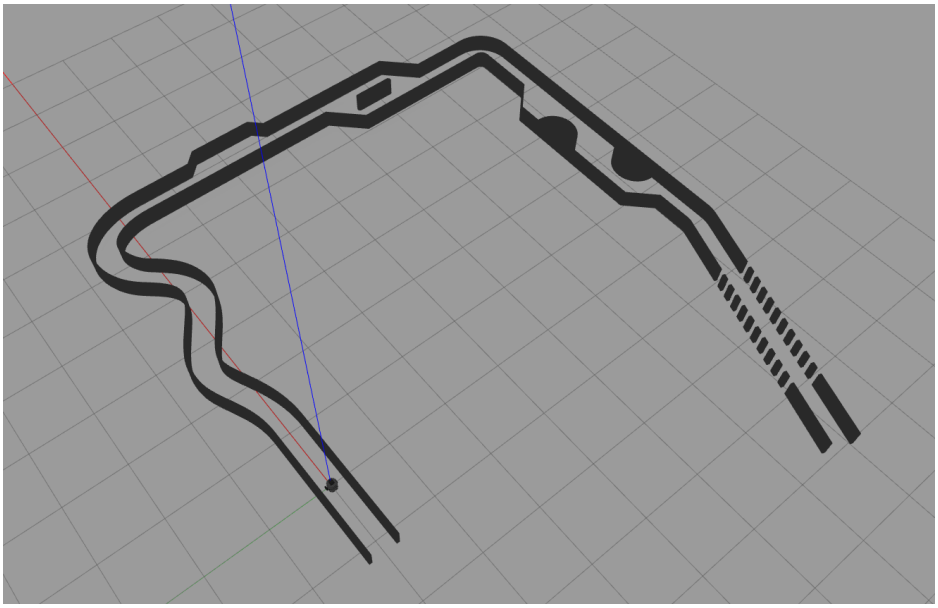
Project Work

Race Track

Task: write a python ROS node that navigates the TurtleBot through the provided race track without hitting any walls.



The goal of the project work is to apply the vehicle-control information you have learned during the course, as well as encourage creative thinking and problem-solving. The race track challenges the vehicle control implementation beyond simple lane following, rudimentarily simulating difficulties that vehicle perception and control encounter: lanes can contain notable curvature, the lane width can change, there might be multiple lanes, sharp turns often occur in intersections, obstacles such as parked cars may block the lane, and lane markings can be incomplete. The race track simulates these conditions for the Lidar. Your task is to control the Turtlebot through the track, without hitting the walls. You are free to use any control strategy for solving the task.

