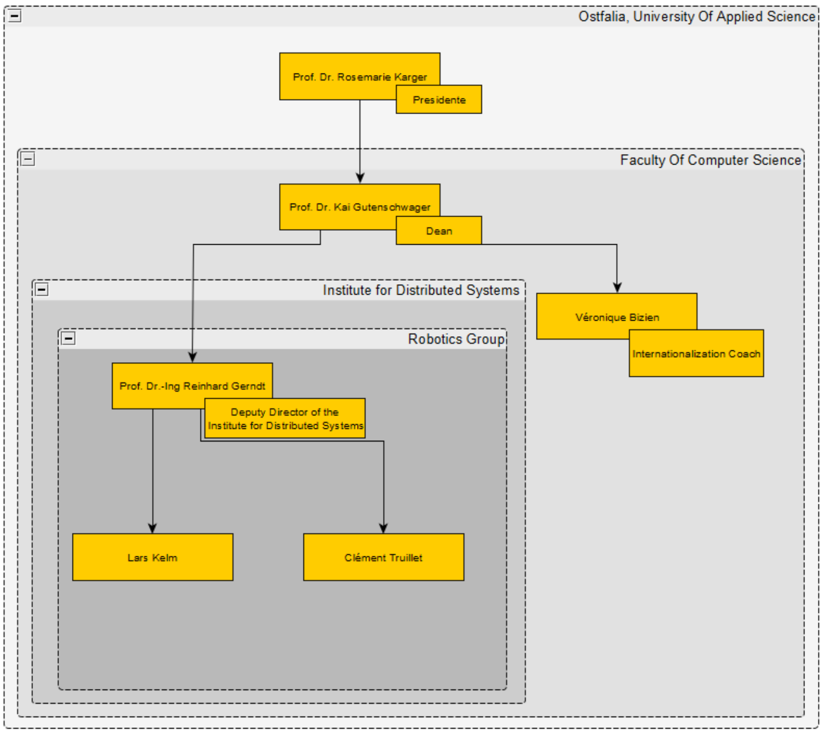

Ostfalia, University Of Applied Science
Prof. Dr. Rosemarie Karger
Presidente
Faculty Of Computer Science
Prof. Dr. Kai Gutenschwager
Dean
Institute for Distributed Systems
Robotics Group
Véronique Bizien
Internationalization Coach
Prof. Dr.-Ing Reinhard Gerndt
Deputy Director of the Institute for Distributed Systems
Lars Kelm
Clément Truillet

Figure 1 : Organisation chart

II. Presentation

As the Youbot's subject is quite complete and falls within a mainly research context, it is appropriate to first study its problematic through a state of the art before presenting the robot itself as well as the technologies used to finally describe the organisation of the work during the course.

II.1 State of the Art

If you are looking for a place to visit near Wolfenbüttel, it is quite common to go to Wolfsburg, at the Autostadt [1], a large museum on the theme of the car and, more precisely, of Volkswagen. If the car is at the centre of the visit, one can nevertheless admire how robotics is taking a major place in the construction of today's and tomorrow's cars, to the point of making the transport between the factory and the museum and the storage of new cars completely automatic.

Robotics, sometimes in combination with other fields such as artificial intelligence or image processing, is taking on an increasingly important role but yet, *un village gaulois résiste encore et toujours à l'envahisseur*[1]*:* construction and public works.

Even though the field of construction is far from being technology-free, it is still one in which the human element is very important, despite the significant risks that exist.

It's therefore legitimate to ask **how robotics can take its place in the field of construction and public works.**

Far from the idea of replacing humans in this field, this issue is based on another observation: humankind is looking more and more towards space and more specifically towards the colonisation of Mars. This planet, currently the only one known to be populated exclusively by robots, is still far from welcoming. While sending pre-built habitats by rocket is not yet a feasible solution, another idea could be to have these habitats built by autonomous robots on site [2].

To do this, the researchers who investigated this question turned to a tool that was very popular at the time, and still is today: the 3D printer.

This was the idea of Danish company COBOD, which launched its BOD robot [3] in 2017, a gigantic 3D printer that prints houses up to 14m long and 10m high [4] (cf. Figure 2).

[1] In French in the text. « *Un village peuplé d'irréductible Gaulois résiste encore et toujours à l'envahisseur* » is the introductory sentence of each volume of Asterix, a famous French comic book.

Mostly autonomous, this robot saves time with a printing speed of up to 1 m/s and reduces the number of workers required.

However, it requires on-site installation and people to add space for doors and windows, making it more suitable for construction on Earth (where it already meets India's housing standards) than on Mars.

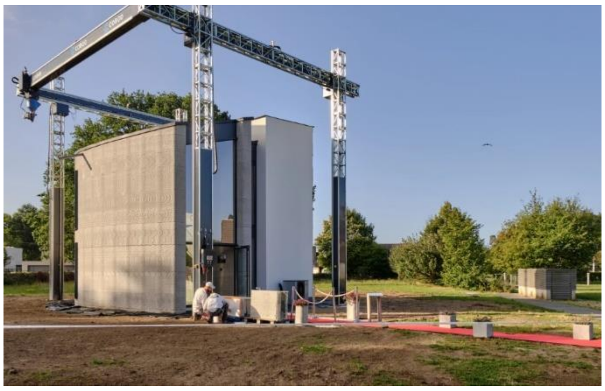


Figure 2 : The BOD robot

In 2014, the Institute for Advanced Architecture of Catalonia (IAAC) [5] took another approach: the Minibuilders.

The idea is to create three types of small robots, all of which have the task of 3D printing a large structure (in the experiment a large vase of 2m height) but all with specificities that a single robot cannot have alone.

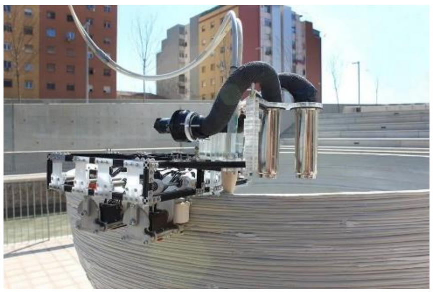


Figure 3 : The Grip Robot

In this project, the base of the structure (the first 20 centimetres) is made with a "*Foundation robot*". This closed shape is then used as a support for the "*Grip Robot*" (cf. Figure 3) which prints behind it and gradually rises to the desired height. These vertical wheels allow it to change the orientation of the printed wall. Finally, in order to reinforce the whole, the "*Vacuum robot*" is used, a robot armed with a vacuum pump allowing it to navigate without constraint on vertical walls and thus increase their thickness.

This project, which is older than COBOD, takes a completely different approach to the problem, since it is based on the collective work of fully autonomous robots. Although the printing time is much longer, it is nevertheless not limited in size by the structure of the robot itself.

Finally, in 2018, a team from Nanyang Technological University in Singapore proposed a new solution between Minibuilders and BOD: a mobile manipulator robot capable of 3D printing [6].

It works in a very different way to the previous solutions, since it relies on the strength of the BOD with its manipulator arm allowing it to reach precise points, but also on the main strength of the Minibuilders thanks to its ability to move thanks to the holonomic mobile base, i.e. being able to move in translation on the plane and in rotation according to the vector normal to the plane without constraint, allowing it to easily reach (when possible) any point of the structure to be printed.

However, this robot brings with it new problems that did not exist with the BOD and the Minibuilders, such as localisation and motion planning. If the BOD allows to reach any point of its working space from above precisely, it is not the case of this robot which must undergo permanent changes of reference points at the cost of a lower precision (nevertheless, the experiments carried out on this robot lead to a precision of 9.8 mm). Similarly, while the order in which elements are printed does not matter with a robot such as the BOD, a mobile robot (cf. Figure 4) must plan its schedule in advance to avoid creating inaccessible and unprinted portions of the structure.

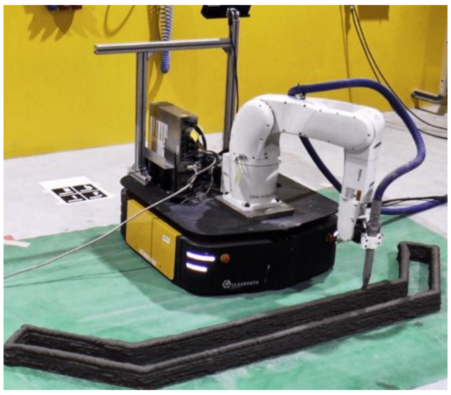


Figure 4 : The printing-while-moving robot

Finally, the use of a holonomic mobile base is certainly an advantage in moving the robot during printing, but it is not very effective on rough terrain such as that of Mars. With this idea in mind, another team from University College London (UCL) is proposing a new version of this robot with a holonomic mobile base adapted to rough terrain: the Mobile Agile Printer (MAP)[7].

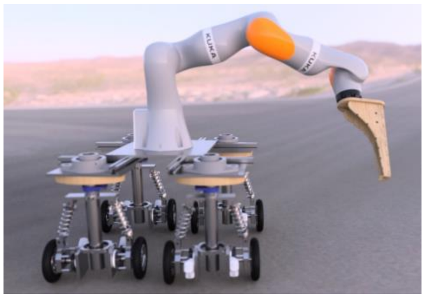


Figure 5 : MAP construction robot

If the base changes its shape completely to exchange its omnidirectional wheels for classic wheels (cf. Figure 5) capable of rotating along the z axis, the principle remains completely the same, giving the same result at the cost of additional inaccuracy (15mm accuracy).

In conclusion, the field of robotics for construction is very young. If some solutions, such as the BOD, are now used outside the laboratories, the vast majority remain at the centre of research, producing several ways of responding to the following problem in different cases.

It is in this context that Prof. Dr Ing. Gerndt supervised an internship [8] on this issue in 2019, showing the growing interest in mobile manipulator robots capable of 3D printing. While this internship mainly focused on the construction and implementation of such a robot with the base of a KUKA youBot, the objective of this one is to realize this 3D printing with this existing base.

As the context of the internship has been established and using ROS and the KUKA youBot is a constraint, it is necessary to study them in order to understand their strengths as well as their specificities.

II.2 Technologies used

In order to carry out the 3D printing simulation with the KUKA youBot, it is obvious that you must first understand the specificities of the youBot. Also, in the same way that the youBot is imposed for the realization of the internship, the use of the middleware ROS2 and the Gazebo simulator are also imposed. It is by its constraints that the KUKA youBot, ROS middleware and Gazebo are presented respectively in this part.

The KUKA youBot

The KUKA youBot (Figure 2), also known as Youbot, is an omni-directional mobile robot produced by the German company KUKA from 2013 to 2016.

If this robot can carry a payload of 20Kg, it can also be equipped with one or two KUKA manipulator arms with 5 degrees of freedom for a 90 minutes autonomy.

