First Interim Report
Gazebo simulations of contract-based design for an automated valet parking system

SURF student: Tom Andersson
Mentor: Richard M. Murray
Co-mentor: Josefine Graebener

June 29, 2020

1 Introduction

1.1 Background

The increased complexity in cyber-physical systems calls for a modular system design that guarantees safe, correct and reliable behavior, both in normal operation, in the presence of dynamic scenario changes and in failure scenarios [2]. One way of achieving a modular system is to use contract-based design and split up the system into layers, where each layer have specific tasks. Information is sent between the layers according to the rules set up by the contracts. This system architecture not only simplifies the system by splitting it up into several subsystems but in a failure scenario this design also makes it clear in which layer the failure occurred. Other layers can then react and adapt to the new restrictions the failures entail.

These advantages match well with several problems within the development of autonomous vehicles since they have to be developed for an environment constantly effected by dynamic scenario changes due to the interaction with other cars, pedestrians and so on. A first step towards full autonomy can be to limit the scope and focus on an automated valet parking system, a system where costumers drop of their cars and an autonomous robot picks them up one at a time and drives them to a free parking spot. The research group at Caltech has developed python simulations of an automated valet parking system, modularly designed using contracts between the system layers. The simulations demonstrate the ability to control several cars that can be parked and recalled within two minutes in the absence of failure scenarios and in the presence of a simulated undesired stop of a car in the system. The project is stored in this Github repository [1]. Figure 1 shows the layout of the valet parking in the simulations [2].

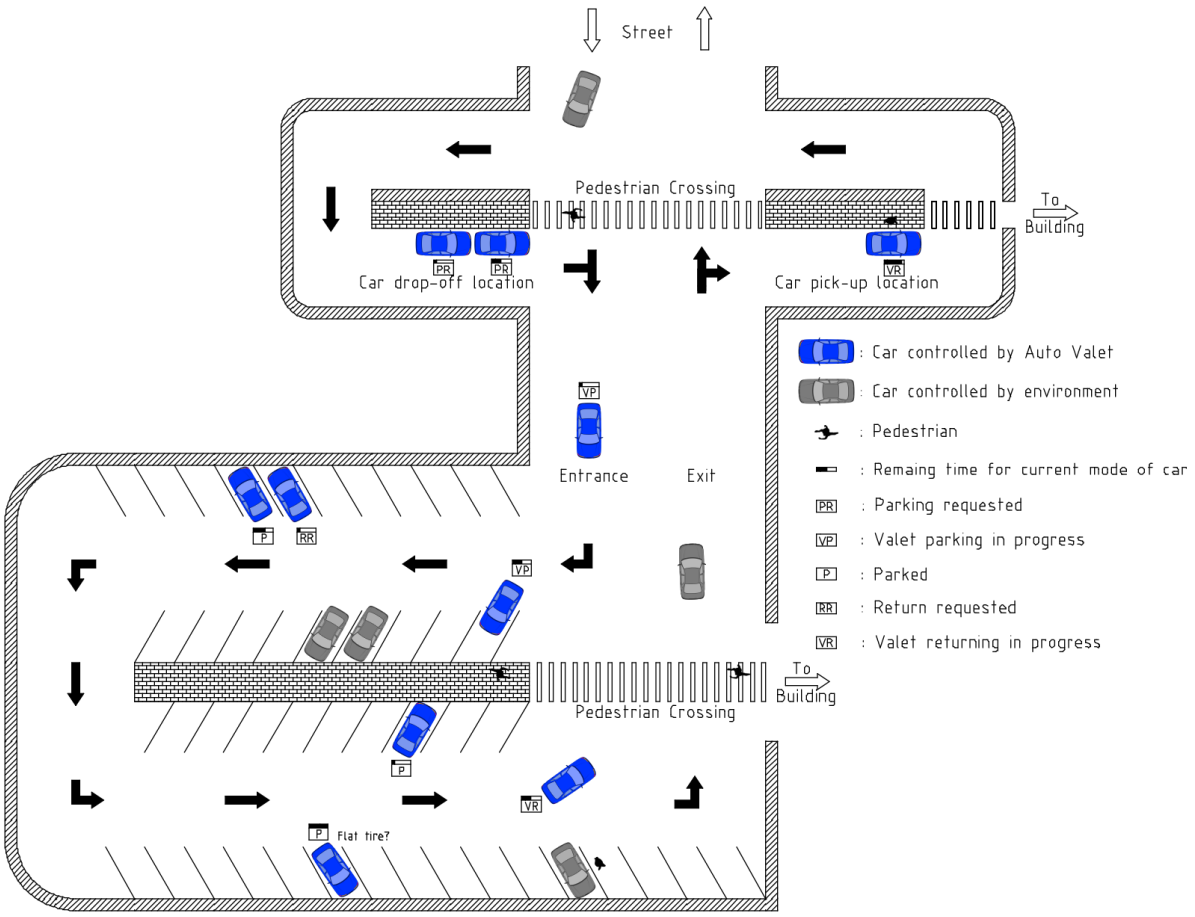

Figure 1: The layout of the valet parking in the python simulations [2].