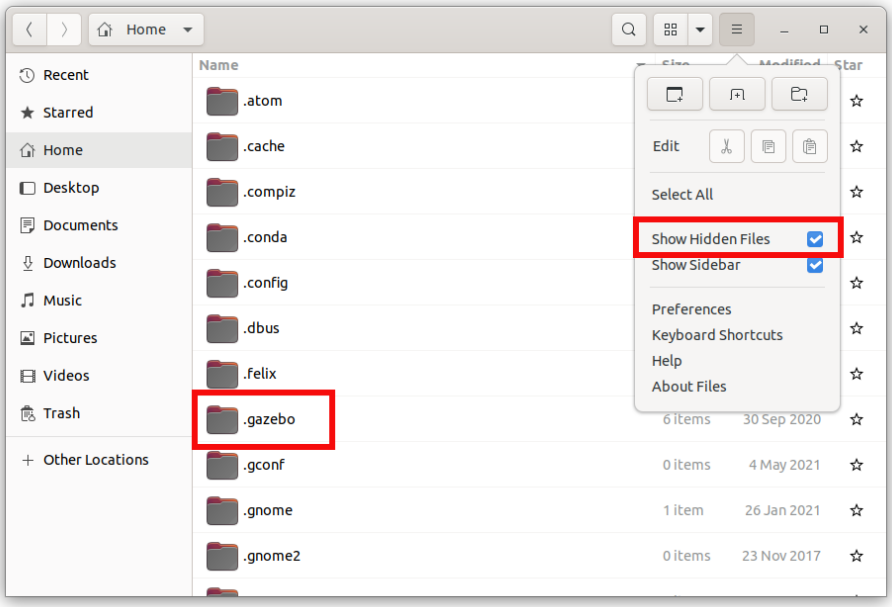
Setup

In the MyCourses-page of the project work, two zip-files are provided: **vmc_project.zip** and **project_track.zip**. The zip-file **vmc_project.zip** contains a template package that you will use for the project work. **project_track.zip** contains a 3D-model of the race track, that the vmc_project package will insert in Gazebo.

1. Extract the folder from **project_track.zip** to the folder ∼/**.gazebo**/**models**/

   The .gazebo folder is a hidden folder, see below for navigation tips

2. Extract the folder from **vmc_project.zip** to ∼/**catkin_ws**/**src**/.



3. Build your catkin workspace, remember to run the command in the ~/**catkin_ws**/ folder.

```
$ catkin_make
```

4. Source your **.bashrc** so that ROS finds the new package

```
$ source ~/.bashrc
```

5. Launch Gazebo with the following command

```
$ roslaunch vmc_project project_world.launch
```

Getting started

Your control implementation must be a ROS-node in the **vmc_project**-package implemented in a file named **racer.py**. Consequently, the ROS node must be executable with the following command

```
$ rosrun vmc_project racer.py
```

Utilising your implementation to Exercise 6 is a good starting point for the project. The following tips should help you clear out the track, a successful control should consist of multiple different components:

- At first, use a low & constant linear velocity. Later on, consider using higher linear velocity and implementing longitudinal control, which slows down the Turtlebot when the Lidar sees walls in front.
- Lateral control can utilise some type of crosstrack error to keep a safe distance to the surrounding walls.
- Processing the Lidar data in sectors might be a good idea. A single reading might not be as reliable as for example 10 readings in a batch.
- Use the Lidar data to detect obstacles in front. Aim for "obstacle free" areas in the Lidar view utilising heading error/goal point to attractive areas.
- Make your control robust to outlier sensor readings, prepare for the occurrence of *inf*-values in the Lidar data.

