UVIC
UNIVERSITAT DE VIC
UNIVERSITAT CENTRAL
DE CATALUNYA

eurecat
Centre Tecnològic de Catalunya

Master on Robotics

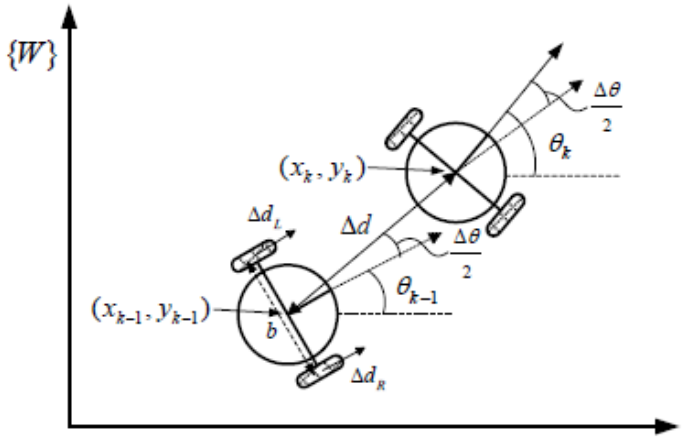
Postgraduate in Industrial Robotics
Postgraduate in Mobile Robotics

Year 2018-2019

Module VII: Ground Robots

Path Planning and collision avoidance

Assignment



Carlos Rizzo (Eurecat)
carlos.rizzo@eurecat.org

Assignment

Objective: to understand separately a global planner, a local planner, and to integrate them in one solution.

- Based on the theoretical material provided in class, and a series of python scripts, the goal is to create a robotics navigation solution in which a robot should be able to follow a global plan while reacting to unmapped obstacles.

- Read the whole assignment before starting to do anything.

- Exercise 3 is a suggestion on the integration. You can also make your own implementation based on the provided scripts, but over the requested map in 3.1.

Assignment

Provided material:

- Python scripts:
  - dijkstra global planner (dijkstra.py)
  - dynamic window approach local planner (dynamic_window_approach.py)
  - A star (A*) global planner (a_star.py). **Only for optional bonus question.**

- Reference:

  D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance," in IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23-33, March 1997.

Prerrequisites:

- Install python (scripts checked in version 2.7.15)

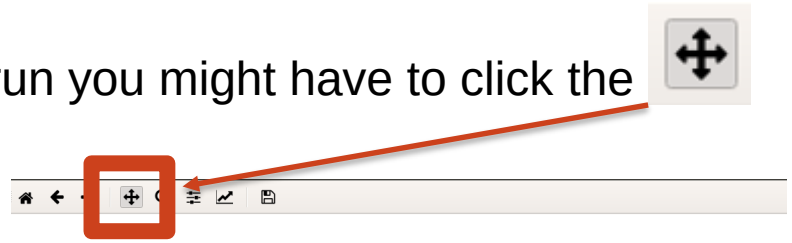
- To run the scripts, type in a terminal (where the scripts are located): python script.py (e.g. python dijkstra.py)