Figures 6 and 7 shows the KUKA youBot and its joints with rotations angles.

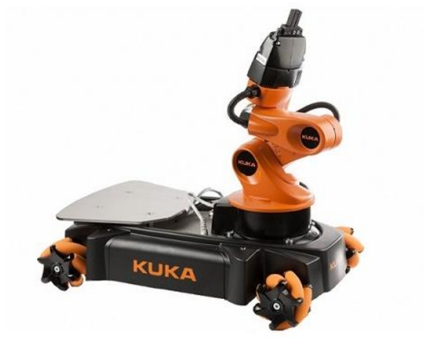


Figure 6 : The KUKA youBot [9]

It has been designed to work in industrial like environment to perform multiple industrial tasks in order to be a support for teaching, research and experimentation as shown by the numerous documentations produced to allow simulation and real time control of the robot with many software such as ROS (ROS Fuerte to ROS Indigo versions), MatLab Simulink, Webots, Modelica, Dymola 5, OpenRave and V-REP but also a strong emphasis on student projects or thesis related to this robot on the Youbot website.

During this internship, I use a Youbot equipped with a single manipulator arm that has undergone a few modifications following participation in competitions like RoboCup@Work by the Ostfalia robotics team (*WF Wolves RoboCup Team*) and by a bachelor work in 2019, the most notable of which are the disappearance of the manipulator arm's clamp for a 3D printing pen as well as the computer controlling the robot.

In the rest of the report, the Youbot describes the KUKA youBot platform with a KUKA arm when the platform and the KUKA arm will be respectively the Youbot platform and the Youbot arm.

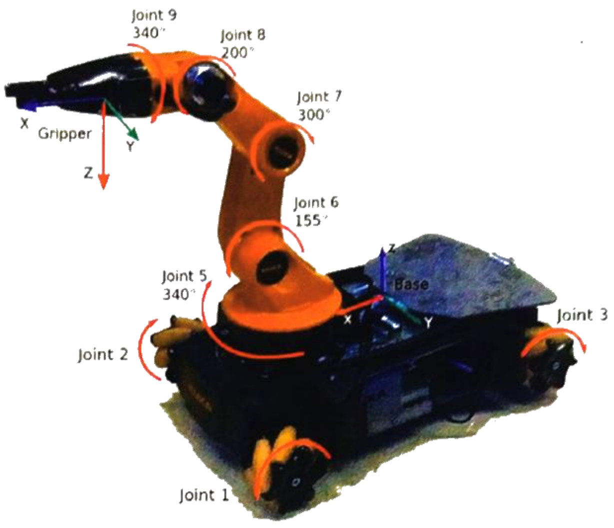


Figure 7 : The KUKA youBot with the positions of the joints [8], [9]

The Youbot arm

The Youbot arm is a 5-degree-of-freedom robotic arm consisting of 5 rotational links and a detachable end member, a gripper [9].

With a total weight of 6.3 kg and a maximum height of 65.5 cm (Figure 8), it can carry a load of up to 0.5 kg.

Due to its design and context (arm on a mobile platform), the arm has a working area of 0.513 m² (Figures 9 and 10) where each joint can move at a speed of 90°/s.

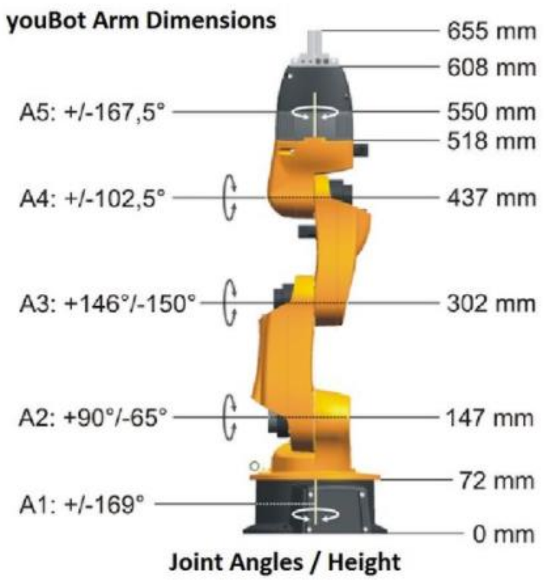


Figure 8 : Youbot Arm dimensions [9]

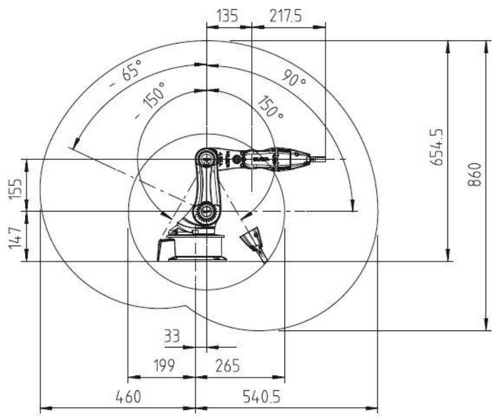


Figure 9 : Youbot Arm workspace (side view) [9]

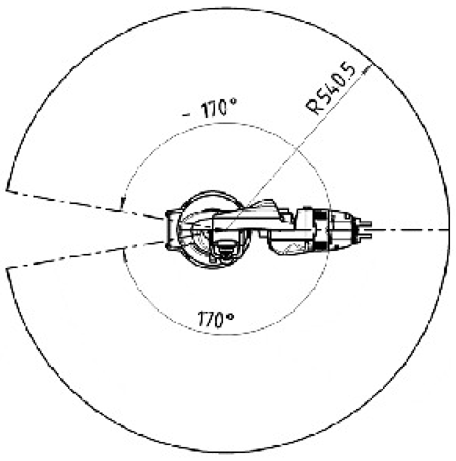


Figure 10 : Youbot Arm workspace (top view) [9]

The Youbot platform

The Youbot platform is composed of 4 omnidirectional wheels (cf. Figure 11) allowing it to move freely by translation along the x and y axes as well as by rotation around the z axis (figure 12).

When empty (without the KUKA arm), this 20 kg platform with a length of 58 cm by 38 cm and a height of 14 cm can carry a load of up to 20 kg at a maximum speed of 0.8 m/s [9], making it a perfect candidate for carrying moderately heavy loads in order to lighten the efforts of logistics employees.

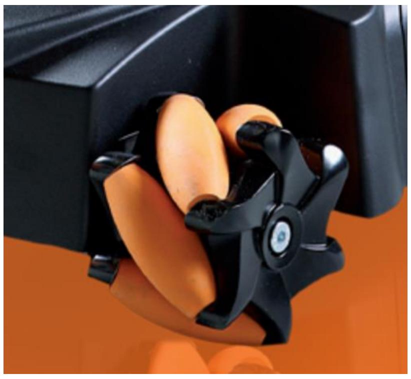


Figure 11 : Youbot wheel

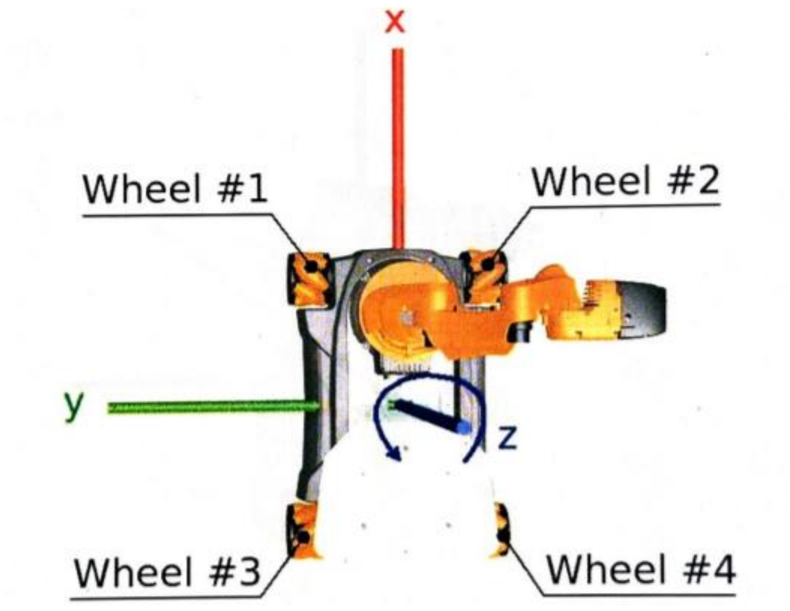


Figure 12 : Youbot platform

Robot Operating System (ROS) - ros.org

ROS (Robot Operating System) is a middleware for the operation of robots.

This middleware was first developed by *Willow Garage* in 2007, to perform operations on their robot named PR2 (Personal Robot 2), to become open source in 2012 and to be maintained until today by *Open Source Robotics Foundation* (OSR) and by the *Open Robotics* company (created by OSR) since 2018.

This framework has allowed ROS to have regular updates every year with a major release every two years (with 5 years support)

With the creation of Open Robotics, the OSR decided to stop the development of new versions of ROS to focus on the development of a new framework taking up the strengths of ROS (idea of starting from scratch).

Thus, ROS becomes ROS 1 and will have a last version for Ubuntu 20.04 LTS named *Noetic Ninjemys* (abbreviated Noetic). This new version of ROS is called ROS 2 and already has a first version (still under development) LTS for Ubuntu 20.04 LTS called *Foxy Fitzroy* (Foxy).

In summary, ROS is made up of easily usable packages, each with a different purpose but identical operation.

A **package** is composed of none, one or more executables written in C++ (or Python since the latest versions) describing the operation of **nodes**.

In the case of robots, there is usually a package called *ROBOT_description* which contains one or more URDF files and texture files (meshes).

The launch of an executable (which can launch one or more nodes) is done using the `roslaunch package_name executable_name` command (ROS1) or `ros2 launch package_name` executable_name (ROS2).

Note that to launch several executables communicating with each other under ROS1, you must first launch the core ROS via the `roscore` command

A node can communicate with other nodes via ports called topics (cf. Figure 13) in the form of messages with a predefined format (a topic can only transmit messages of the same type)[10].

