Exam Admission Requirements

Completing this assignment is mandatory to be eligible for the exam. Only the exam admission from last year remains valid for this year. To gain exam admission, you must complete the following tasks both in a simulation environment and on real robots. You need to book a meeting in the Moodle workspace for the lab date when you will present your simulation results and test your solution on the real robots. The simulation task must be completed before the lab date. In the lab, you will present your results, test your solution on the real robots, and, if time permits, make improvements or explore additional functionalities with the robots.

Technical Prerequisites

The technical prerequisites for this assignment are the same as those for previous assignments. You can use the Retina PCs prepared for the lab exercises. On presentation day, we will provide hardware to run your solution on real robots. Therefore, bring your prepared code on a USB stick or use a cloud service to access your code on the lab PCs. Alternatively, you can bring your own Laptop, but keep in mind you might have to switch for working with the real robot to the lab PCs.

Additionally, you need to modify a few files in the Turtlebot workspace to enable simulation and navigation in our lab environment. Please follow these steps before starting the tasks:

- Build the local Turtlebot workspace as demonstrated in ROS Introduction 1a.
- Download the zip folder additional_files.zip from the Moodle workspace to your ROS workspace.
- Place each item from the zip file in the corresponding folder of your Turtlebot workspace. The folder structure in the zip folder matches your local Turtlebot workspace, so you can directly extract and merge the folders to your ROS workspace.
- Build the workspace with the command: `colcon build -symlink-install`
- Start the simulation with the command:
  `ros2 launch turtlebot3_gazebo turtlebot3_rst_lab.launch.py`
- Start the navigation (for setting a goal in RViz) with the command:
  `ros2 launch turtlebot3_navigation2 navigation2_rst_lab.launch.py use_sim_time:=True`
- Set the robot pose in RViz to the initial position of the robot in the simulation. You can use the `2D Pose Estimate` tool in RViz to set the robot's initial position and orientation.
- Add the `2D Goal Pose` tool in RViz (click on the + button in the `Tools` panel) to set the goal position for the robot on the topic `/goal_pose`.

Motion Planning for Mobile Robots

Motion planning for mobile robots involves determining a sequence of movements that allow a robot to navigate from a starting point to a desired goal while avoiding obstacles and adhering to constraints. This process is crucial for autonomous navigation and involves two main components: **global path planning** and **local online trajectory planning**.

Global Path Planning Global path planning focuses on finding an optimal or feasible path from the robot's starting location to its goal location, considering the entire environment. This stage deals with high-level planning and is usually performed before the robot starts its motion. The key aspects of global path planning include:

Environment Representation The environment is represented using maps such as grid maps, topological maps, or occupancy grids. This representation includes the layout of obstacles and free spaces.

Pathfinding Algorithms Common algorithms used in global path planning include:

- **Dijkstra's Algorithm**: Guarantees the shortest path in a graph with non-negative weights.
- **A***: An extension of Dijkstra's algorithm that uses heuristics to speed up the search process.
- **Probabilistic Roadmaps (PRM)**: Constructs a graph of possible paths by randomly sampling the space and connecting these samples.
- **Rapidly-exploring Random Trees (RRT)**: Expands a tree rooted at the start position by randomly sampling the space and incrementally growing the tree towards the goal.

Path Optimization Ensuring the path is optimal in terms of length, time, energy consumption, or other criteria. This may involve smoothing the path or adjusting it to avoid unnecessary detours.

Local Online Trajectory Planning Local online trajectory planning, also known as local planning or real-time planning, deals with the robot's immediate movements and adjustments based on its current state and dynamic changes in the environment. This stage ensures the robot follows the global path while responding to unforeseen obstacles or changes. The key aspects include:

Reactive Navigation The robot continuously senses its environment using sensors such as LIDAR, cameras, or sonar, and makes real-time adjustments to its trajectory to avoid obstacles.

Local Path Planning Algorithms Common techniques include:

- **Dynamic Window Approach (DWA)**: Considers the robot's kinematic constraints and evaluates possible trajectories within a window of feasible velocities.
- **Vector Field Histograms (VFH)**: Uses a histogram grid to represent obstacle density around the robot and selects a direction with low obstacle density.
- **Timed Elastic Band (TEB) and Model Predictive Control (MPC)**: Optimizes the robot's control inputs over a future time horizon, considering both the robot's dynamics and the environment.