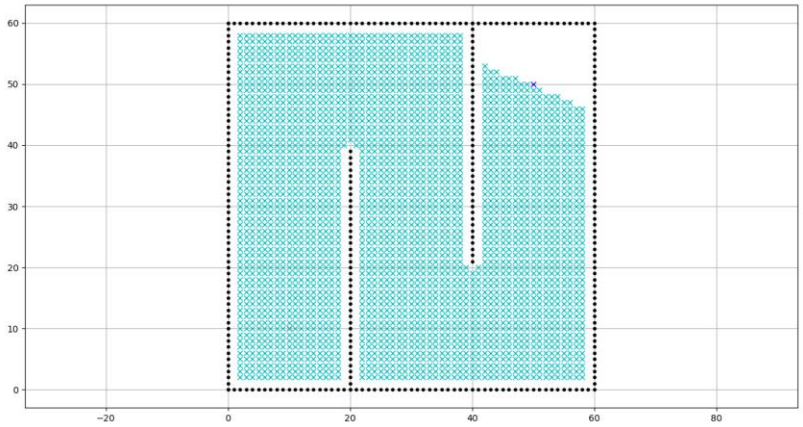
Assignment

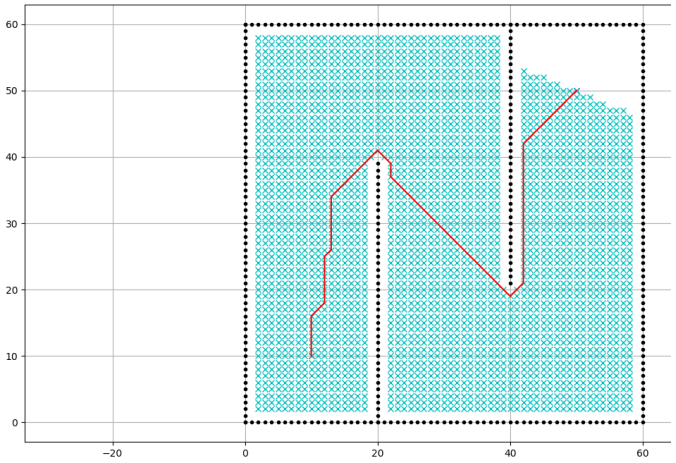
Exercise 1: dijkstra global planner

1.1 run the code (python dijkstra.py)

To visualize the final path, once the script has run you might have to click the button, and then click again on the plot







Assignment

Exercise 1: dijkstra global planner

1.2 Playing with the code:

- Locate start and goal. Change them. Run the script.
- Locate the grid size. Change it. Run the script.
- Locate the robot size. Change it. Run the script.
- Extract the path. E.g. Print the path in the screen (x-y positions)
- Locate the obstacles. Add some obstacles inside the map, at coordinates

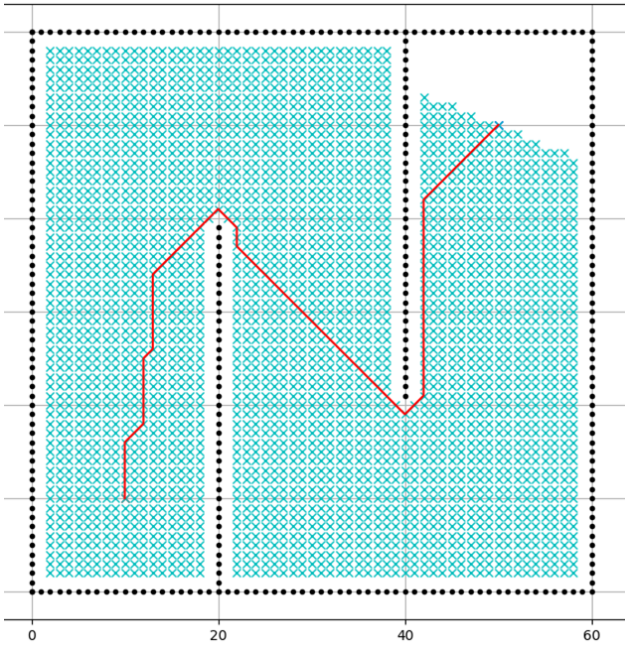
X position	Y position
10	15
16	37
25	34
31	28
42	30

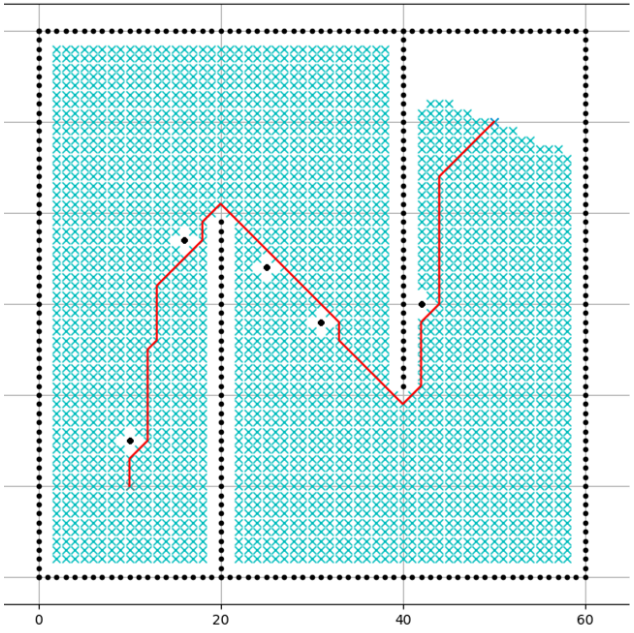
Assignment

Exercise 1: dijkstra global planner

1.2 Playing with the code:

- Locate the obstacles. Add some obstacles inside the map, at coordinates. Did you noticed that the path changed?





