User Manual

Multiple Autonomous Vehicles in Complex Scenarios

December 13, 2021

Version 0.1

CONTENTS

1 INTRODUCTION

The research on autonomous vehicles has become a popular topic over recent years, engaging academia and industry alike. Although the development has matured to a degree such that several companies already offer fully autonomous taxi services and freight transport solutions, there still exist numerous challenges in the field. In particular, the interaction with other traffic agents is an especially challenging aspect.

This user manual will describe how to test scenarios for an autonomous vehicle; both in simulation and with a physical car.

2 PREREQUISITES

In order to operate with the system a computer with (preferably) Ubuntu 20.04 or other operating system with an installation of ROS 1 Noetic [1]. Also, Python 3 should be installed and the required packages can be installed with the command `pip install -r ~/mafiks_ws/requirments.txt`.

3 CONFIGURATION

In the work-space folder there are two bash scripts to help set up the IP-configuration required for ROS, called *setup_physical.bash* or *setup_simulation.bash*. When simulating locally the script for simulation should be executed. When wanting to drive with a physical car or interacting with other users on a network, the physical script should be executed. NOTE: the IP-address corresponding to `ROS_MASTER_URI` needs to be the correct IP-address for the master.

4 QUALISYS

Qualisys Motion Capture System [2] is the service for global localization of a physical car and is installed in Visionen at LiU. It makes use of infrared cameras which detect reflective balls on physical objects to determine its position. These reflective balls are mounted on the cars in a unique pattern to make them distinguishable from one another. The system needs to be started from the stationary computer in Visionen's command-center. For the tracking to function well with our setup, there is a configuration file saved at `A:/mafiks` on the computer. After staring the program with the configuration file, tracking of objects is possible. As said, the object are mounted with reflection balls seen as red dots in the program. To define a body, hold shift and select all the points associated to the object, right-click and choose "define rigid body". The name of the body is important for ROS-topics, as qualisys will publish an odom message on *qualisys/<name>/odom* containing position, orientation, speed and angular velocities of the object.

5 PROJECTORS

There are three projectors in Visionen at LiU. Two of them collaborate to make the canvas on the floor, and the third displays on the wall. For the projectors the second computer in Visionen's command-center should be used. On the computer's desktop there is one script for starting the projectors, and one for turning the projectors off. **Nothing regarding the projectors should be altered with!** One can display either on the floor or on the wall. In our repository there is a Matlab script which subscribes on ROS topics and draws the map and the car's trajectory on the floor. This, unfortunately, requires manual scaling on the floor.

6 DRIVING ENVIRONMENT

The world the cars drive in is a circular path defined in `/mafiks_ws/src/perception/world/config/parame` There one can set the radius, lane width, resolution i.e number of coordinate point, number of segments and track angle. Track angle is used to get a oval path and should not extend $\frac{\pi}{12}$ radians. The limitation is based upon maximal steering angle for the physical car. Number of point needs to be large enough, so that the maximal distance between two coordinates is smaller than the lookahead for the purepursuit (see section 8.1).

7 ROS NODES

In the workspace there are many different nodes and launch-files for ROS. These are the crucial packages for collaboration of the code and for modules. A launch-file can start multiple nodes, modules, but also send parameters to them.

7.1 Launchfile: cascar

One of the nodes in ROS is called *cascar*. This includes the main launchfile for the cars. Once a car has started this can be called with `roslaunch cascar cascar.launch`. This launch starts the inertial measurement unit (imu) for the car, the connection to the arduino for movement, and the lidar tracking. After this launch the car is all set to go. Note that this has to be done on each physical car that is being used. It is recommended to log in to the cars with SSH and run the launch file from the SSH command line.

7.2 Launchfile: init.launch

A package called *launcher_utils* includes what needs to be launched on another computer when *cascar.launch* has been run on the physical cars. Launching this is computationally heavy and requires a relatively pow-

erful computer. A virtual machine with 2 assigned cores on a normal laptop might not be enough, but a computer with at least 4 cores which runs Ubuntu natively should do the work. The launch is called with:

```
roslaunch launcher_utils init.lauch ego_is_physical:=0/1
num_physical_cars:=0/n num_simulated_cars:=0/m
```

The parameters in the command are voluntary and have default values (written first). *ego_is_physical* describes if the smart car also is a physical car, and the latter parameters are used to set how many simulated and/or physical cars that should be launched.

