Robot Planning and its Applications Project 2025/26

General Description

The project consists of different scenarios, which will challenge the students to apply the algorithms and the theoretical knowledge learned during lectures to practical cases.

The project will focus on implementing algorithms for path planning and target planning. The students will have to design and implement one or more nodes of ROS in C++.

Briefly the three projects are:

- Coordinated evacuation, which will challenge the multi-agent coordination;
- Target rescue, which will challenge the design of algorithms to maximize the revenue;
- Pursuit evader, which will challenge the design of algorithms scenarios that change dynamically.

Each group of students, composed at maximum of 3 students

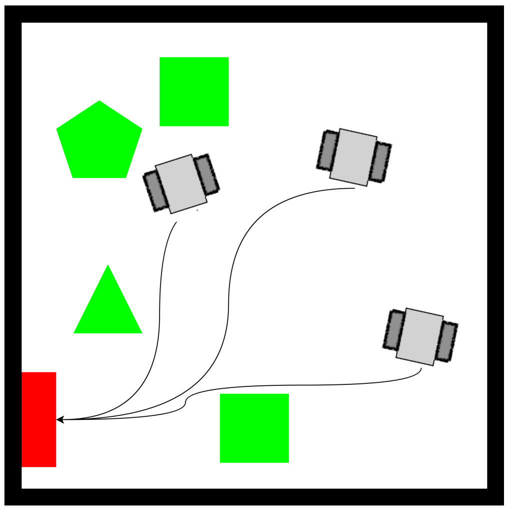
$$(|\text{group}| \in [1, 3])$$

will have to choose 1 of the 3 scenario and develop a solution for it. If you want to implement more than one, feel free to do it. The score for each scenario is the same.

General assumptions are:

- The robots move at constant speed, i.e., use Dubins manoeuvers;
- Touching the borders of the map and/or the obstacle will decrease the point obtained by completely the task successfully.
- The robots (except for the pursuer in the evasion scenario) must reach the center of the gate and with the angle provided in the topic.
- Obstacles may have different shapes, so do not assume they are regular polygons.

A) Coordinate Evacuation



The project is about designing a coordinate evacuation of a site cluttered by several obstacles and robots.

The robots will have to reach a gate in the minimum amount of time avoiding the obstacles *and* avoiding collisions with other robots.

Evaluation

You will gain points for minimizing:

- time to construct the roadmap, i.e., a graph of the map;
- time to plan for the solution of the problem;
- time to execute the task, as the time from the first movement of the first robot to the final movement of the final robot;

You will lose points for:

- robots exiting bounderies and/or touching obstacles;
- robots colliding between each other.

Assumptions

- Robots are initially spawned at desired locations (i.e. defined in loco_planning/launch/multiple_robots.launch)
- Since robots move at constant velocity, they *must* not stop.

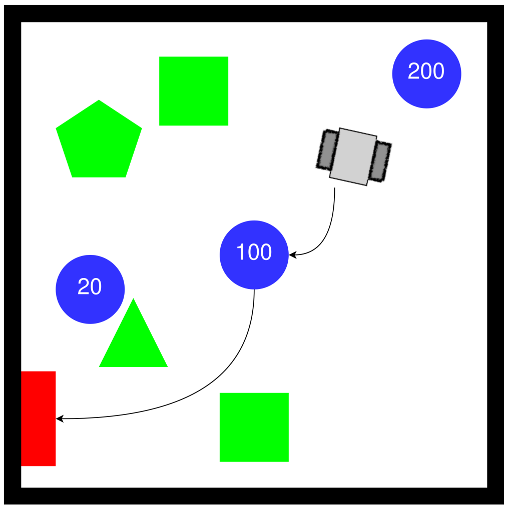
Steps

1. The students will have to create a roadmap that enables the robot to move in minimum time and with guaranteed clearance from the obstacles.
2. The roadmap will have to be decorated with information regarding the nodes and the edges that cannot be occupied at the same time by two robots, **to avoid a collision**.
3. The students will set up an automatic strategy (e.g., optimisation based, heuristic) to decide the motion of the robots so that they evacuate the area without creating collisions. Any algorithm for finding the strategy to move the agents from the given initial nodes towards the escape gate can be used.

Simplification

The student can make the hypothesis of a synchronous behaviour: i.e., the system evolves in a sequence of states and each state is characterised by having the robot located in one of the nodes of the roadmap. In this way, every step of the plan is ended when all robots have reached a location and no robot is allowed to move to the next location before a step completes. Alternatively, the students can also release this assumption and allow the robot to move freely (asynchronous behaviour).

