Introduction to ROS
Group Project: Autonomous Driving

1 General Information

1.1 Timeline

- 17.06.2025: Announcing the projects
- 17.06.2025: Release of code
- 01.08.2025: Submission

1.2 Submission

The project submission consists of three parts:

- Source Code (Deadline: 01.08.2025 - End of day)
- Documentation (Deadline: 01.08.2025 - End of day)
- Presentation (Deadline: Dates will be shared later)

Please register for the exam via TUMonline. The exam is online by now.

The presentation will not be graded but shall help the reviewers to quicker understand your code structure, limitations and issues you had during implementation.

Your source code and documentation must be submitted via a link to a gitlab or github repository. In both cases you should include the src folder of your catkin workspace as well as the source code folder(s).

1.2.1 Source Code

While writing code, please pay attention to writing clean code, think about how to structure your code base into individual ROS packages and nodes. Grading will as well be dependent on the structure of your project.

Please also include a `readme.md` file with detailed step by step installation and launch instructions necessary to run your code base. Make sure that all code can be launched by a single launch file. We don't want to launch your code from multiple terminals. Indicate in your readme which Ubuntu version has been used. As a hint, before submission run your code from a fresh installation of Ubuntu to make sure you have all necessary installation steps in your documentation!

You are allowed to use any free available code but you are NOT allowed to use source code from other groups of this course.

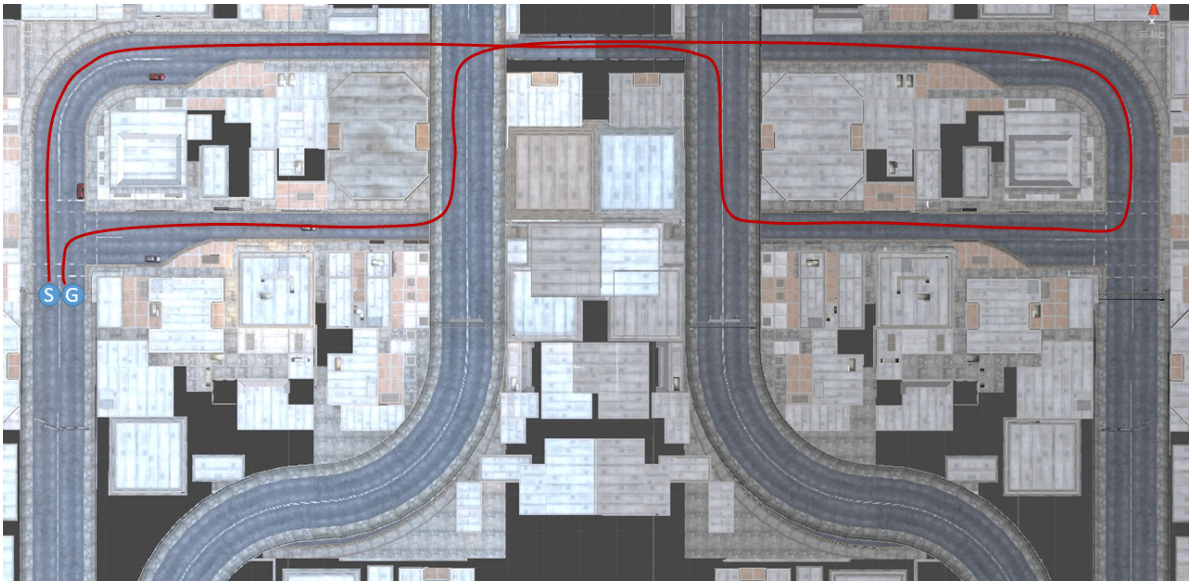

Figure 1: The track to follow

1.2.2 Documentation

The documentation shall be approx. 4 pages but is not limited to that. The goal of the documentation is to help us understanding your code. In your documentation you should include a short description of every ROS node and package and its functionality. Please let us as well know in case you were not able to fulfill all requirements and explain the problems. Document where you used external code, not written by yourself. Finally, discuss who of the team members worked on which components. Beyond that, you must include

- a ROS graph, indicating who worked on which part
- figures, and plots presenting your results,
- a bibliography.

