MEMORIA PRACTICA 3

PLANIFICACION AUTOMÁTICA

R2P2

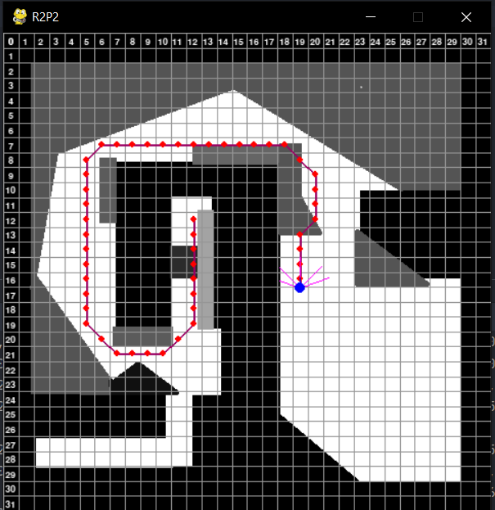
Lucas García de Viedma Pérez & Miguel Guerrero López

1. Get familiar with the R2P2 simulator. Run the simulator with the command python r2p2.py. It will create a plain scenario with a robot teleoperated by the arrow keys.

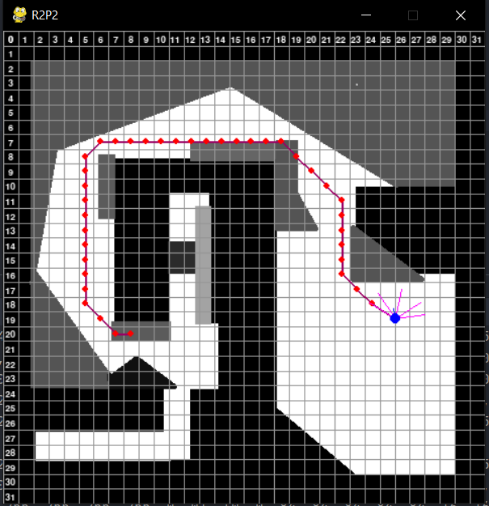
Icono

Descripción generada automáticamente

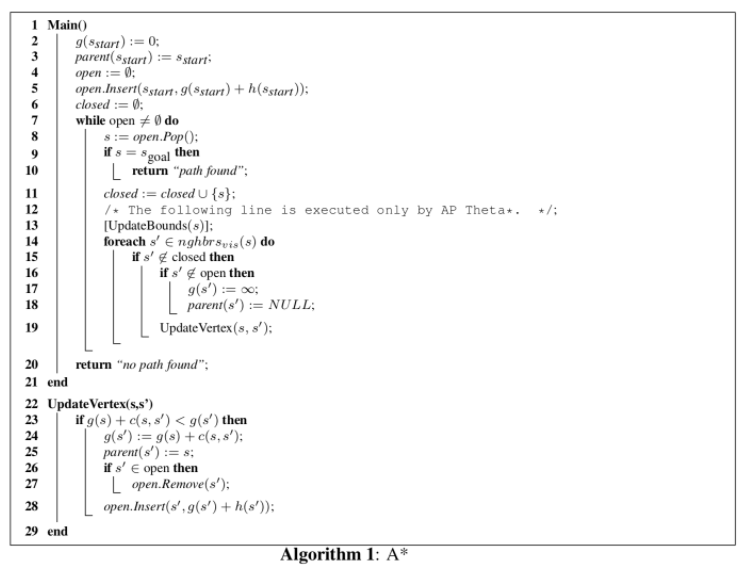
1. Run the Path Planning scenario as follows: “python r2p2.py --scenario ../conf/scenario-pathplanning.json”. Note that the robot is initially located at the position (18,19) and the goal at position (12,12). The simulator visualizes the computed path.



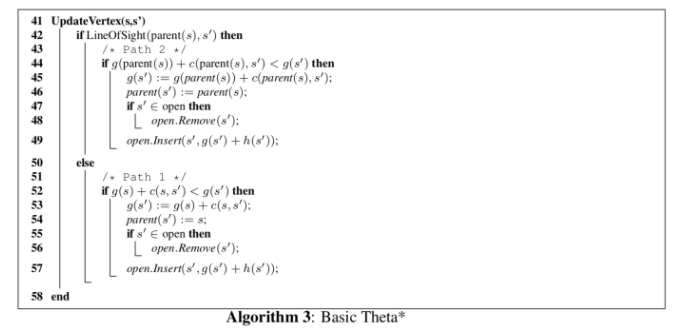
1. Given that the configuration of the robot controller is located in the file conf/controller-pathplanning.json, change the initial point to (27,19) and the goal to (8,20). You can also change the size of the grid. Remember that it is advisable to use values between 15 and 40 in order not to lose precision without compromising efficiency.