B) Target Rescue



The project consists in rescuing some victims using a single robot and exit from the gate. While the order in which the victims are rescued does not matter, they have to be rescued within a time limit. Each victim is associated with a value and the higher the total value of the rescued victims, the better it is. Failing to reach the goal (gate) means failing the task.

Evaluation

You will gain points by:

- rescuing victims;
- minimizing the time it takes to construct the map, plan, and execute.

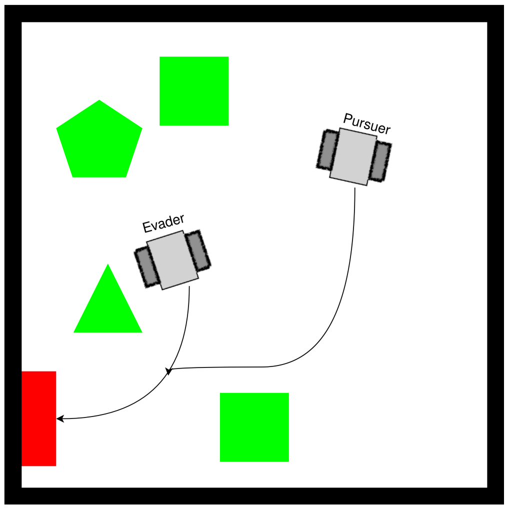
You will lose points for:

- touching the obstacles/borders
- not reaching the gate, which will give 0 points despite the rescued victims.

Assumptions

- While victims are circles of radius 0.5m, the robot must pass though the center of the circle in order to rescue them.
- The robot does not need to stop to rescue e victim.
- The victims are initially deployed in random positions.
- Since robots move at constant velocity, they *must not* stop or slow down.

C) Pursuit-Evasion



The project is about capturing an evader robot using a pursuer robot. Both the pursuer and the evader move in an environment characterized by the presence of obstacles. The map can have one or more exit points and the project has different levels of difficulty, as detailed below. The evader is caught when it reaches a location of the map currently occupied by the pursuer. The students will have to code the behavior of both the pursuer and the evader.

Evaluation

You will gain points for minimizing:

- Success in catching the evader.
- time to construct the roadmap, i.e., a graph of the map;
- time to plan for the solution of the problem;
- time to execute the task, as the time from the first movement of the first robot to the time of catching the evader;

You will lose points for:

- robots exiting bounderies and/or touching obstacles;
- not catching the evader.

Also the points will be given based on the difficulty of the scenario: catching the evader in an easier scenario will give less points than catching it in a more difficult one.

Assumptions

- The evader *is controller by the computer* but it HAS to move following the same roadmap computed by the pursuer. The students will have to publish said roadmap on a topic so that it can be used by the evader algorithm to compute the path.
- You can assume that once the evader takes a edge of the graph, it has to complete the movement and cannot back-off or stop.
- Since robots move at constant velocity, they *must* not stop or slow down.
- The pursuer does not know the evader's future path, hence you cannot read from the evader topic to get the path it will follow.
- Once the evader has chosen a gate to reach, it will follow the shortest path to the gate without considering other smarter policies to avoid the pursuer(s).

Level of complexity

The behaviour of the evader is organised in increasing levels of complexity. If you choose this project, you must account for the following 3 behaviours:

1. There is only one exit and the evader reaches it in minimum time.
2. There are two exits and the evader randomically chooses one of the two gates before moving.
3. There are two gates and the evader chooses one based on the position of the pursuer.

Steps

The students will have to create a roadmap that enables the robots to move in minimum time and/or with guaranteed clearance from the obstacles. The students will setup a strategy (e.g., optimisation-based, heuristic) to decide the motion of the pursuer so that it reaches evader in minimum time. The strategy can achieve different goals depending on the level of behavioural complexity considered for the evader. The minimum requirement is to fulfill the first level of complexity.

Code Structure

The code for the simulation is available at: https://github.com/idra-lab/loco_nav.

The structure of the code is the following:

- `map_pkg`: contains
    - spawning random obstacles;
    - spawning random victims;
    - spawning random gates, only along the borders of the map;
    - spawning the borders;

  This package publishes the following topics:

    - `/obstacles`: contains the obstacles that are present on the map. They may be only cylinders or boxes. In the former, the center (x, y) in $z=0$ and the radius are provided, in the latter, the center (x,y) in $z=0$ and the lengths of the borders are provided.
    - `/map_borders`:
    - `/victims`: contains information for the victims, which are thought as circles of radius 0.5m.
    - `/gate_position`: contains the position and orientation of the gate.

