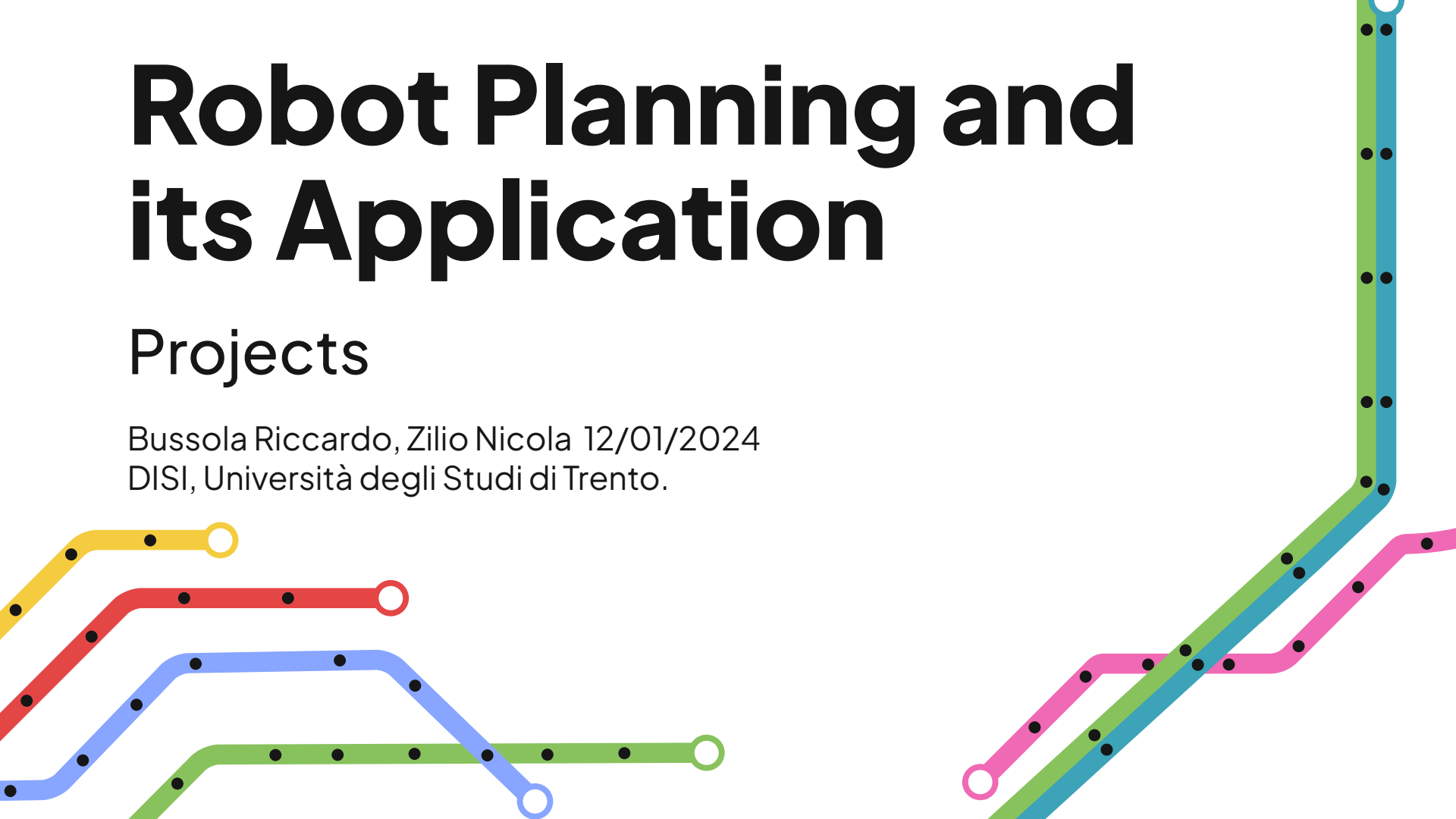
Robot Planning and its Application

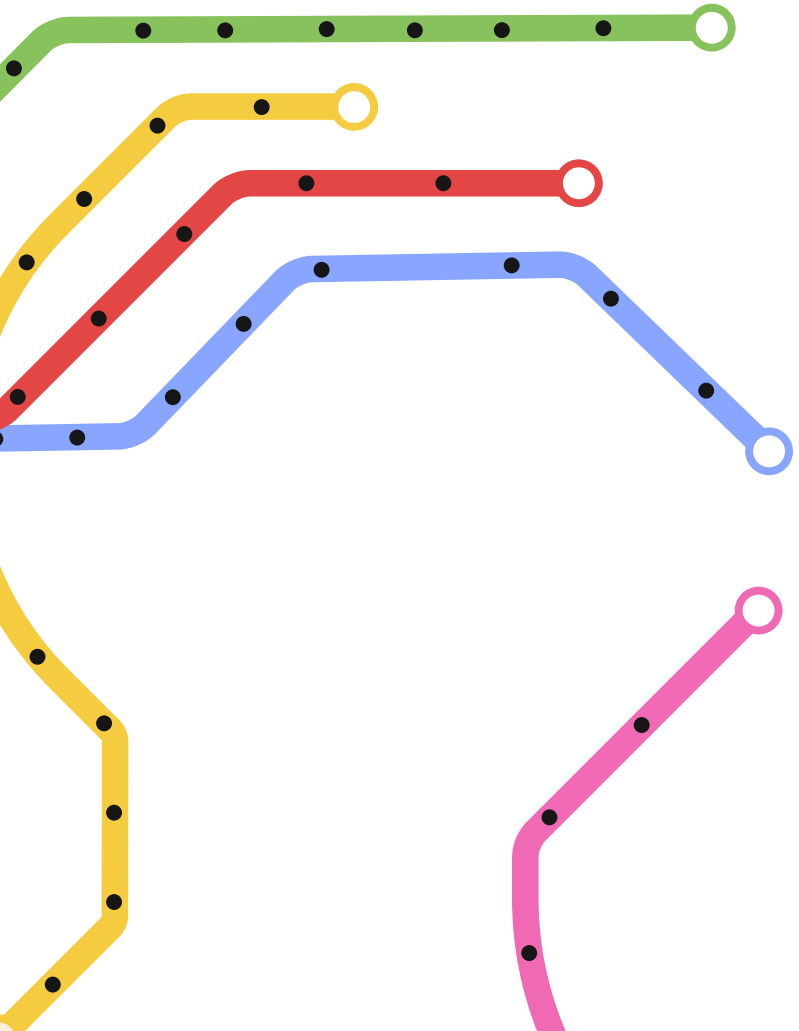
Projects

Bussola Riccardo, Zilio Nicola 12/01/2024
DISI, Università degli Studi di Trento.



Overview

Coordinated Evacuation	
Path-planning framework	The architecture of our solution
Grid-Based Approach	A grid-based approach for multi agent evacuation
Voronoi-Based Approach	A Voronoi-based approach for multi agent evacuation
RTT*-Based Approach	A RTT*-based approach for multi agent evacuation
Victims Rescue	
RTT*-Based Solutions and Results	RTT*-based solution for the victims rescue problem



Coordinated Evacuation

Coordinated Evacuation

Goal

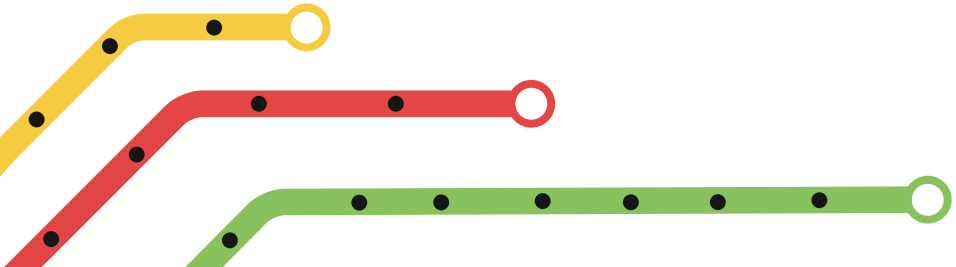
Coordinate the **evacuation** of the room without **collision** and with a **continuous** robot **motion**