Integration of Global and Local Planning For effective navigation, both global path planning and local online trajectory planning need to be integrated seamlessly: The global path provides a high-level reference trajectory for the robot to follow. The local planner uses this path to guide its short-term decisions.A hierarchical approach is often adopted where the global planner updates the path at a lower frequency, and the local planner continuously refines this path at a higher frequency. The robot's position and environmental changes are continuously fed back to both the global and local planners to update the plans as needed.

The eBook `https://de.mathworks.com/campaigns/offers/next/getting-started-with-motion-planning-in-matlab-ebook.html` provides an overview on motion planning for mobile robots, autonomous vehicles and robot manipulators.

Timed Elastic Band Approach for Local Trajectory Planning

The Timed Elastic Band (TEB) approach is an advanced method for local trajectory planning in mobile robots. It optimizes the robot's trajectory in both spatial and temporal domains, considering dynamic constraints and the need to react to changes in the environment in real-time.

The TEB approach treats the planned trajectory as an elastic band that can stretch and bend to avoid obstacles and optimize motion. The key features of TEB include:

- **Elastic Band Representation**: The trajectory is represented as a series of discrete points (poses) connected by elastic constraints that can adjust their positions and timings.
- **Temporal Optimization**: Time intervals between poses are also optimized, allowing the robot to speed up or slow down as needed.
- **Kinodynamic Constraints**: The optimization process respects the robot's kinematic and dynamic constraints, ensuring feasible and safe trajectories.

The TEB approach involves the following steps:

- Initialization: A coarse initial trajectory is generated using a global path planner. This trajectory serves as the initial elastic band that will be refined.
- Optimization Objectives : The optimization process aims to minimize a cost function that typically includes the following components:
  - **Smoothness**: Ensures that the trajectory is smooth, avoiding abrupt changes in direction or speed.
  - **Obstacle Avoidance**: Penalizes trajectories that come too close to obstacles.
  - **Time Optimization**: Adjusts the time intervals to minimize travel time while respecting dynamic constraints.
  - **Kinematic Feasibility**: Ensures that the trajectory is feasible given the robot's kinematic constraints (e.g., maximum velocity, acceleration).
- Optimization Process: The TEB optimization is typically performed using nonlinear optimization techniques. The trajectory is iteratively adjusted to minimize the cost function while satisfying the constraints. This involves:
  - **Pose Adjustment**: Each pose in the trajectory is adjusted to reduce the overall cost.
  - **Time Adjustment**: Time intervals between poses are optimized to balance speed and safety.
  - **Constraint Handling**: Constraints such as obstacle avoidance and kinematic limits are enforced throughout the optimization process.
- Replanning and Real-time Adjustment: The TEB approach continuously replans the trajectory in real-time based on sensor inputs and changes in the environment. This allows the robot to react dynamically to moving obstacles and other unexpected events.

The TEB approach offers several advantages for local trajectory planning:

- **Flexibility**: The elastic band representation allows for flexible and adaptive trajectories.
- **Dynamic Adaptation**: Continuous optimization enables real-time adaptation to changes in the environment.
- **Smooth and Feasible Trajectories**: The optimization process ensures that trajectories are both smooth and kinematically feasible.
- **Integration with Global Planning**: TEB can be easily integrated with global path planning to provide a comprehensive navigation solution.

Navigation - Simulation

In previous labs, we explored various global path planning approaches and Timed Elastic Band (TEB) local planning. This lab will integrate these components to form a comprehensive navigation stack. You can reuse your code from previous labs to complete this lab.

The global planner, using RRT (Rapidly-exploring Random Tree) or PRM (Probabilistic Roadmap), generates a collision-free path from the start to the goal based on the map's information. The TEB planner creates a local plan that adheres to the robot's kinematic constraints, avoids obstacles, and optimizes travel time while following the global path.

Based on the provided MATLAB template, complete the following tasks in the simulation environment:

1) **Establish the communication between ROS and MATLAB**
   - Set up communication between ROS and MATLAB to enable data exchange.
2) **Initialize a publisher for robot velocity commands**
   - Create a publisher for the robot velocity commands on the topic `/cmd_vel`.
   - Set up a subscriber for the odometry data on the topic `/odom`.
3) **Initialize a goal subscriber**
   - Create a subscriber on the topic `/goal_pose`.
     Upon receiving a message, it should trigger a callback function that sets the global goal in a `PoseHandle` object.
     (Hint: Use the `PoseHandle.m` file from the Moodle workspace for implementing the `PoseHandle` class.)
   - Pass this object as an additional argument to the callback function.
4) **Load and visualize the Map**
   - Load the `rst_lab_cropped.pgm` map of the RST lab and convert it to a `binaryOccupancyMap`.
   - Visualize the map, the current robot pose, and the goal pose in MATLAB. Note, that the map represents only a part of the full lab environment.
   - Note that it is required to plot the goal pose in the correct coordinate system (check the helper function `goalHandle2goalPose`).

