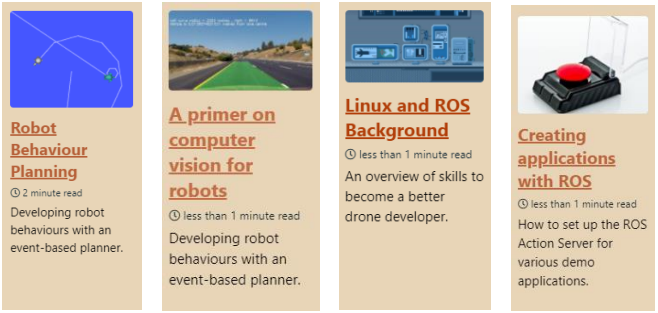
BOT-DRIVES-TO-WAYPOINT PROJECT

This project uses a wheeled bot in Gazebo and programs it to detect and approach a gate. This project, for me, was a primer in the use of a simulator, **Gazebo** with a programmable middleware, **ROS** in a field that interests me, **waypoint following for robotics**. This project is therefore clearly linked to autonomous drones, as they also can navigate by following waypoints. However, the specific usecase here is about:

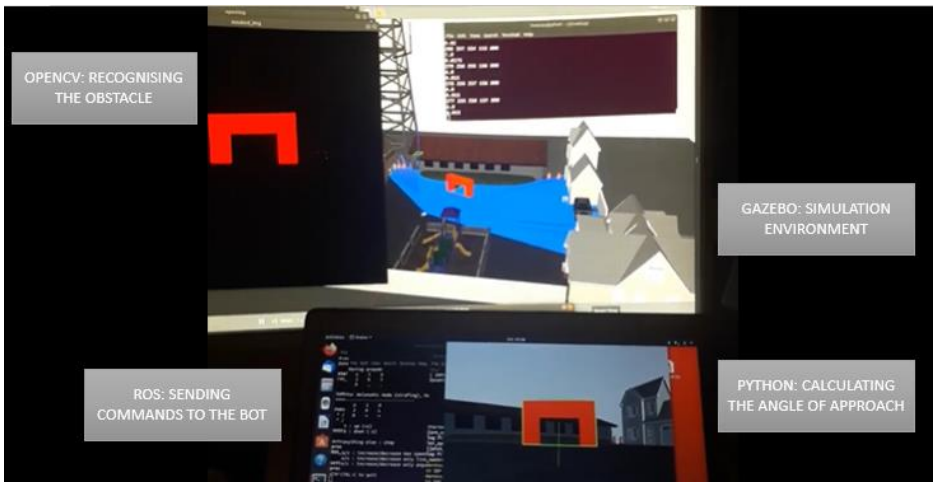
- **coding bot behaviours within the Gazebo simulator based on what the bot observes**,
- while also creating a modular environment intuitive enough for other functionality to be added later.

Various functionalities are designed and we will be able to explore them in greater depth in a series of tutorials. This page, instead, looks at what was achieved in the project. Here are the tutorials for more information.

- Robot behaviour planning
- Basic computer vision for robots
- Linux and ROS background
- Creating applications with ROS



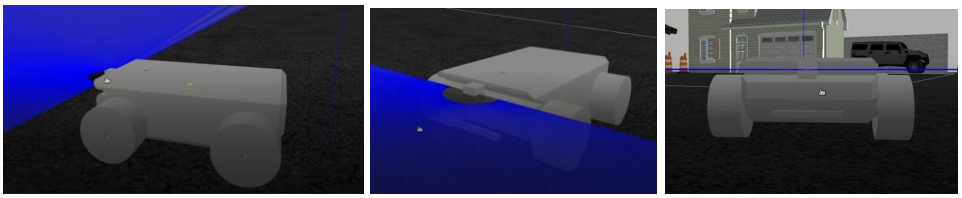
The current github for the project code is available at https://github.com/ThomasCarstens/BotDrivestoWaypoint. The final demo is accessible at https://www.youtube.com/watch?v=rdJYIxeUt-o . It incorporates the technologies shown below.



Note: this document was last updated in August 2020. Please signal any issues at thomas.carstens@edu.devinci.fr .

DESIGNING A WHEELED BOT

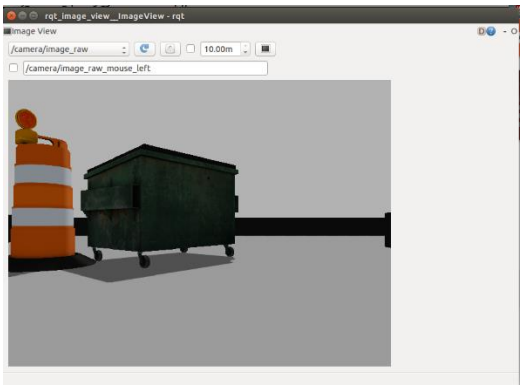
The Unified Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot. I followed a tutorial provided by the ROS documentation to create the URDF format, and another from the Gazebo documentation be able to convert it to SDF (XML file format that is readable by Gazebo).