Metrics

Time to **plan** the mission
Time to **evacuate** the room
Collisions

Algorithms

Voronoi based **Dijkstra** search
RRT* with **Dubins** paths

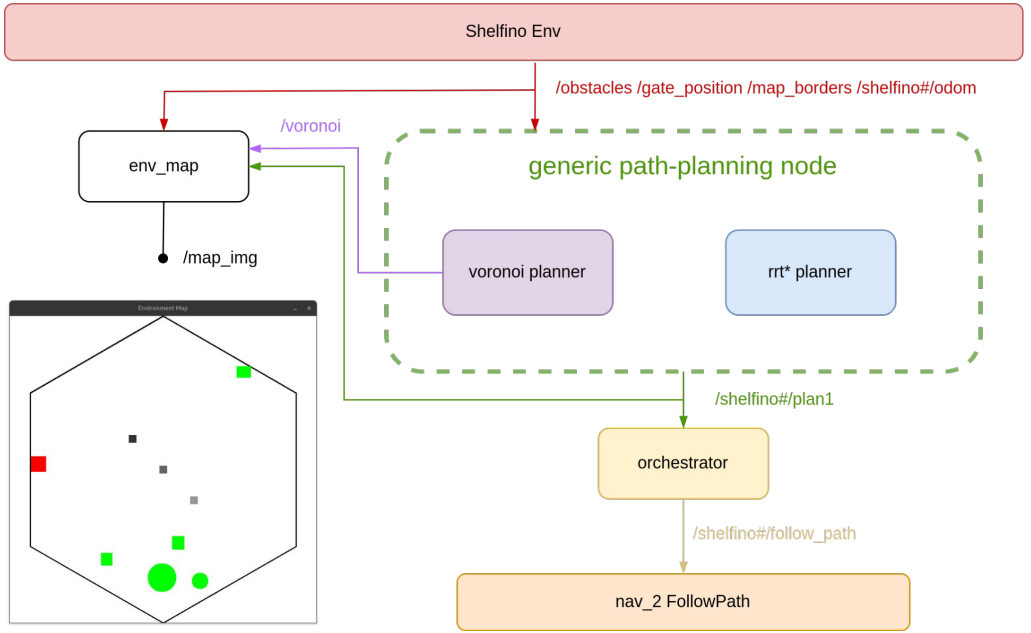


Path-planning framework

Components **reusability**

Fasten the **development** of new solutions.

Provides general **collision avoidance** capabilities

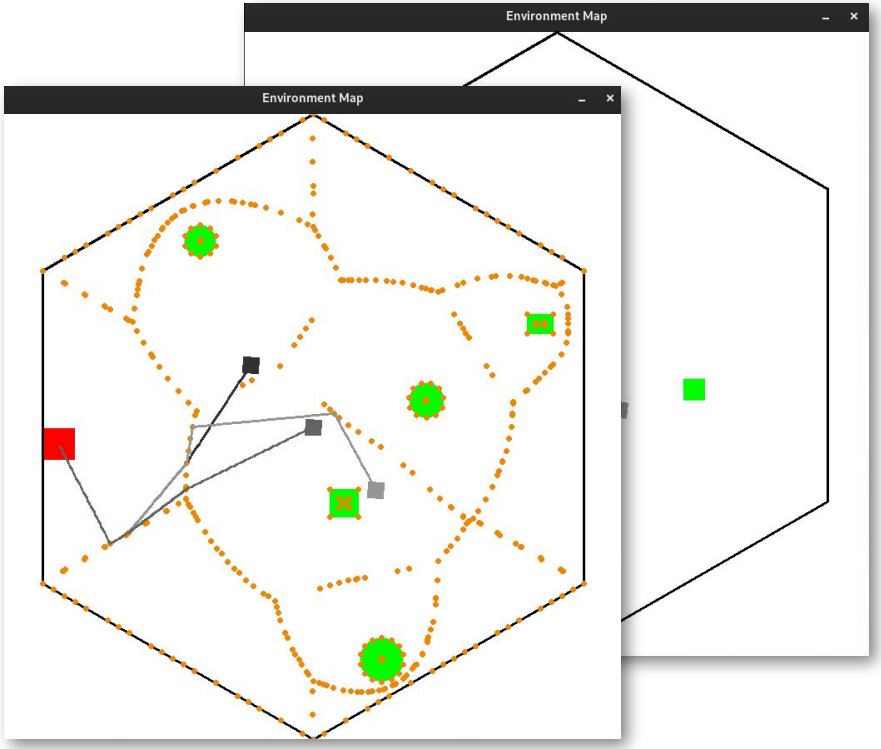


Environment map

A **visual tool** based on OpenCV to display more clearly the **problem**

It supports the visualization of gate, obstacles, map, shelfinos, paths and voronoi diagram

Provide also the topic with the computed image



Orchestrator

Manages the execution of the computed paths

Once received all of them, it compute the **path delay** to add in order to **avoid collisions** between robots