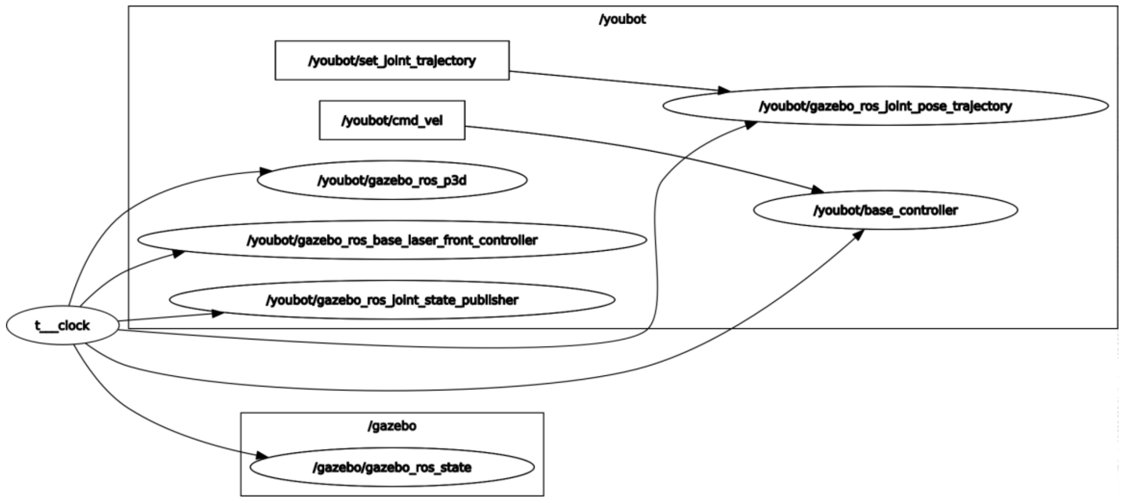

/youbot
/youbot/set_joint_trajectory
/youbot/gazebo_ros_joint_pose_trajectory
/youbot/cmd_vel
/youbot/gazebo_ros_p3d
/youbot/base_controller
/youbot/gazebo_ros_base_laser_front_controller
/youbot/gazebo_ros_joint_state_publisher
t__clock
/gazebo
/gazebo/gazebo_ros_state

Figure 13 : Example of ROS graph (the rectangles are the topics)

Unified Robot Description Format (URDF)

ROS is a middleware for robot development, it has its own robot description format called URDF [11]. It’s therefore a format used by all ROS software except Gazebo. Thus, for the control of a robot, more specifically the Youbot, we have a set of URDF files describing the Youbot platform, the Youbot arm and the combination of the both.

For this purpose, a URDF file is composed of link elements corresponding to the robot's bodies, its inertia and geometry as well a ‘s joints corresponding to the links between the bodies and describing its type (prismatic [2]or revolute[3]), its coefficients of friction and damping, its stops as well as its speed of movement in radians.

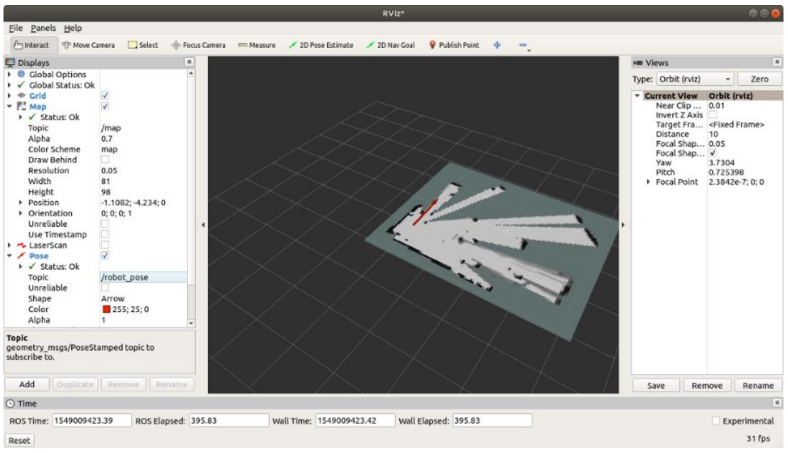
Its relative ease of writing and its rich library of models make it the most popular format in the ROS context.

Simulation Description Format (SDF)

Similar to the URDF format, the SDF file format is used to describe the behaviour and characteristics of robots but also of environments and lights [12].
Mainly used in the context of the Gazebo simulation, it also has a rich model library called fuel (https://app.ignitionrobotics.org/fuel/models).
Despite a very similar structure between this format and the URDF format, it is a much less popular format than the URDF and is mainly used only by Gazebo.

Rviz

Rviz (Ros Visualisation) is a 3D visualisation software for ROS that allows the visualisation of certain types of data passing through ROS topics [13] such as the position and orientation of a robot, the landmarks associated with these links or even the map generated by the robot in real time (cf. Figure 14).



Displays

Global Options	
Global Status: Ok	
Grid	✓
Map	✓
Status: Ok	
Topic	/map
Alpha	0.7
Color Scheme	map
Draw Behind	☐
Resolution	0.05
Width	81
Height	98
Position	-1.1082; -4.234; 0
Orientation	0; 0; 0; 1
Unreliable	☐
Use Timestamp	☐
LaserScan	☐
Pose	✓
Status: Ok	
Topic	/robot_pose
Unreliable	☐
Shape	Arrow
Color	🟥 255; 25; 0
Alpha	1

Topic
geometry_msgs/PoseStamped topic to subscribe to.

Views

Type: Orbit (rviz) Zero

Current View	Orbit (rviz)
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	<Fixed Frame>
Distance	10
Focal Shap...	0.05
Focal Shap...	✓
Yaw	3.7304
Pitch	0.725398
Focal Point	2.3842e-7; 0; 0

Time

ROS Time: 1549009423.39 ROS Elapsed: 395.83 Wall Time: 1549009423.42 Wall Elapsed: 395.83 ☐ Experimental

31 fps

Figure 14 : Rviz software

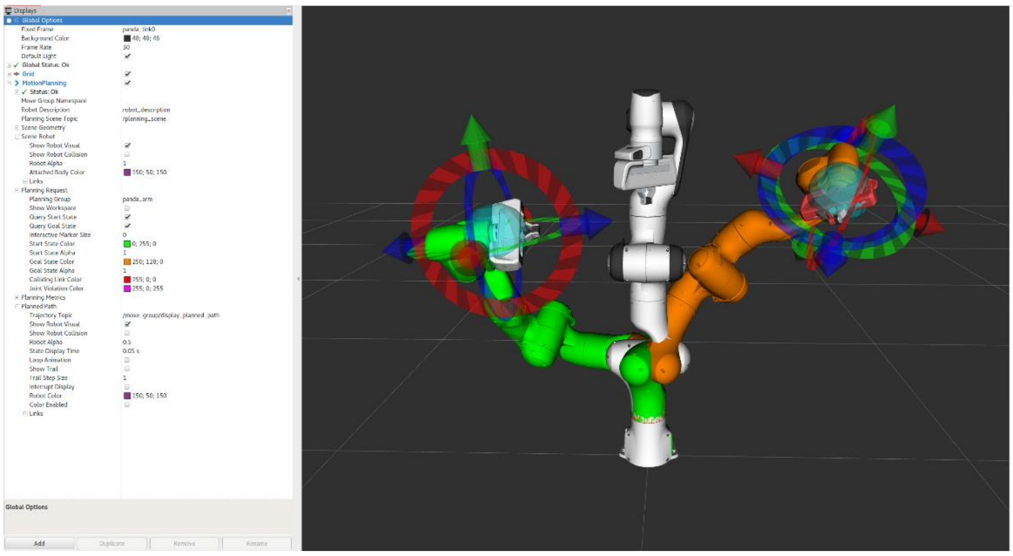
[2] Prismatic joint: a joint that allows translational movement along an axis.
[3] Revolute joint: joint allowing movement by rotation around an axis.

MoveIt

MoveIt is a set of ROS packages created in 2011 by *Willow Garage* and maintained today by the company *PickNik Robotics* [14]–[16] allowing the planning of robot movements and tasks in the ROS environment (Figure 15 shows the interface of MoveIt).

It is a very popular package used with nearly 150 robot models such as the KUKA arms, the Baxter robot (Rethink Robitics) or even the Tiago robot (PAL Robotics)

With the new versions of ROS2, the development of MoveIt follows the same strategy with the development of MoveIt2 running with Rviz2 (version of rviz for ROS2).



```
Displays
Global Options
    Fixed Frame                 panda_link0
    Background Color            48; 48; 48
    Frame Rate                  30
    Default Light               ✓
✓ Global Status: Ok
Grid                            ✓
MotionPlanning                  ✓
  ✓ Status: Ok
    Move Group Namespace
    Robot Description           robot_description
    Planning Scene Topic        /planning_scene
    Scene Geometry
    Scene Robot
      Show Robot Visual         ✓
      Show Robot Collision      ☐
      Robot Alpha               1
      Attached Body Color       150; 50; 150
      Links
    Planning Request
      Planning Group            panda_arm
      Show Workspace            ☐
      Query Start State         ✓
      Query Goal State          ✓
      Interactive Marker Size   0
      Start State Color         0; 255; 0
      Start State Alpha         1
      Goal State Color          250; 120; 0
      Goal State Alpha          1
      Colliding Link Color      255; 0; 0
      Joint Violation Color     255; 0; 255
    Planning Metrics
    Planned Path
      Trajectory Topic          /move_group/display_planned_path
      Show Robot Visual         ✓
      Show Robot Collision      ☐
      Robot Alpha               0.5
      State Display Time        0.05 s
      Loop Animation            ☐
      Show Trail                ☐
      Trail Step Size           1
      Interrupt Display         ☐
      Robot Color               150; 50; 150
      Color Enabled             ☐
      Links
Global Options
```

Figure 15 : MoveIt with the Panda Robot

Gazebo

Gazebo is a robot simulation software created in 2002 by the University of Southern California [17].

Maintained by the *Open Source Robotics* in 2012, this simulator is now fully integrated into the ROS environment while keeping its particularities (the SDF format is part of it).

Its development and its strong community make it one of the most powerful and most used simulators in the field of robotics.

With the appearance of ROS2, its community decided to create a new version of this simulator called Ignition Gazebo [18]. The old version is then called Gazebo Classic. In the following, Gazebo Ignition will be abbreviated to "*Ignition*" and Gazebo Classic to "*Gazebo*". (Interfaces of Gazebo and Ignition are shown on Figures 16 and 17).