1.2.3 Presentation

Please prepare an oral presentation of your documentation. The presentation length shall not be longer than 5 minutes. An introduction to the field and topic is not required. The goal of the presentation is to give us a quick overview over your project, explain how you implemented the individual components, discuss limitations and issues and give us feedback about the project work.

2 Task

2.1 Goals

The goal of the project is driving a predefined track (see figure below with marked start and goal location) as fast as possible in an urban environment without leaving the road, crashing into other cars and obeying traffic lights.

The core parts, including but not limited are:

- A Unity simulation environment. A base version is provided to you.
- A ROS-Simulation-Bridge providing ROS interfaces (topics, services). It will communicate with the simulation via TCP while at the same time providing relevant information to other ROS nodes. A base version will be provided to you. You are free to adjust the code.
- A car controller, enabling acceleration, breaking and steering rate. A base version is provided to you.
- A semantic camera structuring the environment into semantic classes like road, cars, street sign, etc.
- A state machine for your car, managing adjusting to street rules.
- A perception pipeline that converts the depth image, first to a **point cloud** and second to a **voxel-grid** representation of the environment.
- A path planner that generates a path through the environment.
- A trajectory planner that plans a trajectory based on the found path.

It is required that you at least once implement one of the following ROS elements by yourself:

- ROS service - document where you used a ROS service call
- Implement an own message type - document which message you defined by yourself

2.2 Perception

The system is equipped with a depth camera, two RGB cameras, one semantic camera, and an inertial navigation system (INS).

2.2.1 Cameras

All cameras are mounted at the front of the ego vehicle. For the exact pose and coordinate frames of the cameras, refer to Fig. 2 and Fig. 3. The resolution and field of view of each camera can be adjusted in the configuration file; see the project repository for more details.

2.2.2 INS

To acquire precise vehicle positioning, an onboard INS provides the exact pose (including position and orientation) and twist (including velocity and angular velocity). For the sensor's position and orientation, refer to Fig. 2 and Fig. 3.

2.2.3 Your Tasks in Perception

- **Coordinate Frame Build-up:** The transformation from the world frame to the INS is provided. You need to establish coordinate frames for the other sensors using the `tf2` package.
- **Occupancy Map Generation:** Navigation requires an occupancy map. A common approach involves converting depth camera frames into point clouds, and using the `OctoMap` package to generate it.
- **Traffic Light Recognition**: Your vehicle is expected to stop at red lights. Therefore, a traffic light recognition algorithm should be implemented.

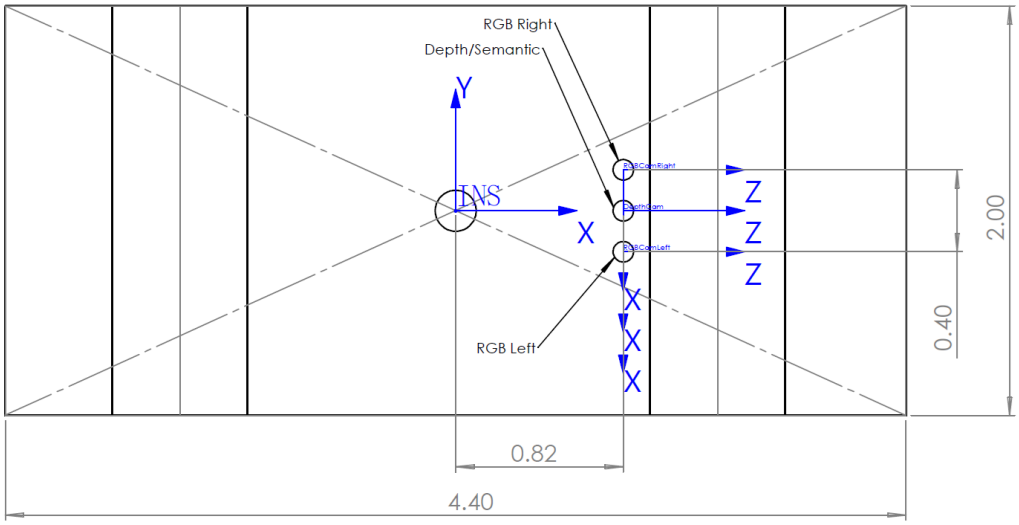

Figure 2: Important vehicle dimensions(bottom view).