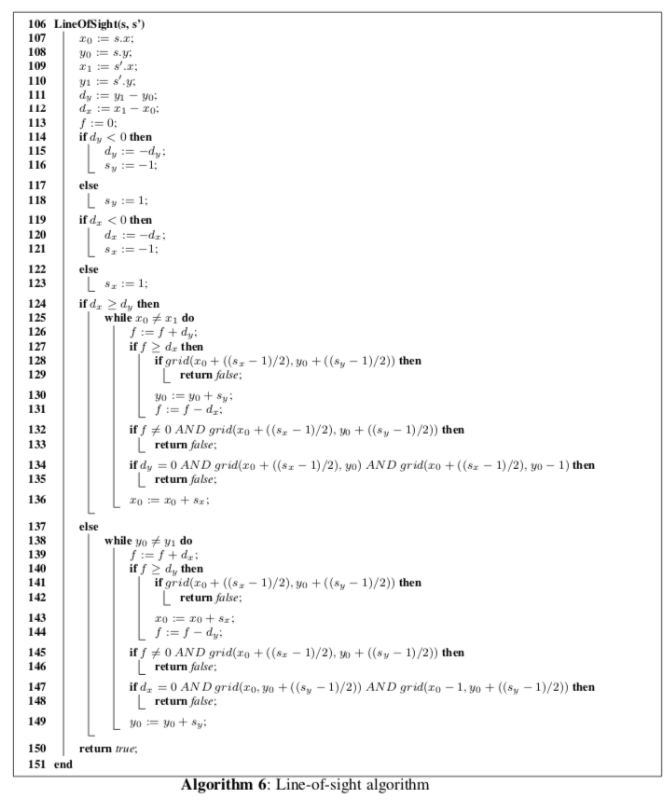
1. In order to change the algorithm (A\*, Dijkstra or Theta\*) just set "A\*" "Dijkstra" or "Theta\*" in the algorithm field of conf/controller-pathplanning.json (caution, the simulator is case sensitive). We provide you with the implementation of A\* and Dijkstra. The implementation of A\* is based on the pseudo-code of the next figure:



1. Implement Theta\* using the following pseudo-code:



Consider that the difference between A\* and Theta\* is the Line of sight. Here is the code to calculate it:



THETA ALGORITHM

The algorithm is in the tStar.py file.

1. Which heuristic is used by default? If needed, use the path-planning visualizer in <https://qiao.github.io/PathFinding.js/visual/> to reconstruct the scenario and observe the node expansion using the different algorithms.

The program uses the naive heuristic by default.

1. Implement the Manhattan, Octile and Euclidean heuristics for A\*/Theta\*. Go to the r2p2/heuristic.py file and fill out the functions Euclidean and Manhattan and create a new one for the Octile heuristic (don´t forget to register it). The functions take the initial and final points as tuples (X, Y) and must return a float with the heuristic value.

HEURISTICS

The heuristics are in the heuristics.py file

1. Create the image given on [slide 38](https://github.com/Malola2015/planningCourse/blob/master/robotics/pathplanning.pdf), set the grid size as 40, run the 3 algorithms and compare the results, changing the heuristic. Capture the images of the solution and upload them with the rest of the files.

A\* SOLUTIONS .