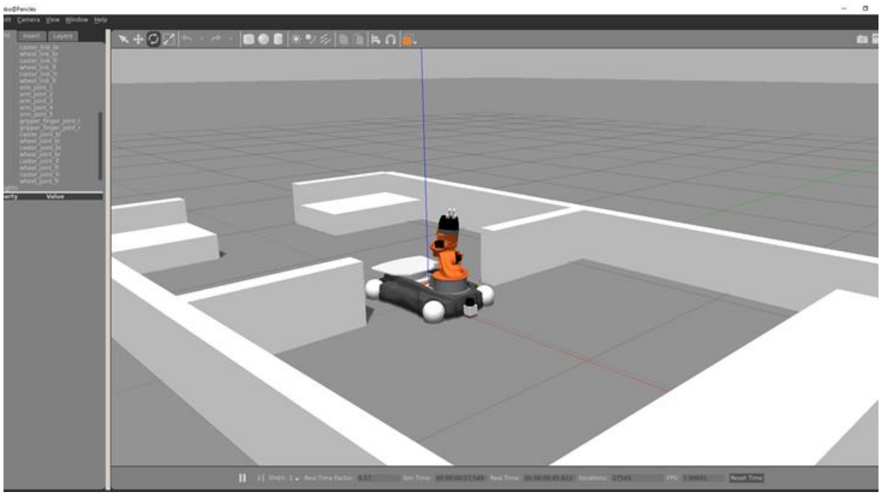


Figure 16 : Gazebo Software

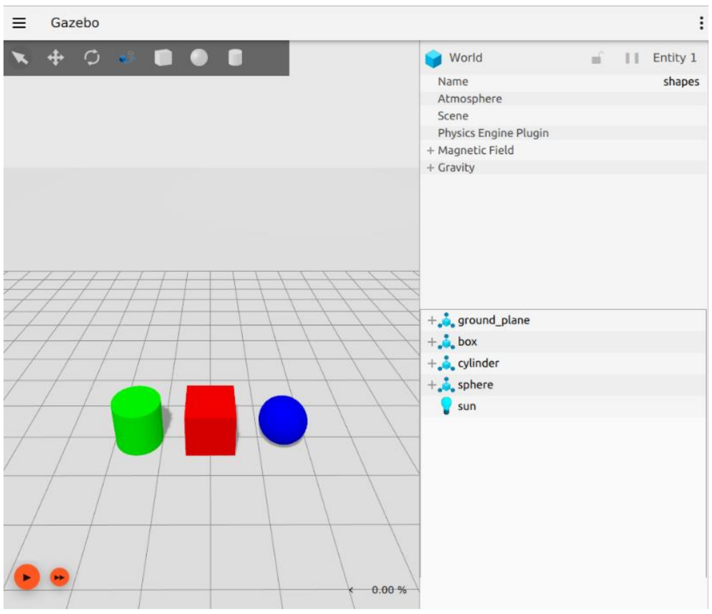


Figure 17 : Ignition Software

In conclusion, the technologies used in this course are popular tools with a strong community. If the learning and mastering of ROS and Gazebo is rather simple by the consequent number of existing tutorials, the state of development of these two tools has an impact on the organisation of the work as seen in the following.

II.3 Organisation

While I was lucky to be in Germany for the duration of the internship, COVID interfered a lot in the practical organisation.

Firstly, because France was classified as a high-risk zone at the beginning of April 2021, I had to carry out a two-week quarantine and, therefore, telework.

Then, the regulations in place at the university imposed a limit of one person in the laboratory where the Youbot was, so I had to book this room in advance, a room open to reservations only on Mondays, Wednesdays, and Fridays.

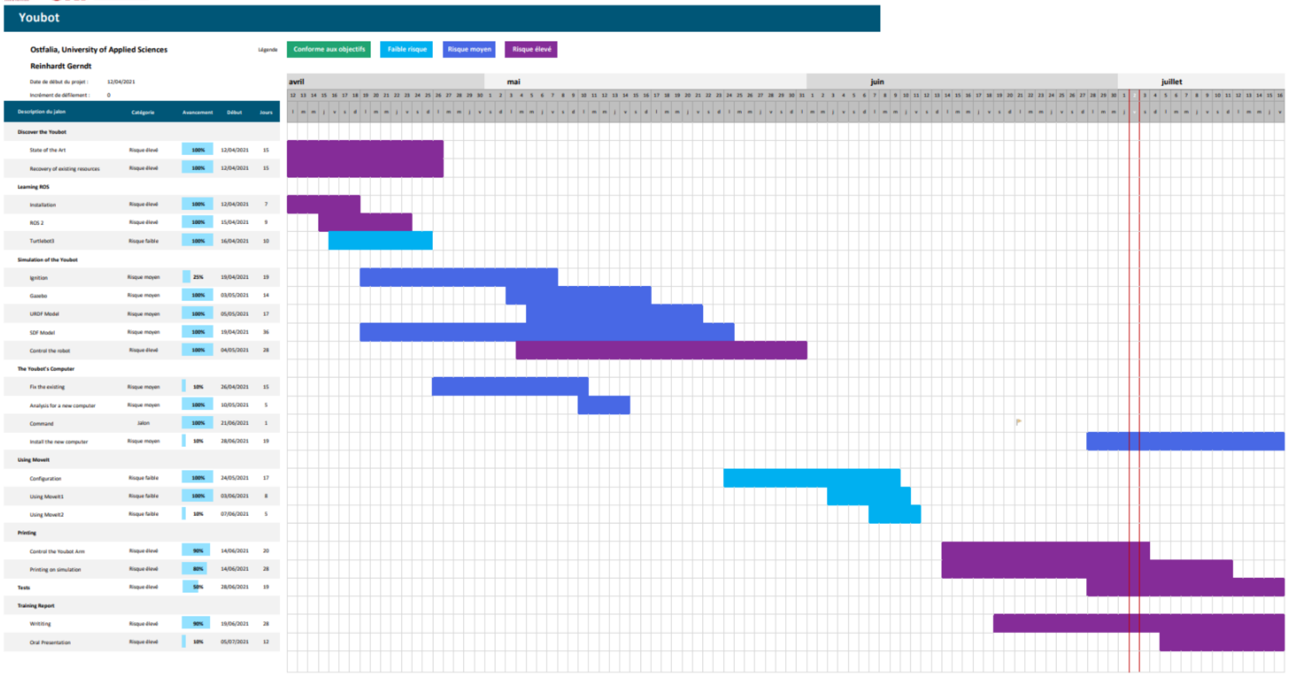
Finally, at Prof Gerndt's request, Raphaël Bizet and I were taking German lessons at B1 level on Tuesday mornings, making it difficult to work on the course during this time, although we made good progress in this language.

In sum, most of the internship was spent working from home. Nevertheless, my proximity to the laboratory (100 m) allowed me to be physically available easily in case of request (especially when it concerned the Youbot computer).

As the subject concerns a lot of recent technologies or still in development, we set up an agile method in order to be able to change course easily while remaining as close as possible to the desired objective at the end of the internship. For this purpose, I set up a Gantt chart evolving from the beginning to the end of the course in order to prioritise the tasks to be done each week according to their importance (cf. Figure 18).

Thus, each week was punctuated by a meeting (on Monday morning) in order to review the work done the previous week and to refocus, if necessary, the objectives of the week to come.

Even if it was not requested at the beginning, and as in all my previous projects, I had the opportunity to use **git** in order to keep a trace of my work during the internship as well as to leave an existing and solid base to my internship master for potential future internships around Youbot.



Youbot

Ostfalia, University of Applied Sciences

Reinhardt Gerndt

Date de début du projet : 12/04/2021

Incrément de défilement : 0

Description du jalon	Catégorie	Avancement	Début	Jours
Discover the Youbot				
State of the Art	Risque élevé	100%	12/04/2021	15
Recovery of existing resources	Risque élevé	100%	12/04/2021	15
Learning ROS				
Installation	Risque élevé	100%	12/04/2021	7
ROS 2	Risque élevé	100%	15/04/2021	9
Turtlebot3	Risque faible	100%	16/04/2021	10
Simulation of the Youbot				
Ignition	Risque moyen	25%	19/04/2021	19
Gazebo	Risque moyen	100%	03/05/2021	14
URDF Model	Risque moyen	100%	05/05/2021	17
SDF Model	Risque moyen	100%	19/04/2021	36
Control the robot	Risque élevé	100%	04/05/2021	28
The Youbot's Computer				
Fix the existing	Risque moyen	10%	26/04/2021	15
Analysis for a new computer	Risque moyen	100%	10/05/2021	5
Command	Jalon	100%	21/06/2021	1
Install the new computer	Risque moyen	10%	28/06/2021	19
Using MoveIt				
Configuration	Risque faible	100%	24/05/2021	17
Using MoveIt1	Risque faible	100%	03/06/2021	8
Using MoveIt2	Risque faible	10%	07/06/2021	5
Printing				
Control the Youbot Arm	Risque élevé	90%	14/06/2021	20
Printing on simulation	Risque élevé	80%	14/06/2021	28
Tests	Risque élevé	50%	28/06/2021	19
Training Report				
Writting	Risque élevé	90%	19/06/2021	28
Oral Presentation	Risque élevé	10%	05/07/2021	12

Figure 18 : Gantt Diagram

To conclude, this internship subject is part of a young context since the first mobile 3D printer robots date from 2019. As the Youbot was not designed for this use, KUKA's desire to make it available for research and education allows its realisation with several free tools such as ROS and Gazebo.

Because the field is not yet well developed and there is still a lot to discover, the agile method is used in the implementation of the project.

III. Implementation of the project

The subject of the course is multi-faceted and has been broken down into five identifiable and distinct tasks.

Firstly, we will focus on the **simulation of the Youbot** robot under ROS 2, Ignition as well as on **the Youbot's computer**. Secondly, we will make **the robot move** in simulation as well as physically and finally we will carry out a **3D printing** in simulation and in the real world.

Lastly, the last part will be a part of **tests and comparisons** between the two models.

However, before the realization of these tasks, a state-of-the-art part (presented previously in this document) takes place as well as a task of handling and discovery of ROS2.

As this last task did not bring any added value, either because of its short duration (less than two weeks) or because of its interest (realization of the tutorials proposed on the ROS and Gazebo Ignition websites), it was chosen to omit it in this report.

III.1 Simulation of the Youbot

Ignition Gazebo

Fortunately, when creating the Ignition model library, fuel, Open Robotics added several models including the Youbot [19] (cf. Figure 19). Thus, there is no need to write the SDF model (3D model of the robot for Ignition and Gazebo) from scratch since the only existing model is a URDF model and therefore not compatible for Ignition.

The next step is then to connect the robot to ROS 2 in order to make it move but also to retrieve the robot's sensor data in real time (value of the joints, position of the robot in the environment and data from the laser sensor positioned at the base of the robot).