Assignment

Exercise 1: dijkstra global planner

1.3 Understanding the code:

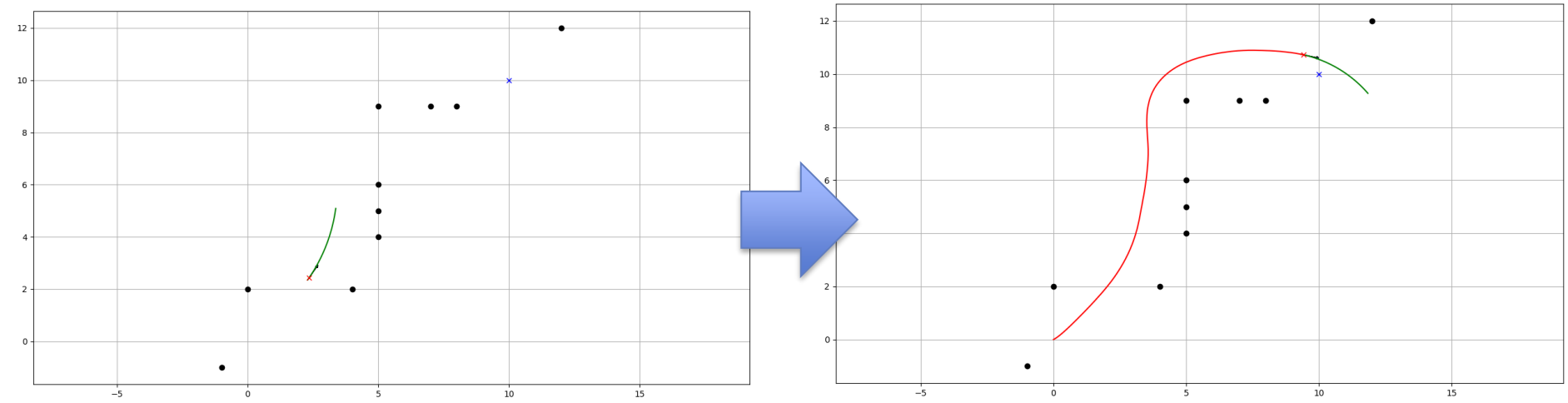
- Explain with your own words the algorithm in the code

- OPTIONAL: Run a_star.py and repeat exercises 1.1 and 1.2. Compare it with Dijkstra. Is it possible to easily convert the a_start script to dijkstra? Implement it. Hint: remember that dijkstra lacks the heuristic factor.

Assignment

Exercise 2: dynamic window approach local planner

2.1 run the code (python dynamic_window_approach.py)



Assignment

Exercise 2: dynamic window approach local planner

2.2 Playing with the code:

- Locate start and goal. Change them. Run the script.
- Locate the obstacles. Add more obstacles inside the map. Run the script.
- Change the DWA parameters. Following a scientific methodology, it is strongly suggested to vary only one parameter at a time.

```
self.max_speed = 1.0  # [m/s]
self.min_speed = -0.5  # [m/s]
self.max_yawrate = 40.0 * math.pi / 180.0  # [rad/s]
self.max_accel = 0.2  # [m/ss]
self.max_dyawrate = 40.0 * math.pi / 180.0  # [rad/ss]
self.v_reso = 0.01  # [m/s]
self.yawrate_reso = 0.1 * math.pi / 180.0  # [rad/s]
self.dt = 0.1  # [s]
self.predict_time = 3.0  # [s]
self.to_goal_cost_gain = 1.0
self.speed_cost_gain = 1.0
self.robot_radius = 1.0  # [m]
```

- Analyze the effect of each parameter over the robot's behavior

Assignment

Exercise 2: dynamic window approach local planner

2.3 Understanding the code:

- Explain with your own words the algorithm in the code. It is recommended to read [D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance," in IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23-33, March 1997]

- Locate the objective function in the code. Is it the same as in the original paper?

4.2 Maximizing the Objective Function

After having determined the resulting search space V_r a velocity is selected from V_r. In order to incorporate the criteria *target heading*, *clearance*, and *velocity*, the maximum of the objective function

$$G(v,\omega) = \sigma(\alpha \cdot \text{heading}(v,\omega) + \beta \cdot \text{dist}(v,\omega) + \gamma \cdot \text{velocity}(v,\omega))$$

is computed over V_r. This is done by discretization of the resulting search space.