As you can see, the front two blocks house the lidar sensor (from which emanates the blue rays) as well as the RGB camera (elongated block). The finished version of this bot is accessible on our github.

ADDING A CUSTOM CAMERA TO THE BOT

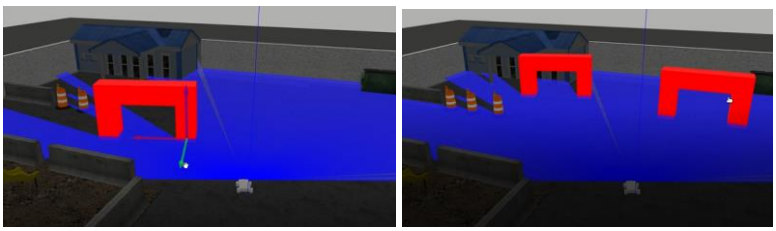
Gazebo allows the integration of cameras in order to visualise the simulation environment. I followed a tutorial provided by Gazebo documentation in order to integrate a custom camera.



MAKING A RED GATE

In order to make the waypoint itself, it was important to examine how textures and colours are created in the Gazebo world, which I discovered in this tutorial.

Interesting note: Adding colour textures, but also custom images, onto an object in Gazebo, actually requires the addition of a separate package for texturing, and it seemed simplest for me to fetch the model from any simulator instance by storing it in the full object in the root /.gazebo/models folder.



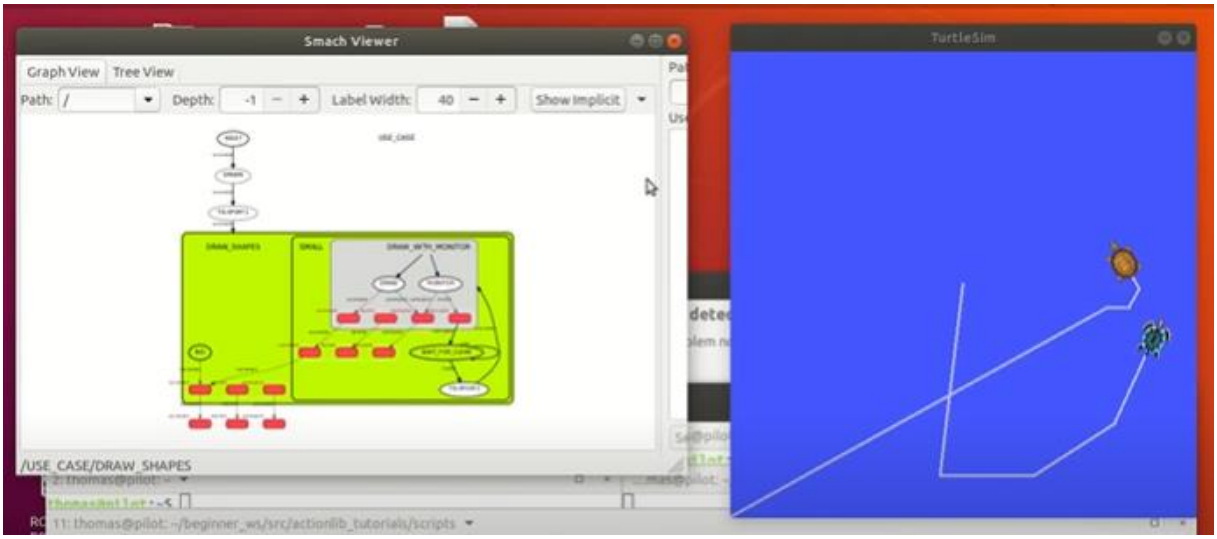
USING A STATE MACHINE TO CONTROL A SEQUENCE OF ACTIONS

Choosing the state machine

My experience previously consisted of designing **digital electronics** state machines. I went on a search for the best framework for my particular case. The different programs that I reviewed were ROSPlan, RSM and Python Smach. The solution needed to work with ROS nodes, but also function independently for the sake of modularity. However, it was not necessary to have a performance-oriented state machine such as ROSPlan, rather one that would be useful for quick prototyping. An added plus would be integration into server-client setups for smooth demoing. The Python SMACH library promised all these elements.

Learning to use the state machine

I followed the official set of tutorials for the Smach documentation. These tutorials work in conjunction with TurtleSim, frequently used to test ROS functionality. The final result of this process is a simulation that is better viewed as a video rather than in pictures, please access it here.



Integrating the state machine into the behaviour script

The behaviour script is available in the github linked in the first page.

Using the behaviour script, we were able to create the demo which you will view on our video.