  This package can be configured using the `map_config.yaml` file inside of `map_pkg/config`. Also check the launch file to have more information on the possible parameters to pass. It also contains some pre-defined worlds like `hexagon.world` and `labyrinth.world`.
- `obstacles_msgs`: contains the interfaces used to publish the obstacles and the victims.
- `limo_description`: it contains the xacro description of the robot.
- `loco_planning`: it contains the launch file to start the simulation in Gazebo and the launch files to start the ROS Nav environment for the LIMO.
- `docker`: contains the DockerFile to generate the docker image
- `limo_hardware_interface`: contains the driver to run on the real robot.

You are free to change the parameters in every of the nodes, but you should carefully explain why you chose to and what benefit it provided.

Remember that each LIMO will have its own namespace so the topics that are dependent on the LIMO.

- The obstacles will be available at the topic /obstacles as a message of type:

  ObstacleArrayMsg.msg:

  ```
  #Message that contains a list of polygon shaped obstacles.
  std_msgs/Header header
  obstacles_msgs/ObstacleMsg[] obstacles
  And ObstacleMsg.msg:
  std_msgs/Header header
  ```

  ObstacleMsg.msg:

  ```
  std_msgs/Header header
  # Obstacle footprint (polygon descriptions)
  geometry_msgs/Polygon polygon
  # Specify the radius for circular/point obstacles
  float64 radius
  ```

- The walls of the map will be available in the topic /map_borders of type geometry_msgs/msg/Polygon
- The gate position will be available in the topic /gate_position of type geometry_msgs/msg/Pose. In the topic you will find the position of the centre of the gate.
- The robot data will be available in the topics:

  ```
  /limo#/scan
  /limo#/odom
  /limo#/joint_states
  /limo#/ref
  ```

  Where limo# is the namespace of the specific robot (limo0, limo1...). The information about the location in real time of each robot can be retrieved subscribing to the /limo#/odom topic.

How to develop the project

So what is missing? The course is called Robot Planing and its Applications, so all the planning part is missing.

You should write C++ (standard 17) nodes that allow the robots to complete the selected scenario. You need to implement **both** a combinatorial and a sampling based strategy and compare them. To summarize some steps:

- Build the roadmap by reading the borders from `/map_borders` and the obstacles from `/obstacles`.
- Create a node that is used for path planning using Dubins. You could reimplement in C++ the class PlannerBase() in planner_base.py and inherit from that to specialize for your planners (e.g have a look to planner_rrt.py) or insert an action client inside plan_path() to ask for the plan to your node.
- Once you have the roadmap and a path planner, you should compute the best path for the robots to complete their tasks.

- Once done you can make the robots move by:
  - publishing to the reference topic /limo#/ref (exploiting the controller of loco_nav (recommended).
  - directly commanding the robots using the topic /limo#/cmd_vel and sending the linear and angular velocities (you need to implement the controller yourself)
  - using the action server move_base_msgs/MoveBaseAction. You send a goal containing a target pose, and `move_base` handles planning + control
    You can keep track of the progress in motion using the /move_base/feedback of the action server (current distance to goal and speed).

Delivery

What you should provide

- A PDF report describing the two approaches followed, stating all the assumptions adopted and the reason why you decided to follow said approaches and the difficulties encountered. You should also, but it's not mandatory, include possible results that you have obtained, for example, statistics on the times it takes to compute a solution and/or the length of the solutions it finds. Any information that you think may be interesting regarding the project should be included.
- An archive containing all the software to allow reproducing the approach written in the report.

When you should provide the what

You should deliver the report and the code at least **1 week** before the date of the exam.

Date of exam	Delivery date
TBD	

How you should provide the what

As said, this is the repository containing the code of the simulation and that is updated to fix bug and/or to add new features.

The best thing would be for each group to create a repository with your planning package inside the loco_nav folder so that you can pull changes from the main repo if new fixes or features are added.

In any case, for the delivery you will have to provide an archive containing the code. Obviously no pre-compiled files nor executable will be taken into consideration.

You should send the archive and the pdf to both Prof. Palopoli and Prof. Michele Focchi in an email and wait for my ACK on the reception of the email. Also, in the email you should state the group components, and a name for the group (if you have chosen one).

P.S.

Be warned that the code and the report will be run for plagiarism, and if found to be plagiarised you will face the legal and ethical consequences of such.