Heavily inspired to X*

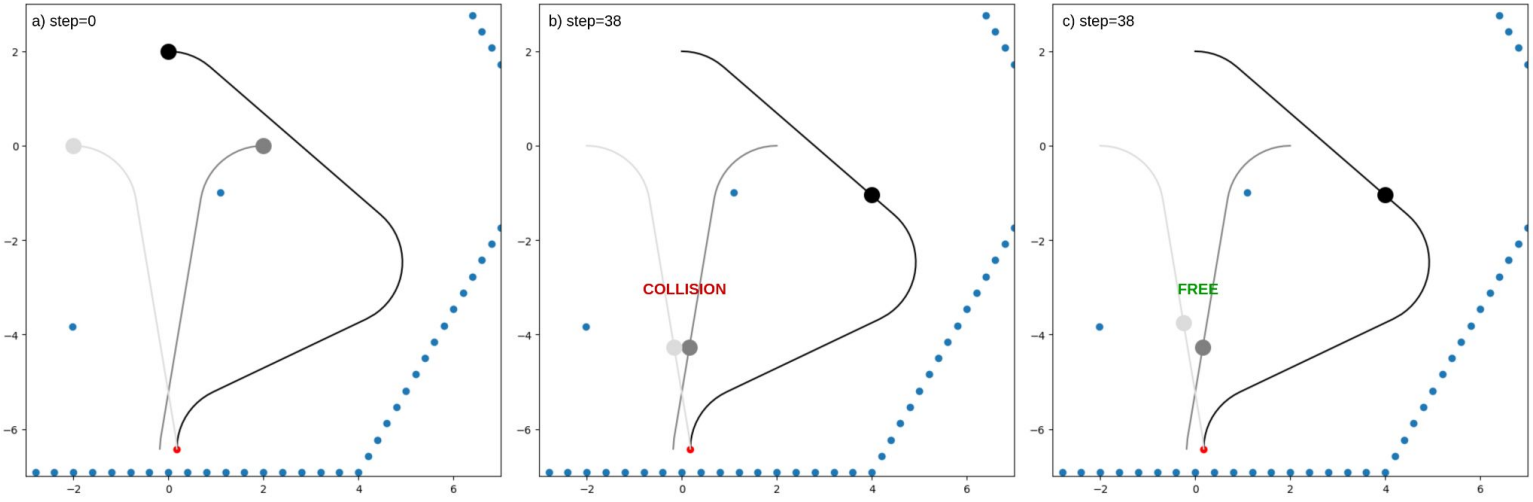
It assumes the **constant** motion **velocity** and the same step **discretization distance**

```
delay0, delay1, delay2 = 0
while no_collisions:
      path0.fill_head(delay0)
      path1.fill_head(delay1)
      path2.fill_head(delay2)
      collision_case =
calculate_distance(path0,path1,path2)
      switch collision_case:
      case path0_path1:
         if path0.size() < path1.size():
                  delay0 += delay_const
            else:
                  delay1 += delay_const
      ... same for case
          path0_path2, path1_path2

      case no_collision:

convert_delay(delay0,delay1,delay2)
```