1.2 Goals

The goals of this project are to prepare the python simulations for a small scale test that would incorporate real life aspects and the possibility to physically separate components in the system by converting the code into ROS-nodes and performing simulations of 6 real robots in the Gazebo simulator. If this is done on time, the system will also be further developed by implementing the functionality to under certain conditions lighten the specifications when a car is unable to achieve its goal. For example, when the route of a car is blocked but there are no other cars nearby the car can be allowed to track the path more loosely, if the car cannot reach its assigned parking spot but there are plenty of other free spots it can be allowed to pick another one or, as a last resort when a goal cannot be achieved, the car can instead focus on not being in the way for any other car in the system until the blocking is cleared.

1.3 Objectives

- Become familiar with the work so far done by the research group. (1 week)
- Learn about Gazebo and choose appropriate robots to simulate. Perform simulations by using commands to set speed and steering angle. (2 weeks)
- Build a model of the valet parking environment in Gazebo. (1 week)
- Adapt the simulations of the camera to detect and distinguish walls, lanes, pedestrian crossings, moving pedestrians, parking spots and moving cars in Gazebo. (1 weeks)
- Adapt the other programs into ROS-nodes and perform the simulations on the Gazebo model. (2 weeks)
- If the project is on time at this point, implement the functionality to under certain conditions lighten the specifications when a car is unable to achieve its goal. This shall be done for three different cases, allowing the car to choose another parking spot if there are plenty of available spots, allowing the car to deviate from its specified route if there are no other cars nearby and, as a last resort, commanding the car to keep lanes and parking spots sought by other cars free until the blocking is cleared. (4 weeks)
- Finalize a written report. (1 week)

2 Progress

After the initial process of learning about the current state of the project the choice of robot to simulate was analysed through a comparison between turtlebot2 and turtlebot3 burger. These two robots were chosen because the research group already had access to 2-3 turtlebot2 robots and turtlebot3 burger is its successor. This comparison was made to optimise for a small scale test at a later point in time. The collected data can be found in figure 2. This comparison shows that turtlebot3 burger is a good choice when it comes to price, official compatibility and size, a smaller size reduces the size of the physical area needed during a small scale test. In parallel with this comparison, the process of learning about the Gazebo simulator was ongoing by going through tutorials and running demos with turtlebots in Gazebo.

Current official compatibility

Robot	Latest ROS version	EOL
Turtlebot2	Kinetic	April 2021
Turtlebot3	Melodic	May 2023

Size

Robot	Size (L x W x H)
Turtlebot2	354 x 354 x 420 mm
Turtlebot3 burger	138 x 178 x 192 mm

Cheapest price from official resellers (US/worldwide)

Robot	Including	Price (USD)	Reseller
Turtlebot2	Everything but computer and camera	1049	CLEARPATH ROBOTICS
Turtlebot2	Everything	1900	Dabit Industries
Turtlebot3 burger	Everything	549	Dabit Industries

Cost to equip the lab with 6 robots

Robot	Nbr in lab already	Total price (USD)
Turtlebot2	2-3	5700 - 7600 3147 - 4196 plus laptops and cameras
Turtlebot3 burger	0	3294

Figure 2: A comparison between turtlebot2 and turtlebot3 burger.

With the help from these tutorials, [3], a launch file for the turtlebot3, one_robot.launch, one launch file for launching multiple turtlebots at specific positions, robots.launch, and one main launch file for launching the turtlebots in an empty world, main.launch was constructed. Figure 3 shows 2 turtlebots launched in an empty world.