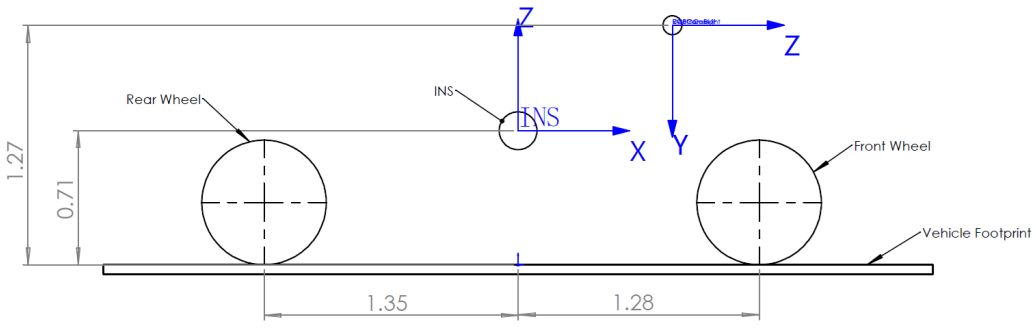

Figure 3: Important vehicle dimensions(side view).

2.3 Planning

To navigate through the city, your vehicle will require a trajectory planner to determine where and when to turn. A path planning module must be developed accordingly.

2.3.1 Your Tasks in Path Planning

- **Short-term Goal Selection:** Based on the track shown in Fig. 1, your planner may use predefined poses as short-term goals for planning. However, you still need to select the next target from this list of goals.
- **Path Planner (optional):** With the target pose defined, you may implement a path planner that computes a geometric path, ignoring kinematic constraints. This is optional, as some methods (e.g., MPC) can directly generate feasible trajectories.
- **Trajectory Planner:** Implement a trajectory planner that incorporates kinematic and dynamic constraints, producing feasible paths for the control module.

2.4 Decision Making

2.4.1 Traffic Lights

Your vehicle must handle four intersections with dynamic traffic lights and come to a complete stop at red lights, regardless of the intended direction. Exact stopping at the waiting line is **NOT** required.

2.4.2 Optional Event I: Vehicle Merging

After the first intersection, a non-player character (NPC) vehicle will merge into your lane from the right. Your ego vehicle should adjust its speed to avoid collisions. Overtaking is optional and up to your design.

2.4.3 Optional Event II: Emergency Brake

While returning, another NPC vehicle will cross and then brake hard in front of your vehicle. Your ego vehicle should detect the hazard and perform an emergency stop to prevent a collision.

2.4.4 Your Tasks in Decision Making

- **State Machine(optional):** You may implement a state machine to coordinate planning and control responses to traffic lights and events. This is not mandatory as long as the behavior is correctly handled.

2.5 Vehicle Control

The control module should receive the planned trajectories and send appropriate control commands to the driving simulator.

2.5.1 Your Tasks in Vehicle Control

- **Control Algorithm:** Implement a control algorithm that follows the planned trajectory by outputting correct throttle, brake, and steering commands.