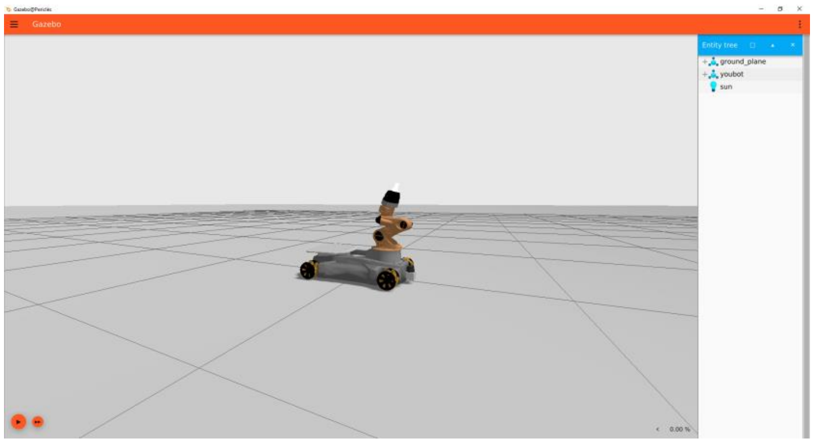


Figure 19 : The Youbot in the Ignition environment

During the implementation stage of the plugin to make the Youbot platform move, it appears that Ignition only provides about ten plugins, only one of which is designed to make a vehicle move. This plugin, named *DiffDrive*, is then fully adapted for a robot like a car or a turtlebot, i.e., non-honomous, and not for moving a holonomic robot such as the Youbot.

A new plugin must be written for this type of movement.

While Ignition is still in the early stages of development, there are not many guides to creating plugins, and they are often aimed at people who are already advanced in this type of activity.

After a research on the feasibility of writing the plugin and the impact on the course, it is decided to pass under Gazebo, by its seniority, already has this plugin as well as more attainable guides for the continuation of the development.

Gazebo

Ignition being an advanced version of Gazebo, there is no problem to migrate the SDF model from Youbot to Gazebo. We can then integrate the **Ros Planar Move** plugin in order to control the position of the base but also **Ros Joint Pose Trajectory** allowing to control the arm in position, speed, or acceleration. The operation of these plugins concerns more the movement of the robot than its simulation, they will be described more fully in this section.

As the Youbot has a laser sensor at its base by default, the **Ros Base Laser Front Controller** (cf. Figure 20) plugin can be used to detect obstacles in front of the robot at an angle of 180°. This plugin communicates via the topic */youbot/gazebo_ros_base_laser_front_controller* which can be displayed on Rviz.

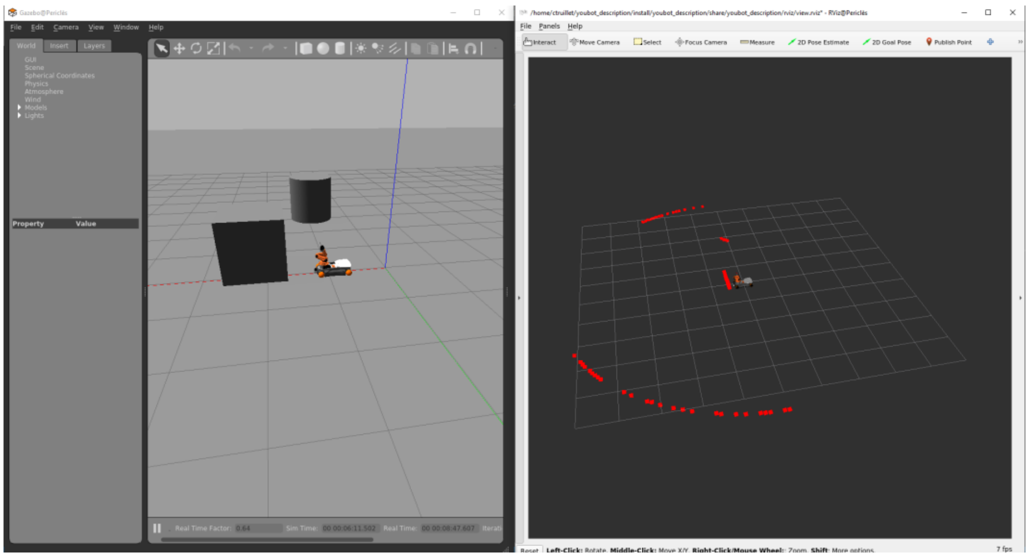


Figure 20 : Obstacle detection using Ros Base Laser Front Controller

Finally, and to save computation time, a virtual sensor is placed at the end of the printing nozzle via the **Ros P3D** plugin. This sensor returns its position in the Gazebo environment reference frame and saves the calculation and use of an inverse geometric model.

If setting up the simulation is essential for the implementation of the subject, the configuration of the Youbot's computer is also essential for the realisation of the 3D printing in the real world.

III.2 The Youbot's computer

From the previous work on the robot, computers remained (cf. Figure 21). Unfortunately, due to the time gap between this internship and the last work on the Youbot, these computers had major flaws: the first one didn't work and the second one kept rebooting.

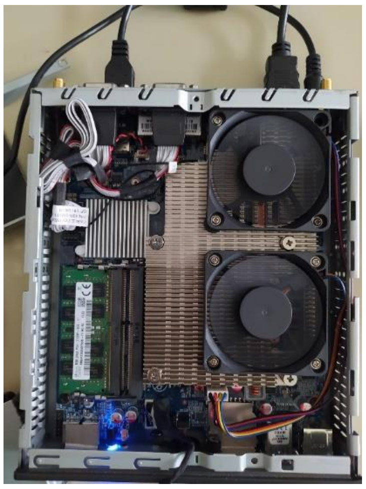
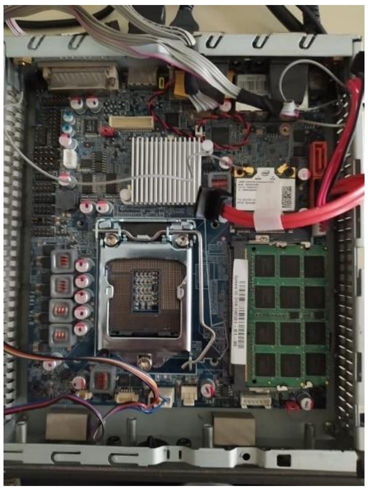



Figure 21 : Youbot's computers

After a few unsuccessful attempts to repair the computer, the decision was made to order a new computer. So, with the help of Mr. Kelm and the indications given by the company KUKA on the minimum configurations required for the Youbot computer, a new computer was ordered and received three weeks before the end of the course. This computer is an *XPC slim Barebone DS10U3* whose configurations are given in figure 22. Particular attention is paid to the dimensions of this computer since it must be positioned in a small space in the structure of the Youbot (cf. Figure 23).

	Youbot computer requirement	**XPC slim Barebone DS10U3**
Dimensions	20 x 16.5 x 3.95 cm	20 x 16.5 x 3.95 cm
CPU	Intel Atom D510 Dual Core 1.66 GHz	Intel Core i3-8145U Dual Core 2.1 GHz
RAM	2 GB	8 GB
Hard Drive	32 GB SSD	32 GB SSD
Ports	6xUSB, 1xVGA, 2xLAN	8xUSB, 1xVGA, 1xHDMI, 1xDisplayPort, 2xLAN

Figure 22 : Comparison between the minimum requirements for the Youbot computer and the chosen computer

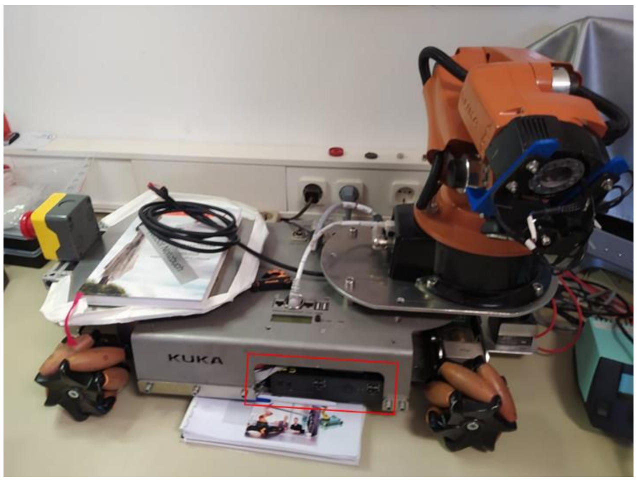


Figure 23 : In red, storage space on the Youbot computer

Once the simulation and the Youbot's computer are set up, we can finally focus on the task of moving the robot through the printing environment. This task is made even more important by the complexity of controlling the Youbot arm in sync with its platform.

III.3 Move the robot

MoveIt

As mentioned in the presentation of the technologies used, MoveIt is the most popular and complete set of ROS packages for controlling robots under ROS1. Although the development of MoveIt2, the version of MoveIt for ROS2, is more advanced, it is not yet complete. Indeed, the *Setup Assistant package* allowing the configuration of a new robot for MoveIt is not yet available under ROS2.

The idea is then to configure the modified Youbot (i.e., without the gripper but with the pen to print) under ROS1 Noetic (cf. Figure 24) to try to port this configuration to ROS2 or to link MoveIt to ROS 2 by using a ROS1 - ROS2 bridge.

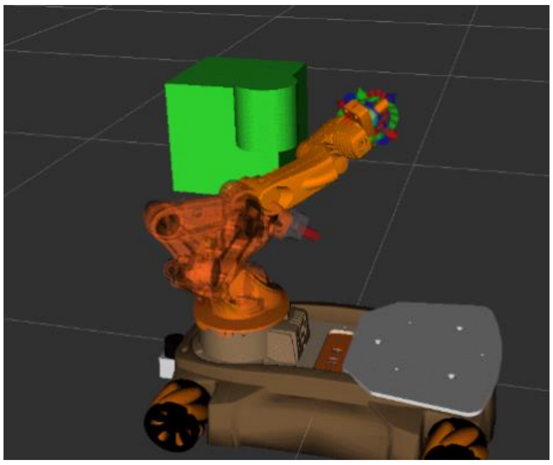


Figure 24 : Youbot control with MoveIt

If the configuration and control of the Youbot is very satisfactory under ROS1, the link to ROS2 is much less so due to the unconventionality of the thing. In view of the time needed to link MoveIt to ROS2 and its real usefulness in the development of the proof of concept, the choice was made to do without MoveIt to control the robot using the Ros Joint State Publisher plugins (arm control) and Ros Planar Move (base control).

This control of the robot is therefore more home-made and less complete. The first version of MoveIt2 was to be released on June 30, 2021 (a release that did not take place on that date), so the idea was to replace these plugins with MoveIt2 from that date.

Fake MoveIt

In order to carry out a 3D-printing, a ROS package named **fake_moveit** is developed, which decomposes into two executables.

The first one, **controlArm**, allows to position the end of the Youbot's arm at a printing position about 50 cm in front of the Youbot's base. Due to the structure of the Youbot not allowing a very large working space (5 x 5 cm square around the printing position), the time remaining to produce a correct simulation and my own limited knowledge about motion planning of a robotic arm on a mobile base (even if motion planning courses for mobile robots and robotic arms are held in the second year, this course is held in the third year according to the SRI program), it is decided to fix the position of the manipulator arm throughout the simulation (cf. Figure 25).