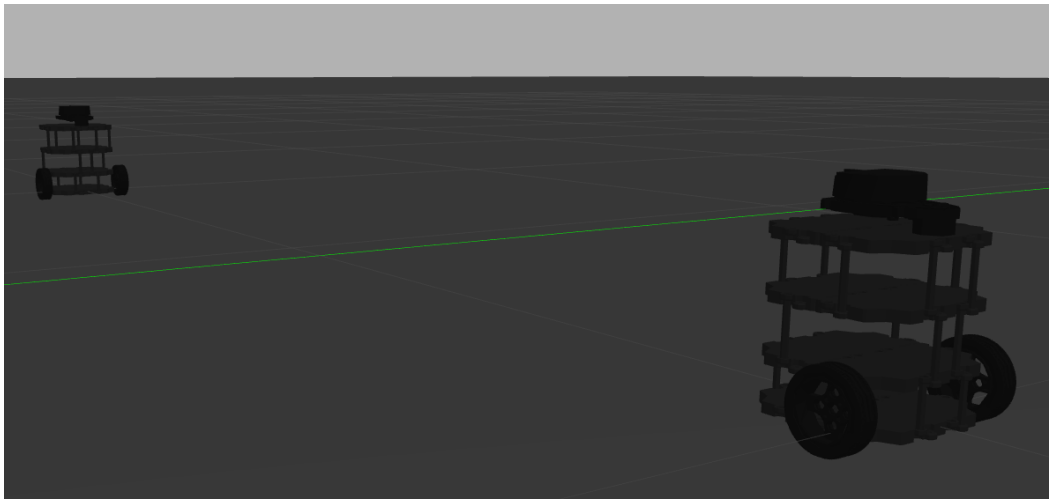

Figure 3: 2 turtlebots launched in an empty world.

The next step was to control the turtlebots through ROS. The turtlebots can be controlled by setting linear and rotational velocity. To keep going, the turtlebots need to continue to receive new data, if the data stops coming the turtlebot will stop. The robots also publishes a ROS topic that describes the current states of the robot, i.e position, orientation, velocity etc. A python program, robot_communication, that publishes robot commands and subscribes to the states of the robots was written. To change the commands that are published and request information of the states of the robots, the class RobotCtrl was written. This class will only receive and send topics when asked to while robot_communication does it at a given frequency. To test this functionality, a test program was written. The described ROS-structure is shown in figure 4. One can choose how many robots to simulate by changing a global variable and change the number of robots in the robots.launch file.

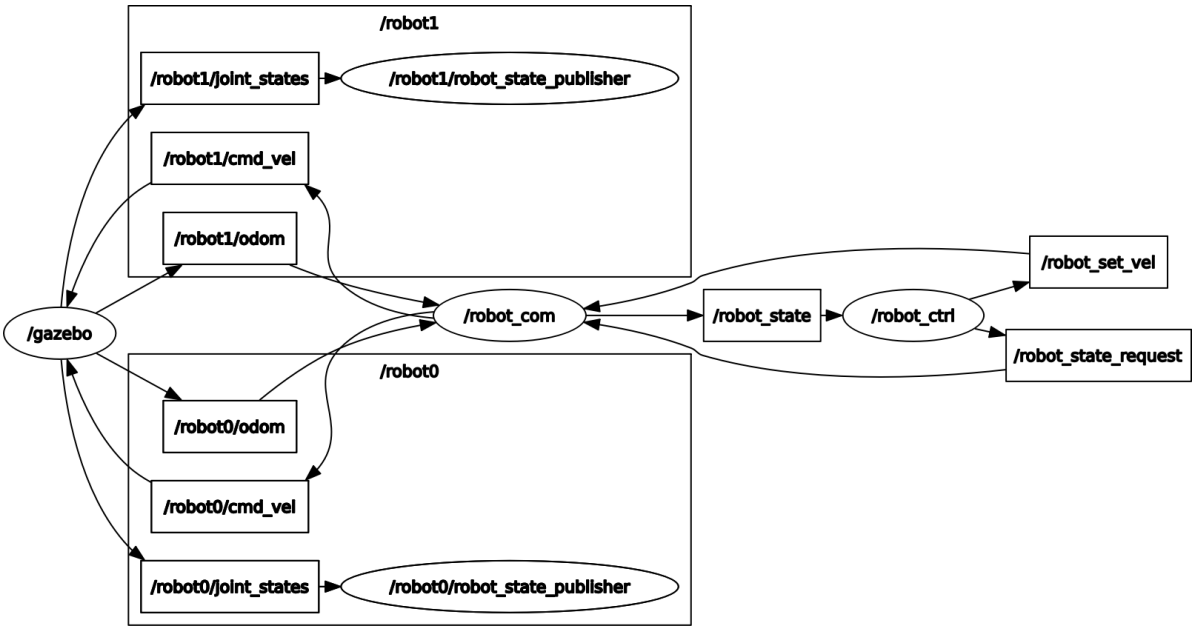

Figure 4: The current ROS structure.