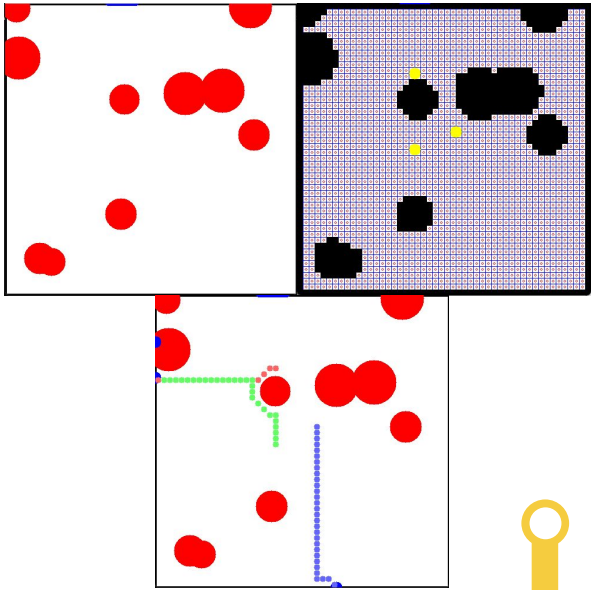
Orchestrator – CA



Grid Based Approach

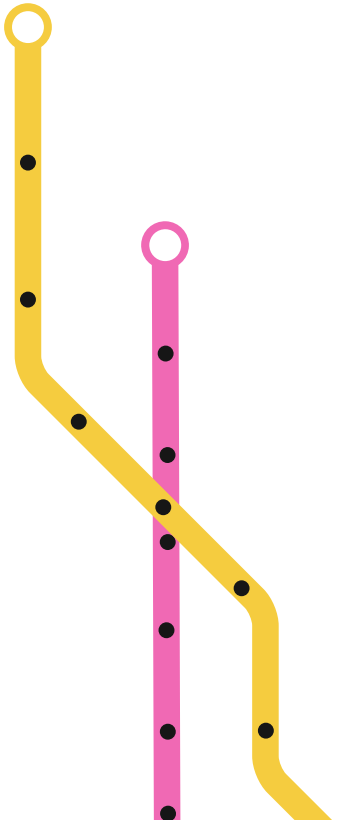
The first approach consists in:

- Building the map
- Constructing the **occupancy grid**
- Constructing the graph
- Apply **A*** to compute the path for the single robot
- Iterate through paths in order to find **collisions** and apply **priority mechanism**



Pros & Cons

Grid-Based



Pros

Good capacity to avoid obstacles

Good movement precision if grid is divided with enough size

Cons

Creating the graph may be difficult

A* path may be complex to approximate with Dubinns maneuvers

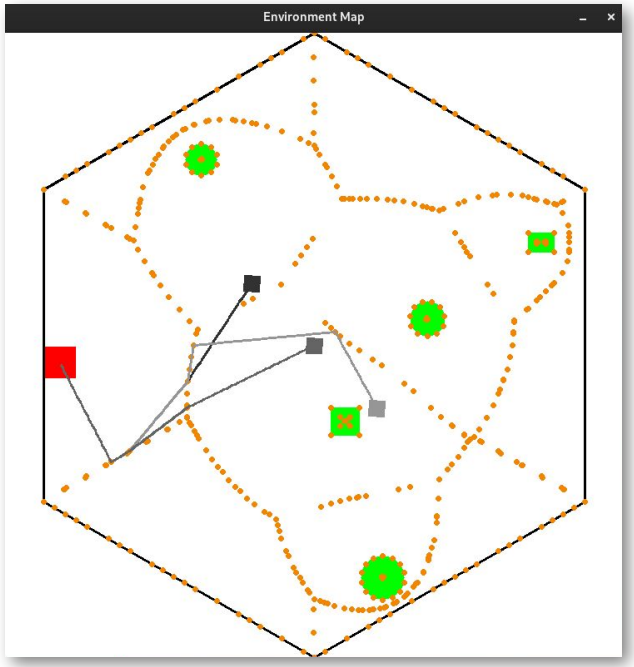
Voronoi–Dijkstra

Combinatorial planning

Maximize **safety**

Unique roadmap to **query multiple times**

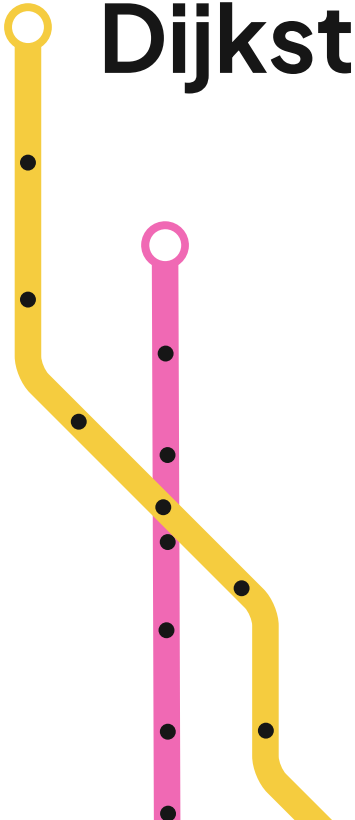
Suboptimal in lengths, optima with respect to the roadmap



```
(reverse-i-search)`voro': ros2 run path_planning voronoi_planner
```