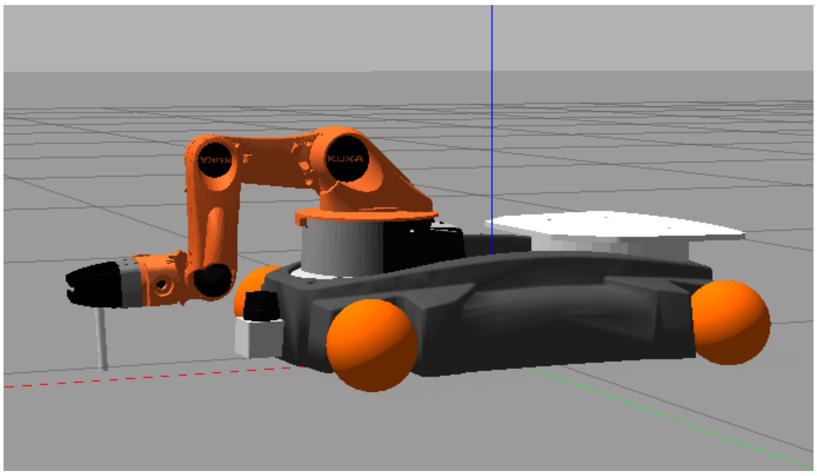


Figure 25 : Youbot in printing position

Thus, we only need to control the Youbot platform with the **controlBase** executable.

This executable takes as input a CSV file describing the 3D shape to be realised in order to generate a series of Twist type messages published on the topic */youbot/cmd_vel* thus controlling the Youbot's platform in linear velocity along the x and y axes and in angular velocity along the z axis.

It may be naively assumed that once the move has been managed, the rest of the project is a formality. However, 3D printing is far from being a formality as detailed below.

III.4 3D Printing

If the objective of the simulation is to display in real time the 3D printing produced by the Youbot, Gazebo is not designed for that. Indeed, the dynamic display of points, lines or shapes is not possible in Gazebo, so 3D models corresponding to the 3D printing must be added progressively.

For that, we have a service named */spawn_entity* provided by Gazebo allowing us to display at a given position in the world a 3D model described by a SDF file. If we display a small cube (1 mm side) at the position of the printing nozzle (position provided by the Ros P3D plugin) throughout the printing process, we quickly realise that the simulation is becoming slower and slower and that the gap, as shown in figure 26, between each cube is important (due to the delay between the emission of the new nozzle position).

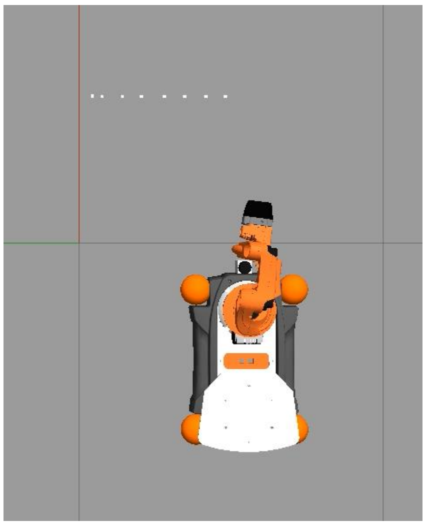


Figure 26 : First prototype of printing

Also, there is a delay between the display of the last printed point and the real position of the printing nozzle due to the performance of my computer and the obligatory delay of Gazebo when calling the */spawn_entity* service.

A second solution, more advanced, is, instead of displaying a multitude of cubes, to display rectangular shapes connecting two points emitted by the Ros P3D plugin (cf. Figure 27) . This solution is then very satisfactory since it produces a continuous line describing the 3D printing of the Youbot.

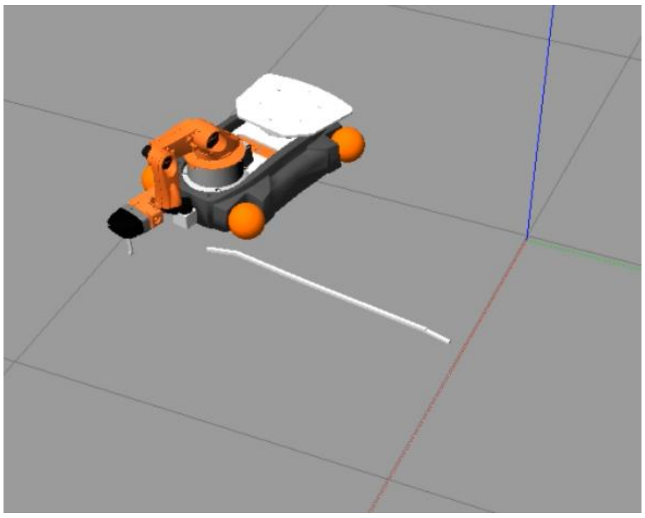


Figure 27 : Simulation of printing with rectangular shapes

Last, and as in all projects, it is essential to carry out tests in order to validate the development and progress of the project. Although done and crucial for the project, we will not present the different unit tests because of their lack of interest.

III.5 Tests

Mainly because of the delivery time of the Youbot computer, it is not possible to make comparisons between the 3D printing simulation and the real-world printing. However, tests are carried out to ensure that the simulation is working properly.

These tests are divided into two categories: the printing of a shape and the printing of a wall, i.e., a line at various heights.

Printing a shape

When printing a shape, the Youbot is simply asked to print a predefined continuous line (dotted red line in Figure 28). We then see differences between the actual print and the expected print on the changes of direction of the line.

If these accuracy errors are due to the delay between each transmission of the nozzle head position, one can try to reduce these errors by decreasing the speed of the robot in the simulation. This comes up against a limitation of the Gazebo simulator which, with its delay of a few seconds between the request and the display of 3D printing "blocks", reaches the limit of its queue more quickly and then produces holes in the 3D printing.

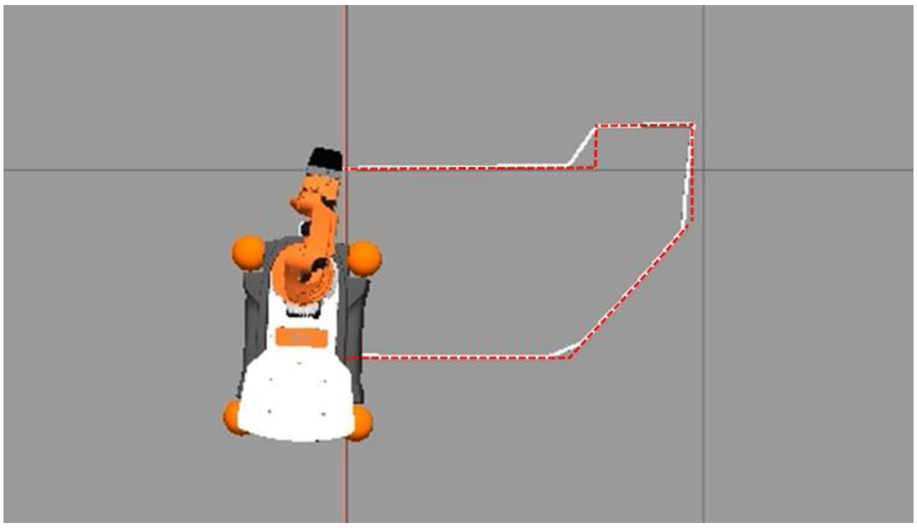


Figure 28 : Printing of a shape by the Youbot, in dotted red the expected shape

Also, a clear negative point of this simulation is that the robot only moves on the plane and never rotates. For this reason, a new prototype is produced which, at each change of direction, performs a rotation in order to remain always on the external side of the form to be printed. As the rotation axis of the Youbot is positioned in the middle of its platform, we calculate the elementary displacement (see Equation 1) necessary to move this rotation axis to the position of the printing nozzle.

$$dy = -\sin(d\theta)\, d$$
$$dx = d(\cos(d\theta) - 1)$$

Equation 1 : Theoretical elementary movement of the Youbot

One of the major problems is that the simulation time of Gazebo is different from the real time (1s may be equivalent to 0.6 to 0.8s in Gazebo). This simulation time, not fixed, depends on the computer (in the context of the training course, my personal computer) but also on the use of the processor. However, if the transmission of the commands is done at constant time intervals (real time), the reception is done at constant intervals according to the simulation time (thus non-constant time intervals according to the real time). This leads to significant precision errors (see Figure 29), particularly in the rotation of the robot around a point (error between the rotation performed and the expected rotation).

Several solutions to this problem have already been explored: one could calculate the ratio between the simulation time and the real time in order to store the commands at constant time according to the simulation time. Unfortunately, Gazebo does not provide the simulation time and therefore makes this solution impossible to implement.

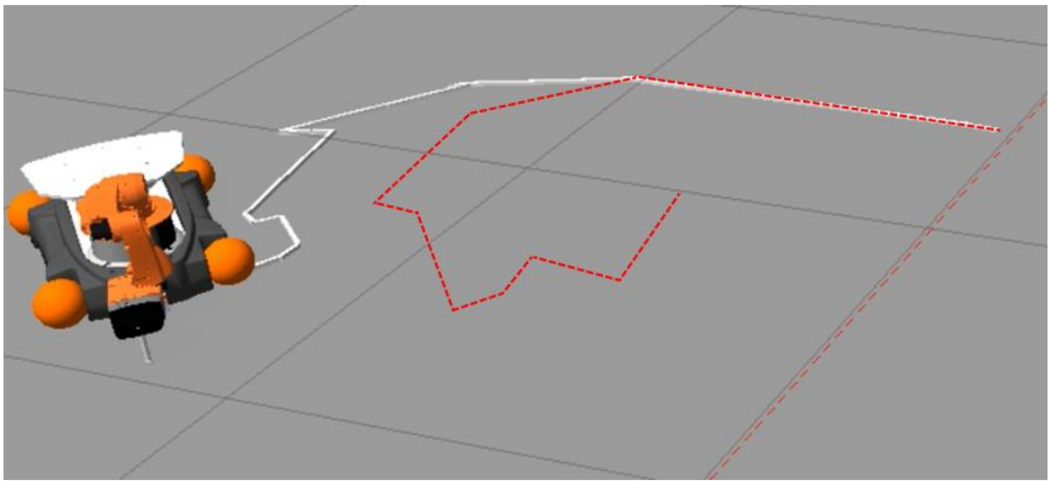


Figure 29 : Printing a shape with rotation of the Youbot, in dotted red the expected shape

Also, one can use a more powerful computer under Linux. One would then be almost certain that the simulation time is equal to the real time. Although this is the simplest solution, it cannot be implemented for reasons of available hardware.

Finally, the last solution is to use MoveIt2 which is much more accurate (since developed by a company since 2011) and which simulates the robot with Rviz. Gazebo is just a mirror of the simulation.