5) **Implement global planners (RRT and PRM)**

- Implement the RRT and PRM global planners using the corresponding MATLAB functions (`plannerPRM` and `plannerRRT`). The global planner should generate a path from the current robot pose to the goal pose.
- Visualize the generated path in MATLAB.

6) **Implement the TEB local planner**

- Optimize the planned global path using MATLAB's `optimizePath` function.
- Plot the optimized path alongside the original path within the RST map.
- Generate velocity commands based on the current robot pose (odometry), the optimized global path, and the map using MATLAB's `ControllerTEB` object.
- Publish the velocity commands on the `/cmd_vel` topic.
- If no feasible solution is found, replan the global path.
- Visualize the robot's current progress in MATLAB while it moves.

Navigation - Real Robot

This part is not updated for ROS2 yet. It will be updated before the mandatory assignment starts.

This part executes the navigation stack on a real robot. In case all previous tasks are solved, the navigation stack can be executed on the real robot only with slight changes and adapted launch files.

All lab PCs are already prepared with a running turtlebot workspace. You only have to start the correct launch files and the robot executes the commands you send from Matlab.

Instead of the simulation, the real robot is used. Therefore, it is required to access the Raspberry Pi of the robot via SSH and start the correct launch file. Furthermore, it is required to change some ROS-specific URIs and IPs as follows:

1) **IP settings lab PC**
   - Open a terminal on the lab PC.
   - Type `hostname -I` to get the IP address `<lab_pc_ip>` of the lab PC.
   - Type `cd` to change to the home directory.
   - Type `gedit .bashrc` to open the bashrc file.
   - Add `export ROS_MASTER_URI=http://<lab_pc_ip>:11311` to the end of the file.
   - Add `export ROS_IP=<lab_pc_ip>` to the end of the file.
   - Add `export TURTLEBOT3_MODEL=<model_name>` to the end of the file.
   - Type `source .bashrc` to apply the changes or start a new terminal
2) **Start ROS on the lab PC**
   - Open a terminal or new terminal tab on the lab PC.
   - Type `roscore` to start the ROS master.
3) **Access the Raspberry Pi of the robot via SSH.**
   - Open a terminal on the lab PC.
   - Connect to the Raspberry Pi of the robot via SSH with the command `ssh ubuntu@<robot_ip>`, where `<robot_ip>` is the IP address of the robot (label on the robot).
   - Type the password `turtlebot` to log in.
4) **Change the ROS-specific URIs and IPs.**

- In the terminal on the Raspberry Pi, type `nano .bashrc` to open the bashrc file.
- Find the line with `export ROS_MASTER_URI=http://<some_ip>:11311` and change `<some_ip>` to the IP address of the lab PC `<lab_pc_ip>`.
- Check if the line with `export ROS_HOSTNAME=<some_ip>` is set to the IP address of the robot label. If not inform the supervisor.

5) **Bringup the robot**

- In the terminal on the Raspberry Pi, type `roslaunch turtlebot3_bringup turtlebot3_robot.launch` to bring up the robot.
- If this command does not go further than `[INFO] [1718103720.725373, 0.000000]: Connecting to /dev/ttyACM0 at 115200 baud`, refer `https://emanual.robotis.com/docs/en/platform/turtlebot3/opencr_setup/#opencr-setup` and run the OpenCR startup and the Hardware Assembly setup. Make sure you set Noetic on the top page.

6) **Start the navigation stack**

- In a terminal on the lab PC, type `roslaunch turtlebot3_navigation turtlebot3_navigation_rst_lab_real.launch` to start the navigation stack. This launch file is only available on the lab PCs.
- Check if the robot is set correctly in RViz or use the 2D Pose Estimate button to set the robot's position.
- Start Matlab. For setting a goal use the 2D Nav Goal button in RViz after starting up the corresponding subscriber in Matlab.
- The same Matlab code as in the simulation can be used to send goals to the real robot.