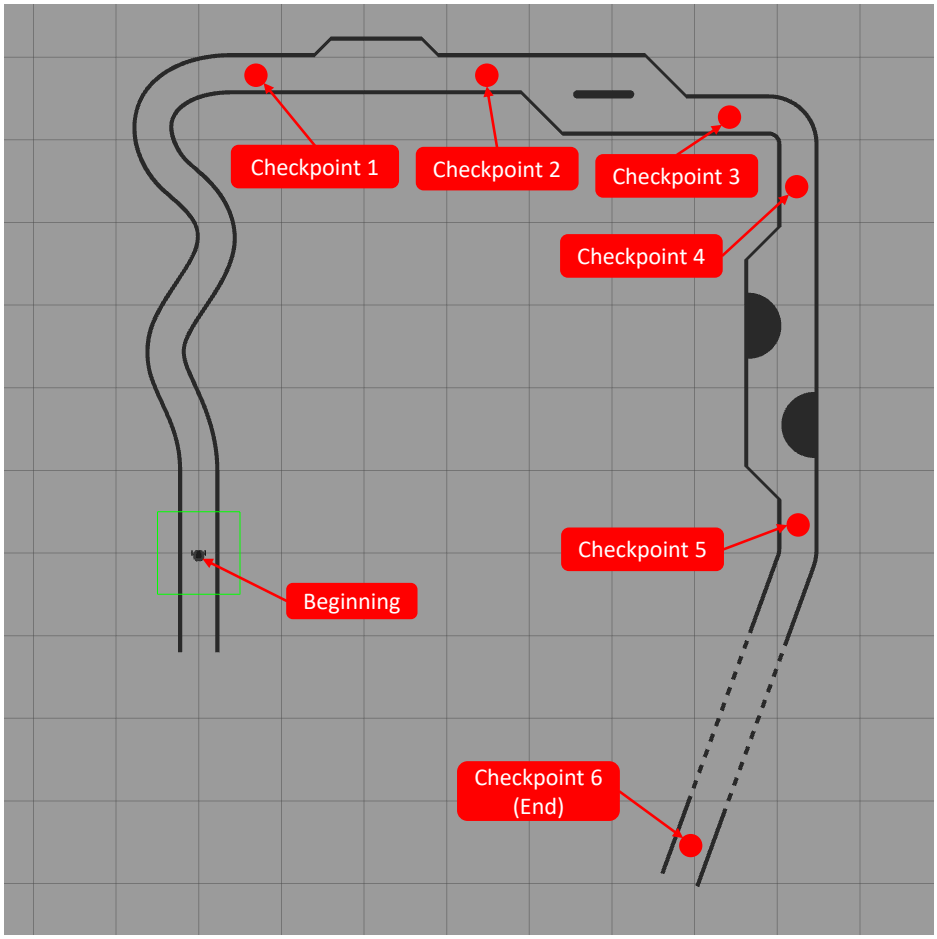
Grading

In order to get full points (75p) for the project work, the following requirements have to be met:

- Your implementation must be capable of driving the Turtlebot through the track from beginning to end without hitting any walls. (60p)
- The Turtlebot must drive through the entire track in less than three minutes. (10p)
- You must present an overview of your solution at the Project Work Gala. (5p)

The race track is divided into "checkpoints", which determine your grade if your solution is capable of only partially clearing the track. Reaching the checkpoints grants you 10 points each, with a total of 6 checkpoints. The checkpoints are highlighted in a figure below. To prevent bottlenecks caused by getting stuck at a certain part of the track, you are free to skip parts you find difficult

by "teleporting" to the checkpoints. Any checkpoints you teleport to do NOT naturally grant you the points for reaching them. Bash-commands for teleporting the Turtlebot to each checkpoint are provided at the end of the document.



Note that the completed checkpoints must be reached with a single control solution, the one you submit implemented in your racer.py-node. You are NOT allowed to change the control (modify code, different PID parameters, etc.), between teleporting the Turtlebot to different checkpoints.

In addition to pure results, solid effort is also taken into consideration in the grading. If you are unable to reach some checkpoint, you can document your tested control approaches and the reasoning behind them. This will award you some points for the checkpoint, based on the clarity of the documentation and logic of the control approach.

Below there is a concrete example to clarify the grading process:

Example project work grading

- The Turtlebot is capable of driving from the beginning to checkpoint 3 without hitting any walls. (30p)
- After being teleported to checkpoint 4, the Turtlebot is capable of driving to checkpoint 6 (end of track) without hitting any walls. (20p)
- The control solution is presented at the Project Work Gala. (5p)

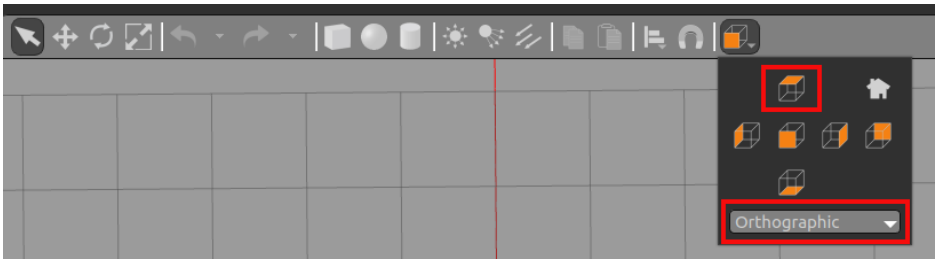
Final grade for the Example project work is 55/75 points. 5 checkpoints were reached, and the work was presented at the Gala.

Project work gala presentation

The project work presentation has a time limit of five minutes. The presentation should consist of around 3-5 slides, which give a short overview of the selected control strategy, results, and observations of the control performance (pros and cons). The presentation is given at the Gala, location and date can be checked from MyCourses.