Assignment

Exercise 2: dynamic window approach local planner

2.3 Understanding the code:

- OPTIONAL bonus question. Read the ROS code of the dynamic window approach at

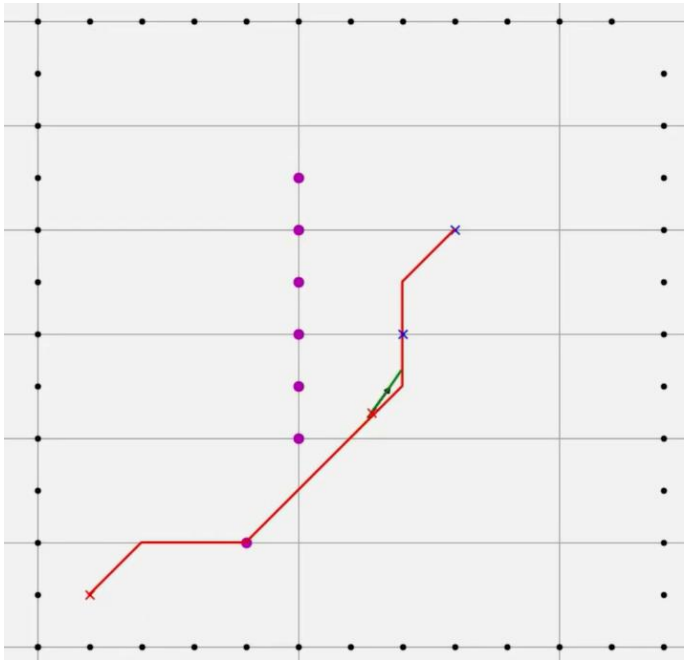
https://github.com/ros-planning/navigation/tree/melodic-devel/dwa_local_planner/src

And try to understand its implementation.

Assignment

Exercise 3: global + local planner integration

The goal of this exercise is to integrate both algorithms into one. It is recommended to integrate dynamic_window_approach into dijkstra.

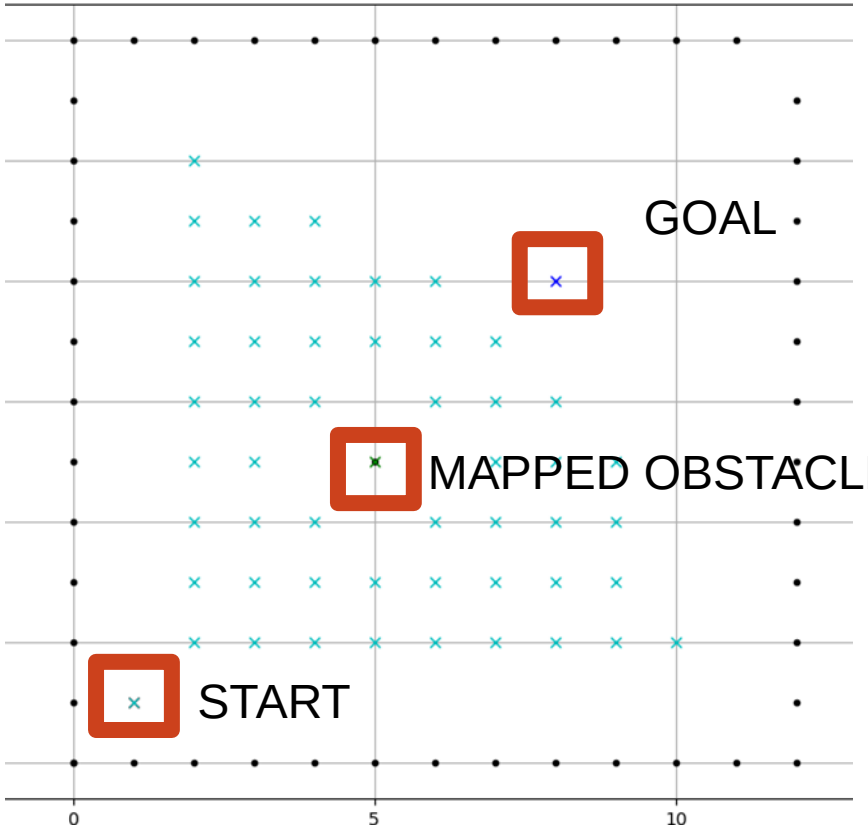


Assignment

Exercise 3: global + local planner integration

3.1 Before integrating the DWA planner, create/modify an environment in dijkstra.py:

- Create/modify a map of 12 x 12
- The start position is (1,1)
- The goal position is (8,8)
- Only one obstacle (besides the walls) will be visible to dijkstra. Locate it at position (5,5), as shown in the following figure.
- More unmapped goals will be added later