After the control of the robots was implemented a parking lot environment was built in a Gazebo world. The environment was reduced in size, compared to figure 1, to be better fitted for simulations with only 6 robots and to take up less space during a future physical test. This world was built by adding an image of a reduced parking lot as a ground plane in the world. This image was scaled so that the turtlebot3 burger would fit in the parking spots. Thereafter, walls were added on top of the ground plane image. See figure 5. The next task will be to add pedestrians into the Gazebo world.

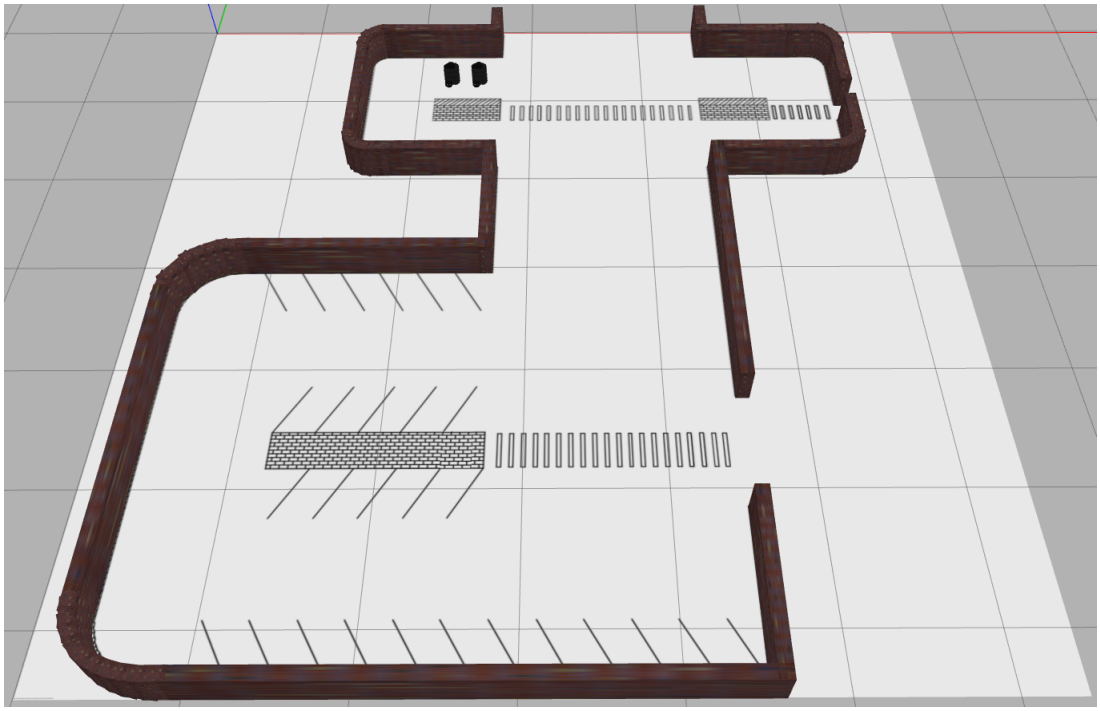

Figure 5: The parking lot environment implemented in a Gazebo world.

3 Challenges

So far, the major challenges has been to identify how ROS, Gazebo and the turtlebot description are supposed to interact with each other. For example, how to launch turtlebots in a gazebo world, how to set velocity of a robot and how to read the states of a robot. This was solved by taking one problem at a time and search the web for demos and tutorials. A so far unsolved problem is how to make a dynamic launch file so one only has to change a global variable to change the number of robots being launched instead of also having to change a launch file. An anticipated challenge that is coming up is how to adapt the current control functions to work with how the turtlebots are controlled, i.e by setting linear and rotational velocity rather than a throttle and a steering angle.

References

[1] Josefine Graebener et al. *Automated Valet Parking*. URL: `https://github.com/jgraeb/AutoValetParking` (visited on 07/01/2020).

[2] Richard M. Murray and Josefine Graebener. *SURF 2020: Hardware Implementation of Contract-Based Design for Automated Valet Parking System*. URL: `http://www.cds.caltech.edu/~murray/wiki/index.php?title=SURF_2020:_Hardware_Implementation_of_Contract-Based_Design_for_Automated_Valet_Parking_System` (visited on 07/01/2020).

[3] Arif Rahman. *ROS QA 130 – How to launch multiple robots in Gazebo simulator?* URL: `https://www.theconstructsim.com/ros-qa-130-how-to-launch-multiple-robots-in-gazebo-simulator/` (visited on 07/01/2020).