Pros & Cons

Voronoi Dijkstra



Pros

Maximum clearance

If a solution exist, always founded

Dijkstra compute optimal roadmap path

Cons

Boost C++ library int constraint

KDTree not implemented as needed

Does not consider orientation,
NO Dubins

SLOW

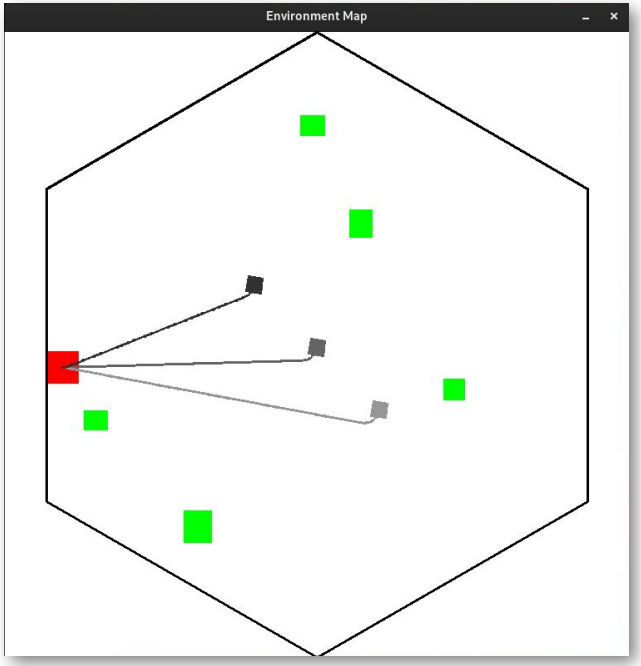
RRT*–Dubins

Sampling planning

Developed with Dubins built-in

Randomic, not always found the path

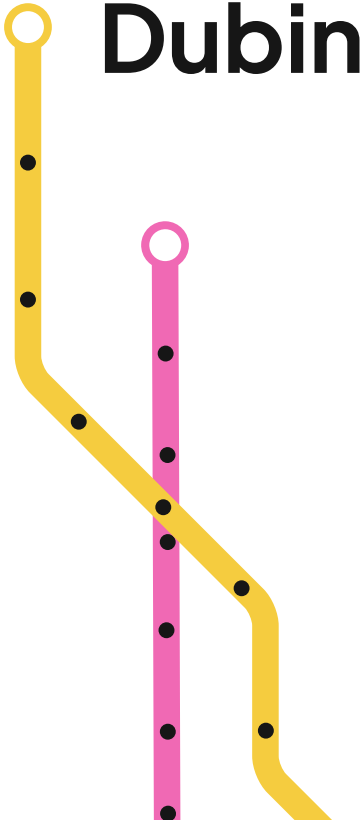
Asymptotically suboptimal




Jan 11 13:08
root@archlinux-desktop: ~/ros2_ws
(failed reverse-i-search)`rrr': ros2 run path_planning rrt_star_planner
[0] 0:python3- 1:bash*
"archlinux-desktop" 12:08 11-Jan-24

Pros & Cons

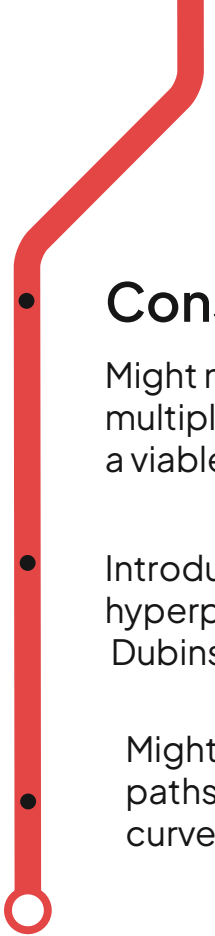
RRT*
Dubins



Pros

Dubins path planning

EXTREMELY FAST!