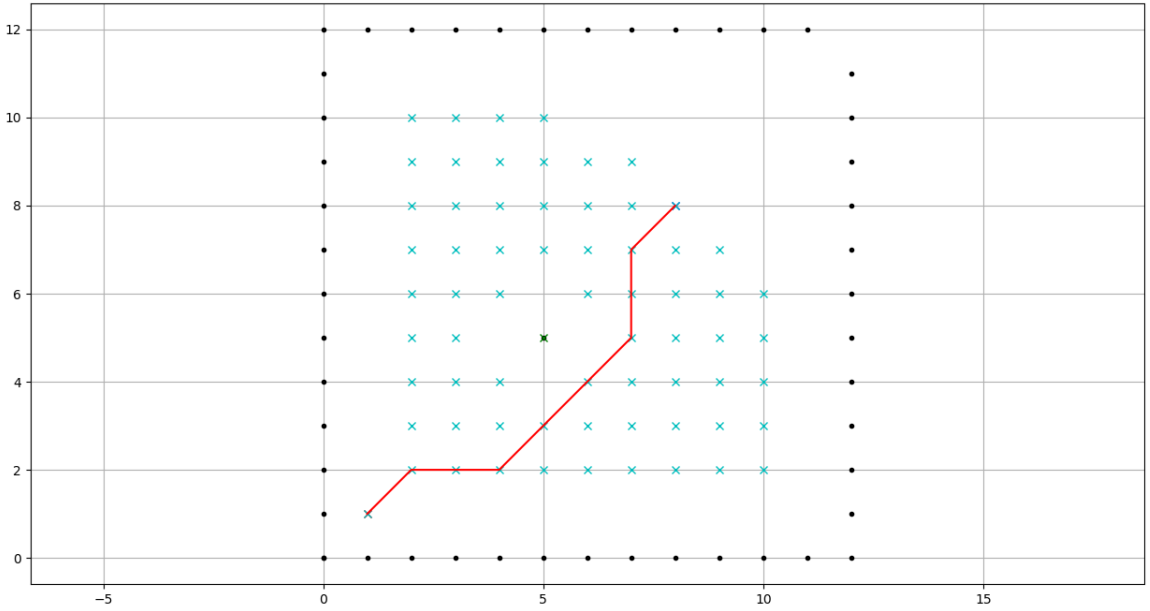


Assignment

Exercise 3: global + local planner integration

3.2 Calculate the global path

If 3.1 was well implemented, once you run dijkstra over that environment the path should look like this. Print the path's coordinates.



Assignment

Exercise 3: global + local planner integration

3.3 Integrate the DWA

Integrate the code of DWA into your modified Dijkstra with the new environment (copy + paste will work, taking into account some considerations. E.g. only one main function, and so on).

We have two layers of obstacles: the mapped obstacles (only one obstacle at (5,5) , integrated in the previous step), and the unmapped obstacles (e.g. persons). Lets add these unmapped obstacles. Remember, Dijkstra CAN'T see this obstacles

X position	Y position
4	2
5	4
5	5
5	6
5	7
5	8
5	9

Assignment

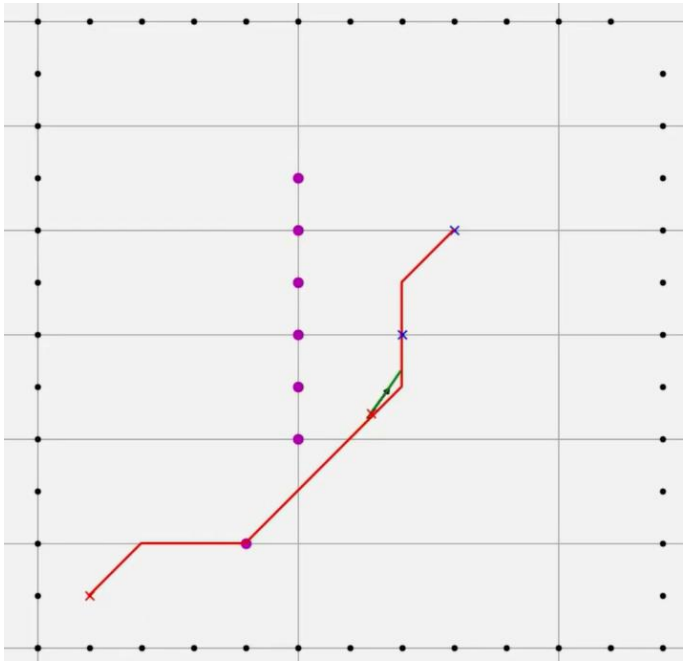
Exercise 3: global + local planner integration

3.3 Integrate the DWA

The unmapped obstacles should look like the ones in magenta.