This launch file is somewhat complex, but launches all that is needed in the correct order and on the right namespaces for each car. This launch file starts the subscription from Qualisys tracking and starts RVIZ . It also launches the state estimator, controller, behavior layer and path planner that is needed for each car.

describe w it is

8 OPERATING WITH THE CAR

The physical car is a miniature car which gets it's power from batteries, which should be handled with care. Once connecting the battery the car starts, but to start the Arduino the switch seen in Figure 1 needs to be flipped, upon a melody is played to know that it is turned on. The raspberry pi on the car can either be plugged into a separate screen and operated as a normal computer, or one can use SSH to operate it remotely. We suggest the latter. The raspberry is connected to the Arduino with a USB-cable, so if the connection fails on startup (or launching cascar) this can be checked. The same goes for the lidar, which also gets it's power from the raspberry. When the lidar is launched it should spin.

8.1 Parameters

The car uses a pure-pursuit as lateral controller. This can be changes in *controller_params* in ∼`/mafiks_ws/src/planning_regulator/controller/src/main.py`. A typical lookahead is 0.6m where a shorter distance gives narrower path following but more oscillatory and a larger gives a offset from planned path. Here, one can also change car specific parameters which not should be altered unless the car is changed and the *goal_tol* which determines the tolerance distance for the goal.

The lidar is mostly used as a safety measure to not collide with obstacles. The lidar detects obstacles in a cone ahead of the car and the distance is tuned to be smallest possible without collition. This distance can be altered in *lidar_opts* in ∼`/mafiks_ws/src/planning_regulator/behavior_layer/src/behavior_layer_definition/behavior_layer.py`.

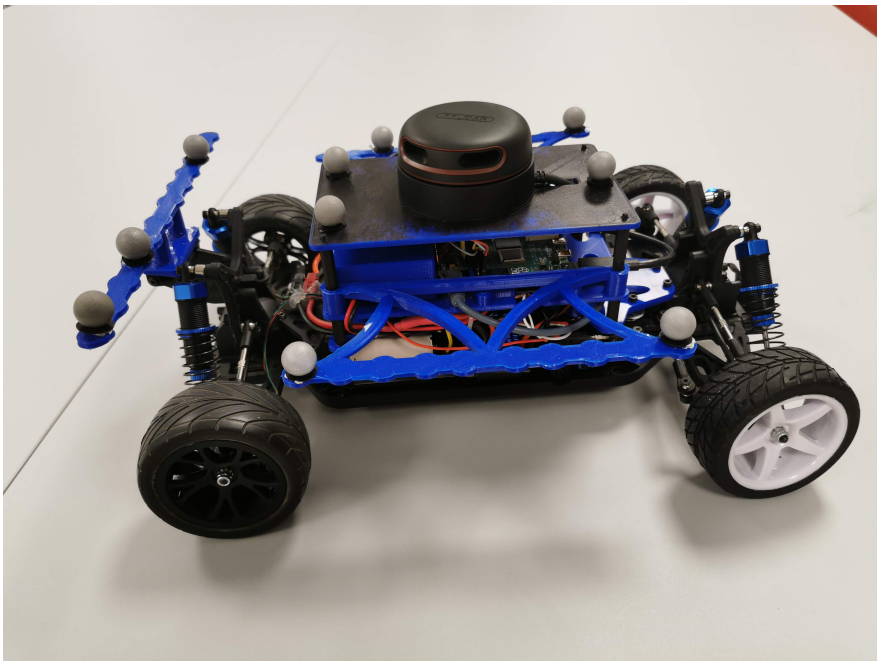

Figure 1: The RC-car, motion capture balls for tracking and lidar on top.

REFERENCES

[1] ROS, *Robot operating system*, 2021-11-22. [Online]. Available: http://wiki.ros.org/noetic.

[2] Qualisys, *Motion capture – mocap*, 2021-11-22. [Online]. Available: https://www.qualisys.com/.