Project documentation

Document your results by recording your screen while the Turtlebot drives through the track. If you are only completing certain checkpoints from different parts of the track, you can record multiple videos, each highlighting the Turtlebot passing the specific checkpoints. If your Turtlebot is capable of driving through the entire track in under 3 minutes, make sure that the Gazebo "sim time" is clearly visible in the video to get the points for your accomplishment. For clearer videos, please switch the Gazebo view to "top-view" and "orthographic" (see figure below). See



the example project work result video for reference here.

You can use any tool you want for recording the screen. The Ubuntu desktop has a built in screen-recording tool, which can be initiated and finished via the same keyboard shortcut: **ctrl-shift-alt-r**. An orange dot should appear in the top right corner of your desktop, indicating

that the recording is in progress. Note that the default maximum length of the recording is 30 sec. Before recording, this maximum length should be extended to for example 600 sec

```
$ gsettings set org.gnome.settings-daemon.plugins.media-keys
↪  max-screencast-length 600
```

In addition to the video, you must submit your **vmc_project** package containing your **racer.py** implementation. The submitted **racer.py** implementation must be capable of replicating the results presented in your video(s)!

If you wish to gain points for effort put into checkpoints you were unable to clear, you must submit clear documentation showing your experimentation process. This documentation should contain written/graphical explanations of the tested control approaches in PDF-format, as well as screen recordings highlighting the results achieved with the tested approaches.

Deliverables

Submit a single zip-file, containing the following documents

1. The **vmc_project** package, containing your **racer.py** implementation.
2. Video(s) showing the Turtlebot clearing the track/checkpoints
3. Optional: documentation highlighting the effort you've put into checkpoints that you were unable to reach.

Commands for teleporting the Turtlebot

Before teleporting the Turtlebot, remember to shut down your control node (**racer.py**).

Then, set Turtlebot velocity commands to zero:

```
$ rostopic pub -1 /cmd_vel geometry_msgs/Twist "
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
"
```

Below are teleportation commands for different checkpoints on the track

Teleport to the beginning:

```
  $ rostopic pub -1 /gazebo/set_model_state gazebo_msgs/ModelState "model_name: 'turtlebot3_burger'
pose:
  position:
    x: 0.0
    y: 0.0
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 0.0
twist:
  linear:
    x: 0.0
    y: 0.0
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 0.0
"
```

Teleport to checkpoint 1:

```
rostopic pub -1 /gazebo/set_model_state gazebo_msgs/ModelState "model_name: 'turtlebot3_burger'
pose:
  position:
    x: 5.8
    y: -0.5
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.707
    w: -0.707
twist:
  linear:
    x: 0.0
    y: 0.0
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 0.0
"
```

Teleport to checkpoint 2:

```
rostopic pub -1 /gazebo/set_model_state gazebo_msgs/ModelState "model_name: 'turtlebot3_burger'
pose:
  position:
    x: 5.8
    y: -3.5
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.707
    w: -0.707
twist:
  linear:
    x: 0.0
    y: 0.0
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 0.0
"
```

Teleport to checkpoint 3:

```
rostopic pub -1 /gazebo/set_model_state gazebo_msgs/ModelState "model_name: 'turtlebot3_burger'
pose:
  position:
    x: 5.3
    y: -6.5
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.707
    w: -0.707
twist:
  linear:
    x: 0.0
    y: 0.0
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 0.0
"
```

Teleport to checkpoint 4:

```
rostopic pub -1 /gazebo/set_model_state gazebo_msgs/ModelState "model_name: 'turtlebot3_burger'
pose:
  position:
    x: 4.5
    y: -7.25
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 1.0
    w: 0.0
twist:
  linear:
    x: 0.0
    y: 0.0
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 0.0
"
```

Teleport to checkpoint 5:

```
rostopic pub -1 /gazebo/set_model_state gazebo_msgs/ModelState "model_name: 'turtlebot3_burger'
pose:
  position:
    x: 0.3
    y: -7.25
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 1.0
    w: 0.0
twist:
  linear:
    x: 0.0
    y: 0.0
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 0.0
"
```