Printing a wall

The second test carried out is that of multi-layer printing. For this, the simple objective is to build a wall by moving back and forth.

The calculation of the linear trajectories being light, we do not notice the impact of the difference between real time and simulation time.

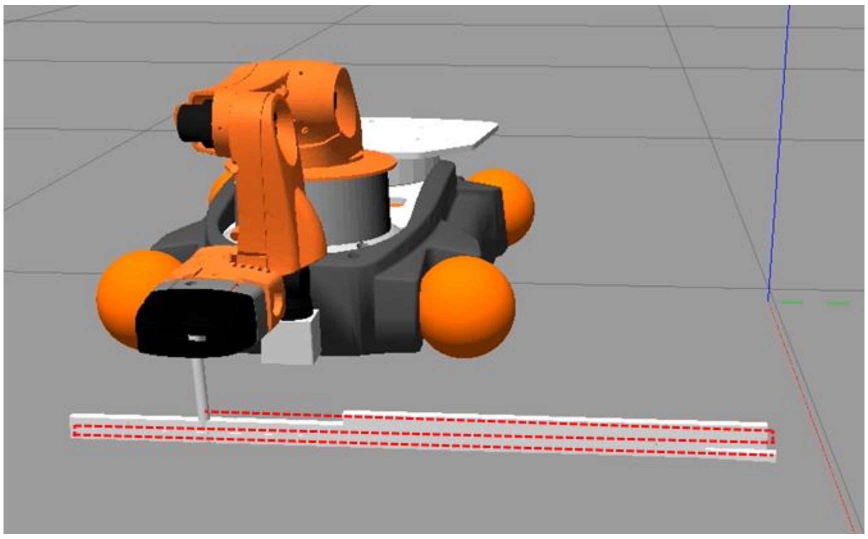


Figure 30 : Multilayer printing by the Youbot, in dotted red the expected print

In conclusion, the current implementation of the project uses many ROS packages developed exclusively for the Youbot. This results in simplifications in the control of the Youbot have an impact on the operation of the 3D printing. The interest of this realization lies in its simplifications which allow it to dive into the heart of many subjects such as the management of parallel processes, management invisible if the progress of the development of MoveIt2 had allowed its use.

Conclusion

As Prof. Gerndt reminded me many times, the subject of this course is the realisation of a proof of concept and not the realisation of a finished product. Thus, even if the realization is important, the reliability and the accuracy of the 3D printing of the Youbot are not of major importance, we are looking to know if it is possible to 3D print large structures with a Youbot controlled with ROS2 and simulated under Gazebo.

As presented during the state of the art, different types of robots already exist with the objective of 3D printing, some of which, like the printing-while-moving robot (controlled with ROS1), have an operation quite like the Youbot. It is therefore quite easy to conclude that, yes, given the existing and functioning Youbot as presented in this report, it is possible to 3D print large structures in the Gazebo environment and controlled with ROS2.

However, some conclusions can be drawn from this proof of concept.

First, due to its particularities, the Youbot does not allow for a very large working area to print accurately around the platform. As a robot intended to do mainly pick and place, printing is only possible in front of the robot, unlike robots such as the MAP or the printing-while-moving robot. Although it is quite possible to do 3D printing with the Youbot, it is much heavier (26 Kg) than the printing-while-moving robot can be and has very few resources for its control via ROS2. Its use then requires a lot of work of porting its resources from ROS1 to ROS2 (as done during the internship) but also of its ROS packages and drivers which have not been updated.

Finally, this proof of concept shows the importance of controlling the robot in a closed loop at the simulation level (the impact of a command on the robot is checked and corrected). Indeed, if the simulation of this proof of concept is in an open loop (the robot is controlled without correcting the control) due to the delay that this implies. It is nevertheless possible to minimise this delay by writing Gazebo plugins directly or by using the Ignition simulator when its development is completed.

In conclusion, ROS2 and the Gazebo simulator are suitable tools for the simulation of a Youbot 3D printer of large structures. Thanks to their strong communities, their regular updates, and the open-source nature of these two tools, it is quite possible for a beginner to quickly master them to produce more and more advanced prototypes as well as for someone who has mastered them to produce a finished and reliable product.

It's with continuous pleasure that I was able, during this internship, to put into practice many of my courses given at UPSSITECH. Whether it is subjects I expected such as ROS, Python as well as the whole field of robotics (modelling and motion planning mainly) but also unexpected subjects such as those related to parallelism but also to critical systems showing me another vision of these subjects and their applications different from the one seen at UPSSITECH.

In the same way, this internship allowed me to affirm even more the skills I had such as the use of Git but also of WSL2 in the use of ROS 2.

Despite my knowledge of ROS on arrival, I was directly immersed in its version 2 and all its strengths and weaknesses due to its stage of development. If sometimes the hours of research around such and such specificities did not bear fruit and led the subject of the internship to be slightly deviated, it is my persistence and the moral support of Prof. Gerndt that allowed me to always push my research further and to come back to Toulouse with a solid base of competence in ROS2.

Being immersed in the German world, the practice of my French language was quite minor, leaving room for German sometimes but mostly for English. Although I had started taking German courses since secondary school and had only passed the TOEIC test one month before my departure, I arrived in Wolfenbüttel with some doubts about my ability to practice these languages in everyday life. I abandoned these doubts when I returned home, confident but always with the will to progress.

Finally, I had the chance to work on a subject that brings new problems and where the solution is not established. This work has allowed me to continue to confirm my interest in the world of research, an interest already established during my last internship last year on the realization of another proof of concept.

Bibliography

[1] ‘Die Autostadt in Wolfsburg’. https://www.autostadt.de/en/start (accessed Jun. 23, 2021).

[2] The B1M, *Why This 3D-Printed House Will Change The World*, (2020). Accessed: Jun. 15, 2021. [Online Video]. Available: https://www.youtube.com/watch?v=XHSYEH133HA

[3] COBOD, ‘The BOD - The first 3D printed building in Europe’, *COBOD*. https://cobod.com/the-bod/ (accessed Jun. 23, 2021).

[4] CNET, *Giant 3D-printer builds a TWO-STORY house in one piece 🏠*, (2020). Accessed: Jun. 15, 2021. [Online Video]. Available: https://www.youtube.com/watch?v=lZh8E6zZdzk

[5] IAAC, ‘Minibuilders’. http://robots.iaac.net/# (accessed Jun. 15, 2021).

[6] M. E. Tiryaki, X. Zhang, and Q.-C. Pham, ‘Printing-while-moving: a new paradigm for large-scale robotic 3D Printing’, in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019, pp. 2286–2291. doi: 10.1109/IROS40897.2019.8967524.

[7] J. Sustarevas, D. Butters, M. Hammid, G. Dwyer, R. Stuart-Smith, and V. M. Pawar, ‘MAP - A Mobile Agile Printer Robot for on-site Construction’, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018, pp. 2441–2448. doi: 10.1109/IROS.2018.8593815.

[8] Mun Seng Phoon, ‘Mobile Robot for Additive Manufacturing’, Ostfalia, University of Applied Sciences, Faculty of Mechanical Engineering, Institute of Mechatronics, Ostfalia, University of Applied Sciences, Faculty of Mechanical Engineering, Institute of Mechatronics, Bachelorarbeit 70468762, Feb. 2019.

[9] ‘youBot Store’. http://www.youbot-store.com/developers/documentation (accessed May 27, 2021).

[10] ‘ROS 2 Documentation’, *ROS 2 Documentation*. doc.ros.org/en

[11] ‘urdf - ROS Wiki’. http://wiki.ros.org/urdf (accessed Jun. 10, 2021).

[12] ‘SDFormat Home’. http://sdformat.org/ (accessed May 27, 2021).

[13] ‘rviz - ROS Wiki’. http://wiki.ros.org/rviz (accessed Jun. 10, 2021).

[14] ‘MoveIt Motion Planning Framework’. https://moveit.ros.org/ (accessed Jun. 10, 2021).

[15] T. Coleman David, Ioan A. Sucan, Sachin Chitta, and Nikolaus Correl, ‘Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study’, May 2014, doi: 10.6092/JOSER_2014_05_01_P3.

[16] M. Gorner, R. Haschke, H. Ritter, and J. Zhang, ‘MoveIt! Task Constructor for Task-Level Motion Planning’, in *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada, May 2019, pp. 190–196. doi: 10.1109/ICRA.2019.8793898.

[17] ‘Gazebo’. http://gazebosim.org/ (accessed May 27, 2021).

[18] ‘Ignition’. https://ignitionrobotics.org/home (accessed May 27, 2021).

[19] ‘Kuka YouBot’. https://fuel.ignitionrobotics.org/1.0/OpenRobotics/models/Kuka YouBot (accessed Jun. 24, 2021).

Annexes

All annexes are available on the repository used throughout the course at the following address: https://github.com/ctruillet/youbot

Annexe 1 : Light version of the SDF file of the Youbot

```
<?xml version="1.0"?>
<sdf version='1.6'>
	<model name='youbot'>
		<!-- BASE -->
		<link name='base_footprint' >
			<sensor name='base_laser_front' type='gpu_ray'>
				<ray>
					<scan>
						<horizontal>
							<min_angle>-1.57</min_angle>
							<max_angle>1.57</max_angle>
						</horizontal>
					</scan>
					<range>
						<min>0.05</min>
						<max>5.6</max>
					</range>
					<noise>
						<type>gaussian</type>
						<mean>0.00</mean>
					</noise>
				</ray>
				<plugin name='gazebo_ros_base_laser_front_controller'
				        filename='libgazebo_ros_ray_sensor.so'>
					<ros>
						<namespace>/youbot</namespace>
						<argument>--ros-args --remap
~/out:=scan</argument>
					</ros>

	<output_type>sensor_msgs/LaserScan</output_type>
				</plugin>
			</sensor>
		</link>

		<!-- ARM -->
		<link name='arm_link_1' />
		<link name='arm_link_2' />
		<link name='arm_link_3' />
		<link name='arm_link_4' />
		<link name="pen" />
		<link name="printer" />
		<joint name='arm_joint_1' type='revolute' />
		<joint name='arm_joint_2' type='revolute' />
		<joint name='arm_joint_3' type='revolute' />
		<joint name='arm_joint_4' type='revolute' />
		<joint name='arm_joint_5' type='revolute' />
		<joint name='pen_joint' type='fixed' />
		<joint name="printer_joint" type="fixed" />
```

```
             <!-- PLATFORM-->
             <link name='caster_link_bl' />
             <link name='wheel_link_bl' />
             <link name='caster_link_br' />
             <link name='wheel_link_br' />
             <link name='caster_link_fl' />
             <link name='wheel_link_fl' />
             <link name='caster_link_fr' />
             <link name='wheel_link_fr' />
             <joint name='caster_joint_bl' type='revolute' />
             <joint name='wheel_joint_bl' type='revolute' />
             <joint name='caster_joint_br' type='revolute' />
             <joint name='wheel_joint_br' type='revolute' />
             <joint name='caster_joint_fl' type='revolute' />
             <joint name='wheel_joint_fl' type='revolute' />
             <joint name='caster_joint_fr' type='revolute' />
             <joint name='wheel_joint_fr' type='revolute' />

             <!-- PLUGINS -->
             <plugin name="gazebo_ros_joint_pose_trajectory"
filename="libgazebo_ros_joint_pose_trajectory.so" />
             <plugin name='base_controller'
filename='libgazebo_ros_planar_move.so' >
                    <robot_base_frame>base_footprint</robot_base_frame>
             </plugin>
             <plugin name="gazebo_ros_p3d" filename="libgazebo_ros_p3d.so">
                    <body_name>printer</body_name>
      </plugin>
      </model>
</sdf>
```