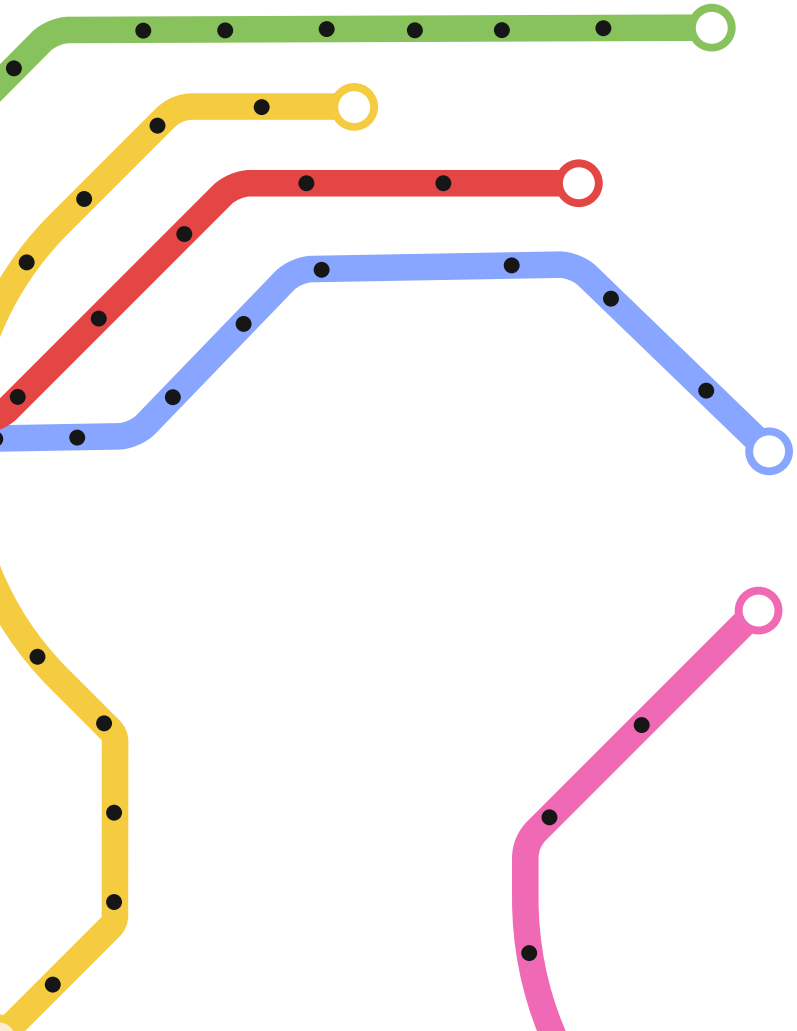


Cons

Might needs multiple runs to find a viable path

Introduce new hyperparameters like Dubins curve ration

Might generate weird paths (with a lot of curves)



Target rescue

Target Rescue

Goal

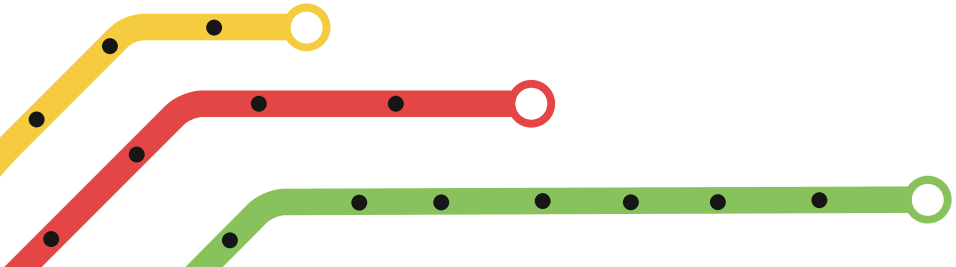
Reach the **goal** and **rescue** as much **victims** as possible in a **limited time**

Metrics

Time to **plan** the mission
Number and **value** of rescue victims
Time to reach the goal
Collisions

Algorithms

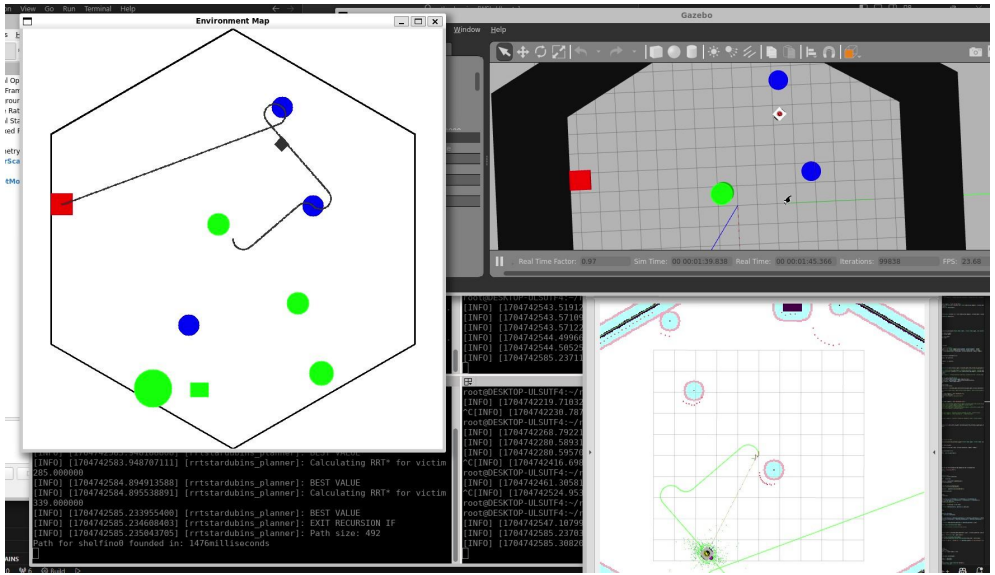
RRT* with **Dubins** paths
knapsack problem