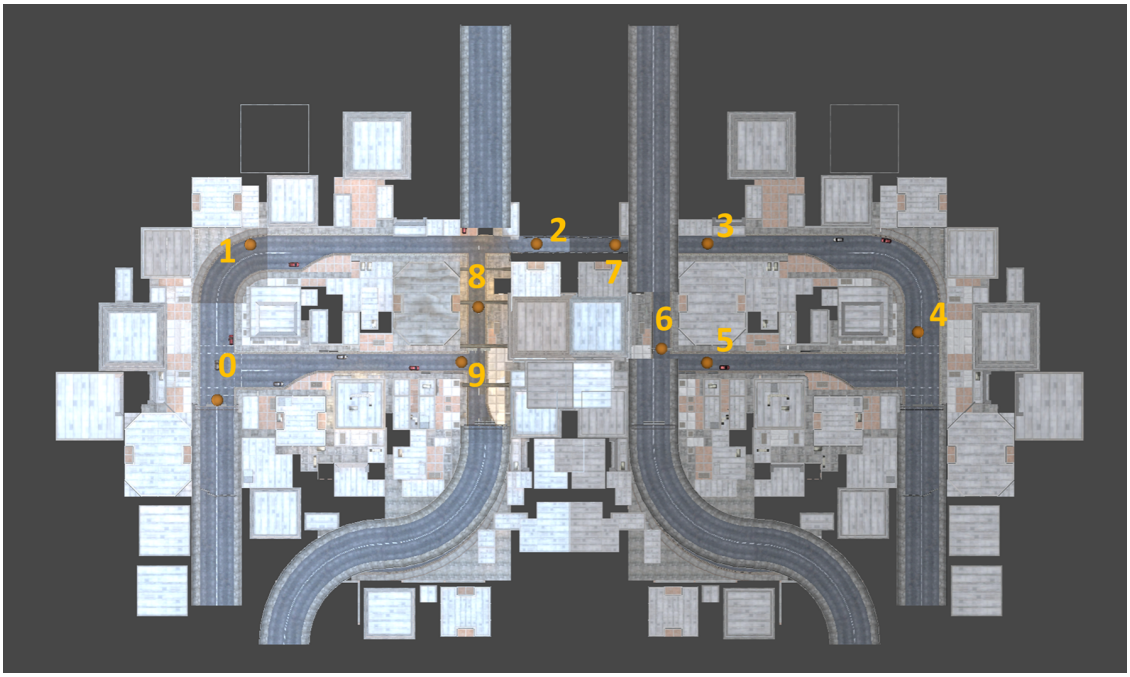

Figure 4: Available spawn positions. The spawn position **0** is for the final benchmark.

3 Simulation Configuration

To streamline the tuning process, the configuration file allows you to set spawn positions, enable or disable optional events, and configure sensor settings.

- **Event Deactivation:** To disable optional events, set the corresponding boolean in the `eventEnable` array field to `false`. The associated NPC vehicles will spawn at their final positions.
- **Spawn Position Selection:** You can choose from 10 spawn positions, as shown in Fig. 4. If a spawn point after an event is selected, that event will be automatically skipped.
- **Camera Configuration:** The resolution and FOV of cameras are adjustable, but the default settings are recommended for stable performance. Increasing resolution may significantly reduce simulation frame rate.
- **Ground Removal:** The depth camera detects the distance to objects in its field of view, including the road surface. Without additional processing, the road may be later included in the occupancy grid, which can interfere with navigation. If you're having trouble removing the road, you can enable automatic removal by setting `groundRemoval` to `true`. This will exclude the road from the depth images.

For more details of the configuration file, please refer to the `README.md` in the repository.

3.1 Grading

You can reach a maximum of 100 points in the following categories:

- Functionality and Performance (max. 50p)
  - Successfully working perception pipeline (max. 10p)

  - Successfully working path planning (max. 8p)
  - Successfully working trajectory planning (max. 8p)
  - Successfully avoiding other cars (max. 6p)
  - Successfully stopping/driving at street lights (max. 12p)
  - Time to complete the mission (max. 6p)
- Code and Architecture Quality (max. 30p)
  - Sensible separation in packages, nodes and functions (max. 10p)
  - Sensible separation in services, topics, etc (max. 10p)
  - Code quality (use in a group same coding style!), readability, and traceability (max. 10p)
- Written Summary (max. 20p)
  - Sound explanation of your architecture, model, and design choices (max. 15p)
  - Clear documentation who did what, which code is your own, which code is reused (max. 5p)
- Bonus Points (max. 10p)
  - Solving the problem without using semantic camera (max. 10p)

Very important: We will subtract up to 30p if your code does not build as explained in your readme or performes differently then documented.

3.2 Tips

Here are a couple of hints regarding the implementation. The hints are just suggestions, you are free so solve the task differently:

- Start by taking a look to the README.md files in the folder 'src' and the folder 'src/simulation' of the project folder of the course repository to get started.
- Generating point cloud from depth image: use *depth_image_proc* in http://wiki.ros.org/depth_image_proc.
- Generating occupancy Grid: use *Octomap* in http://wiki.ros.org/octomap.
- Merge semantic map with point cloud or occupancy grid *Octomap* in http://wiki.ros.org/octomap.
- **Please ping us in case you have any questions or if you get stuck in some subtasks.**