Annexe 2 : ControlArm Node

```
import subprocess
import rclpy
from rclpy.node import Node

from std_msgs.msg import String
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist, PoseWithCovariance, Pose, Point,
Quaternion
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
from math import inf



class ControlArm(Node):
    def __init__(self):
        super().__init__('youbot_fake_moveit')
        self.get_logger().info("subscribe")

        self.publisherJointTrajectory_ = self.create_publisher(JointTrajectory,
'/youbot/set_joint_trajectory', 10)
        self.timer = self.create_timer(0.5, self.setArmPosition)


    def setArmPosition(self):
        msg = JointTrajectory()
        msg.header.frame_id="world";
        msg.joint_names = ['youbot::arm_joint_1', 'youbot::arm_joint_2',
'youbot::arm_joint_3', 'youbot::arm_joint_4', 'youbot::arm_joint_5']
        msg.points = [JointTrajectoryPoint(positions=[3.1, 3.0, -1.0, 0.3,
2.9])]

        self.publisherJointTrajectory_.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    minimal_subscriber = ControlArm()
    rclpy.spin(minimal_subscriber)
    minimal_subscriber.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Annexe 3 : ControlBase Node

```
import subprocess
import rclpy
import csv
import copy
import numpy as np
import time
from rclpy.node import Node
from std_msgs.msg import String
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist
from std_msgs import Int32
from math import atan2, sqrt, pi, cos, sin

windRose = {
    "W" : pi/2.0,
    "NW" : pi/4.0,
    "N" : 0.0,
    "NE" : -pi/4.0,
    "E" : -pi/2.0,
    "SE" : -3*pi/4.0,
    "S" : pi,
    "SW" : 3*pi/4.0,
}

class ControlBase(Node):

    def __init__(self):
        super().__init__('youbot_fake_moveit')
        self.controlPointFile = 
"./src/youbot_fake_moveit/resource/print_Shape.csv"
        self.position = np.array([0.0,0.0,0.0])
        self.angle = 0.0
        self.angleYoubot = 0.0
        self.endMove = True

        self.publisherControlBaseTrajectory = self.create_publisher(Twist, 
'/youbot/cmd_vel', 10)
        self.publisherControlLevel = self.create_publisher(Int32, 
'/youbot/cmd_level', 10)

        self.controlList = []
        self.readFile()

        self.move(self.controlList)

    def readFile(self):
        with open(self.controlPointFile, mode='r') as csv_file:
            csv_reader = csv.DictReader(csv_file)
            line_count = 0
            for row in csv_reader:
                if line_count == 0:
                    #print(f'Column names are {", ".join(row)}')
                    line_count += 1

                direction = row["direction"]
                distance = row["distance"]
                level = row["level"]
                print(f' | {direction}\t{distance}\t{level}')

                self.genCommand(row["direction"], float(row["distance"]),
int(row["level"]))

                line_count += 1

    def genCommand(self, direction, distance, level):
        angle = windRose.get(direction,self.angle)
        mvt = np.array([round(cos(angle) * distance,2),
                        round(sin(angle) * distance,2),
                        level - self.position[2]])

        # LEVEL
        if(abs(mvt[2]) > 0):
            msg = Twist()
            msg.linear.z = mvt[2]

        # ANGLE
        # ToDo
        mvtAngle = abs(angle - self.angleYoubot + pi/2.0)%pi

        for i in range (int( round(16*abs(mvtAngle)/pi,0))):
            msg = Twist()
            msg.angular.z = mvtAngle/abs(mvtAngle) * pi/16.0
            self.controlList.append(msg)
        self.angleYoubot = angle
        self.controlList.append(self.genStopMsg())

        if distance <= 0.0:
            return

        # MOVEMENT
        for i in range (10*int(round(distance/0.1,0))):
            msg = Twist()

            if (round(cos(angle),3) == 0.0):
                msg.linear.x = 0.0
            else :
                msg.linear.x = round(cos(angle),3)/abs(round(cos(angle),3)) *
0.1

            if (round(sin(angle),3) == 0.0):
                msg.linear.y = 0.0
            else :
                msg.linear.y = round(sin(angle),3)/abs(round(sin(angle),3)) *
0.1

            msg.angular.z = 0.0

            self.controlList.append(msg)
        self.controlList.append(self.genStopMsg())
        # print(f' >> {round(cos(angle),2)} - {round(sin(angle),2)}')
        # print(f' >> {direction}\t{angle}\t{distance}\t{level}')

    def move(self, controlList):
        print(" BEGINNING OF THE MOVEMENT ")
        for msg in controlList:
            # print(msg)
            if(msg.linear.z != 0):
                self.changeLevel(msg.linear.z)
                pass
            else:
                self.publisherControlBaseTrajectory.publish(msg)
            time.sleep(0.2)


        self.publisherControlBaseTrajectory.publish(self.genStopMsg())

        print(" END OF THE MOVEMENT ")

    def changeLevel(self, level):
        self.publisherControlLevel.publish(level)

    def genStopMsg(self):
        msg = Twist()
        msg.linear.x = 0.0
        msg.linear.y = 0.0
        msg.linear.z = 0.0
        msg.angular.z = 0.0
        return msg

def main(args=None):
    rclpy.init(args=args)
    minimal_subscriber = ControlBase()
    rclpy.spin(minimal_subscriber)
    minimal_subscriber.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

Annexe 4 : Printer Node

```
import subprocess
import rclpy
import time
import copy
from rclpy.node import Node
import numpy as np

from std_msgs.msg import String
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist, PoseWithCovariance, Pose, Point,
Quaternion
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
from std_msgs import Int32

from math import inf, sqrt, cos, sin, atan2

xOffset = 0.484
yOffset = -0.042
zOffset = 0.0


class Printer(Node):
	def __init__(self):
		super().__init__('youbot_print')
		self.get_logger().info("subscribe")
		self.subscription = self.create_subscription(
			Odometry,
			'/youbot/p3d',
			self.listener_callback_P3D,
			10)
		self.subscription = self.create_subscription(
			Int32,
			'/youbot/cmd_level',
			self.listener_callback_CMD_LEVEL,
			10)
		self.subscription  # prevent unused variable warning
		self.printCurently = False
		self.controlPoints = []
		self.zlevel = 0.0
		self.lastPosition = np.array([0.0, 0.0, 0.0])

	def listener_callback_P3D(self, msg):
		P = np.array([round(msg.pose.pose.position.x,
3),round(msg.pose.pose.position.y, 3),round(msg.pose.pose.position.z, 3)])

		if (np.linalg.norm(P - self.lastPosition) > 0.05):
			lastP = copy.copy(self.lastPosition)
			self.lastPosition = copy.copy(P)

	self.get_logger().info("==================================================
================================")
			self.get_logger().info("GET POINT")
			self.get_logger().info("\t {P:{x:%.3f, y:%.3f, z:%.3f}}" %
(P[0], P[1], P[2]) )

	self.get_logger().info("==================================================
================================")
			self.waitUntil(self.printCurently)
```

```
                    self.printing(lastP, P)

       def listener_callback_CMD_LEVEL(self, msg):
             self.zlevel = msg

       def printing(self, A, B):
             self.printCurently = True
             xA = A[0]
             yA = A[1]
             zA = A[2]
             xB = B[0]
             yB = B[1]
             zB = B[2]

             distance = np.linalg.norm(B - A)
             if (distance <= 0.05 or distance >= 100):
                    self.printCurently = False
                    return

             angle = atan2(yB - yA, xB - xA)

             #self.get_logger().info(" >>> Distance : %f" % distance)

       self.get_logger().info("==================================================
===============================")
             self.get_logger().info("PRINTING")
             self.get_logger().info("\t {A:{x:%.3f, y:%.3f, z:%.3f}}" % (xA, yA,
zA) )
             self.get_logger().info("\t {B:{x:%.3f, y:%.3f, z:%.3f}}" % (xB, yB,
zB))
             self.get_logger().info("\t {angle:%.3f}}" % (angle))

       self.get_logger().info("==================================================
===============================")
             msg = "ros2 service call /spawn_entity 'gazebo_msgs/SpawnEntity'
'{name: \"cube"
             msg = msg + str(xA) + "" + str(yA) + "\", xml: \"<?xml
version=\\\"1.0\\\"?> <sdf version=\\\"1.5\\\"><model
name=\\\"cube\\\"><static>true</static><link name=\\\"box\\\"><pose frame=''>"
             msg = msg + str((xA+xB)/2 + xOffset) + " " + str((yA+yB)/2 +
yOffset) + " " + str((zA+zB)/2 + zOffset + self.zlevel*0.2) + " 0.0 0.0 " +
str(angle)
             msg = msg +
"</pose><inertial><mass>0.01</mass><inertia><ixx>0.01</ixx><ixy>0</ixy><ixz>0</
ixz><iyy>0.01</iyy><iyz>0</iyz><izz>0.01</izz></inertia></inertial>"
             msg = msg + "<visual name='box_visual'><geometry><box><size>"
             msg = msg + str(distance) + " 0.01 0.01"
             msg = msg +
"</size></box></geometry></visual><self_collide>0</self_collide><kinematic>0</k
inematic><gravity>1</gravity></link></model></sdf>"
             msg = msg + "\"}'"

             # self.get_logger().info(" >>> MSG : %s" % msg)

             returned_value = subprocess.call(msg, shell=True)  # returns the
exit code in unix

             self.printCurently = False

       def waitUntil(self, boolean):
             while boolean:
                    time.sleep(0.1)
                pass

def main(args=None):
        rclpy.init(args=args)
        minimal_subscriber = Printer()
        rclpy.spin(minimal_subscriber)
        minimal_subscriber.destroy_node()
        rclpy.shutdown()


if __name__ == '__main__':
        main()
```