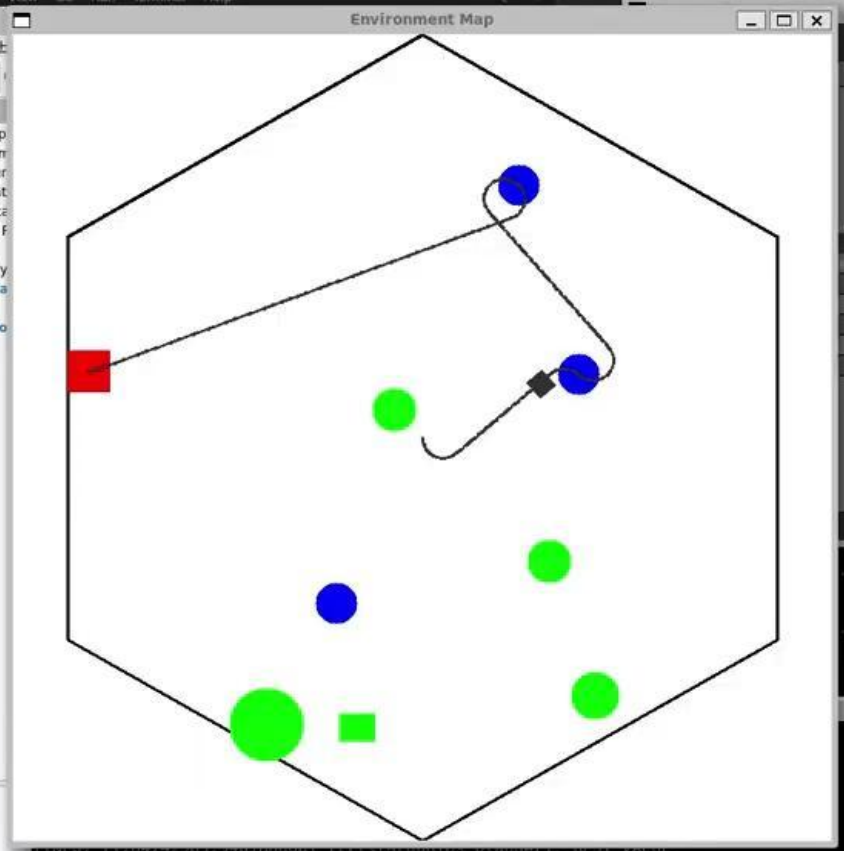


Target Rescue

To solve this problem we proceeded as follows:

- Build the map as the previous problem
- Find **path** to **goal**
- Find **path** start-**victims**-goal
- Try recursively to insert other victims in the computation
- Send the path to orchestrator





Gazebo

Window Help

Real Time Factor:

/root/ros2_ws/install/shelfino_navigation/share/shelfino_navigation/...

File Panels Help

Move Camera Select Focus Camera Measure 2D Pose Estimate

[INFO] [1704742583.948707111] [rrtstardubins_planner]: Calculating RRT* for victim 285.000000
[INFO] [1704742584.894913588] [rrtstardubins_planner]: BEST VALUE
[INFO] [1704742584.895538891] [rrtstardubins_planner]: Calculating RRT* for victim 339.000000
[INFO] [1704742585.233955400] [rrtstardubins_planner]: BEST VALUE
[INFO] [1704742585.234608403] [rrtstardubins_planner]: EXIT RECURSION IF
[INFO] [1704742585.235043705] [rrtstardubins_planner]: Path size: 492
Path for shelfino0 founded in: 1476milliseconds

Thanks!

Robot Planning and its Application

Bussola Riccardo, Zilio Nicola 12/01/2024
DISI, Università degli Studi di Trento.