Important: see that at (4,2) we have an unmapped obstacle over the trajectory.

Your implementation should be able to avoid this obstacle while following a path.



Assignment

Exercise 3: global + local planner integration

3.3 Integrate the DWA

Now, to integrate the DWA with the calculated path, the strategy is not to provide the final goal (8,8), but a series of sub-goals: the path calculated by dijkstra.

You will need to implement a sub-goal manager in a form of a loop. This loop will assign the coordinates of the dijkstra path, one by one, to the DWA function. Assign one, and once the robot has reached that sub-goal, assign the next one… until the goal is reached.

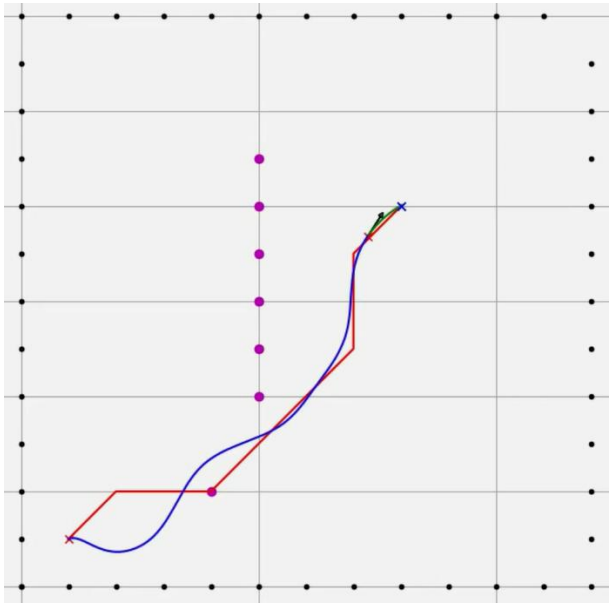
You can play with the trajectory resolution (give the DWA only some sub-goals, not all of them). You can also play with the DWA parameters (e.g. change the simulation time).

Assignment

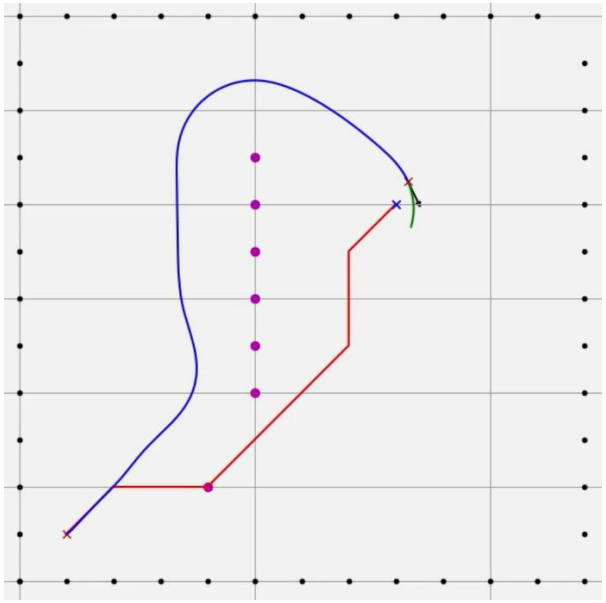
Exercise 3: global + local planner integration

3.3 Integrate the DWA

At last, implement a variable that controls to follow or not the path (e.g. bool follow_dijkstra). If true, it should follow the path. If false, the loop should provide the algorithm only the final goal. See the desired behavior in the attached videos.



follow_dijkstra = true



follow_dijkstra = false

UVIC
UNIVERSITAT DE VIC
UNIVERSITAT CENTRAL
DE CATALUNYA

eurecat
Centre Tecnològic de Catalunya

Master on Robotics

Postgraduate in Industrial Robotics
Postgraduate in Mobile Robotics

Year 2018-2019

Carlos Rizzo (Eurecat)
carlos.rizzo@eurecat.org