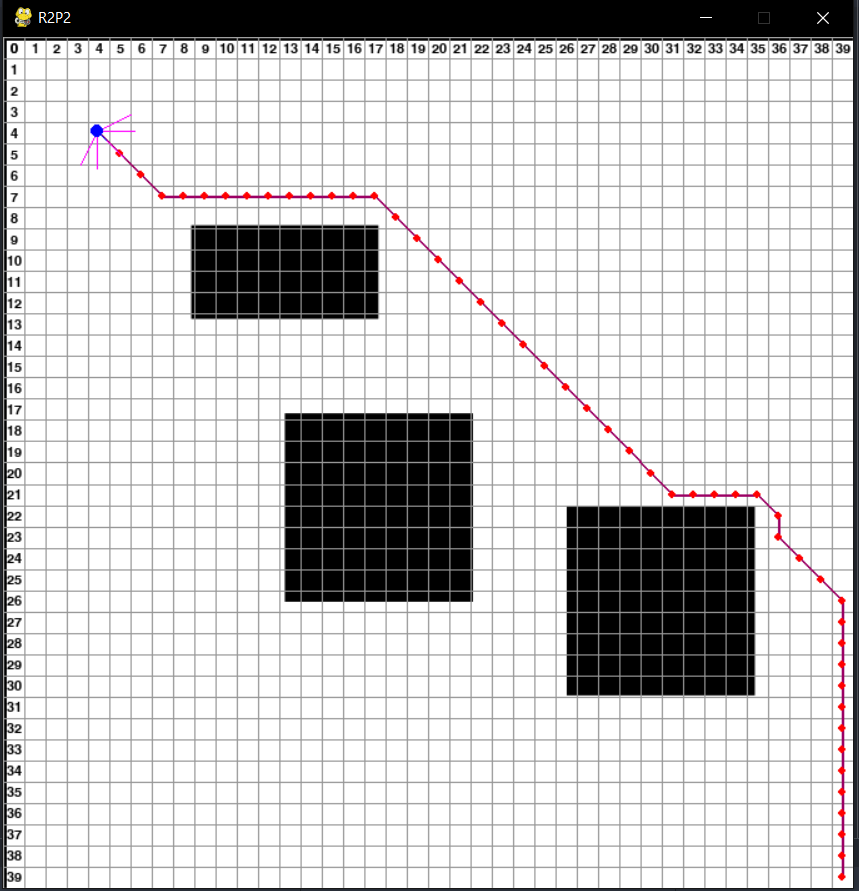
Gráfico

Descripción generada automáticamente Gráfico

Descripción generada automáticamente

Euclidean Manhattan

Gráfico

Descripción generada automáticamente 

Naive Octile

T\* SOLUTIONS

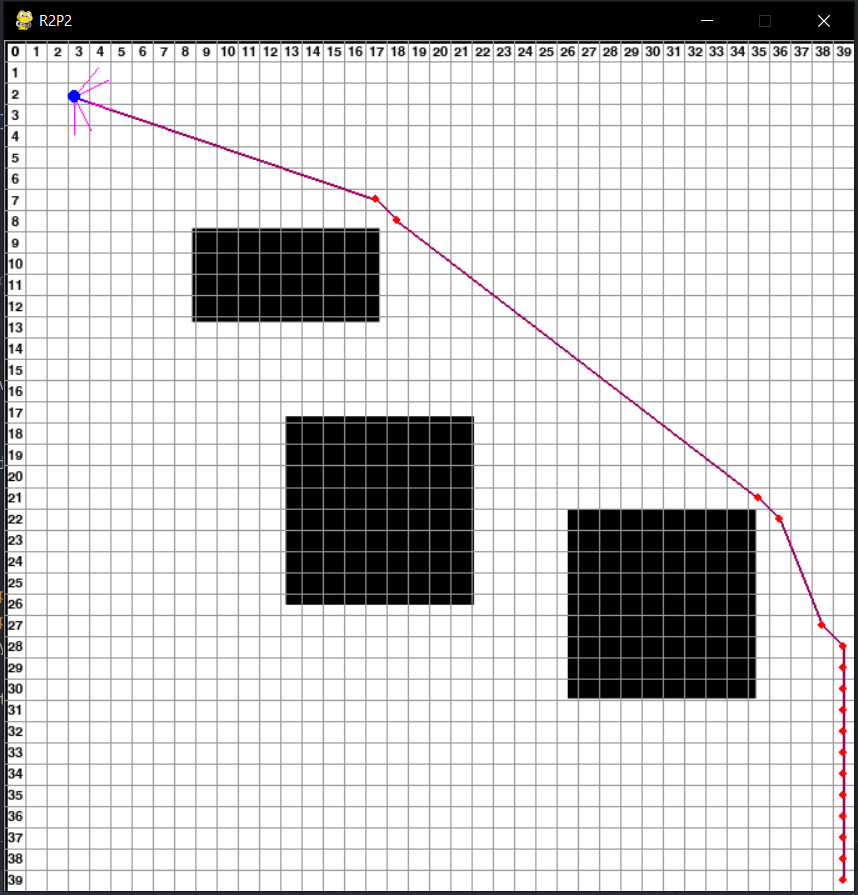
Gráfico

Descripción generada automáticamente Gráfico

Descripción generada automáticamente

Euclidean Manhattan

Gráfico

Descripción generada automáticamente 

Naive Octile

INTEGRATION

1. Modify the [Planetary exploration on Mars](https://github.com/Malola2015/planningCourse/blob/master/assignments/PlanetaryExploration.md) domain to consider the coordinates as objects of the type Point. For example, X=6 Y=10 will be represented as the object P0610. Now, you can delete all the obstacles since the path will be now delegated to the path-planner algorithm, freeing the PDDL planner from it.

MARS EXPLORATION MODIFIED .

The mars exploration domain and problem are in the d-planetaryExploration.pddl and d-planetaryExploration.pdd files.

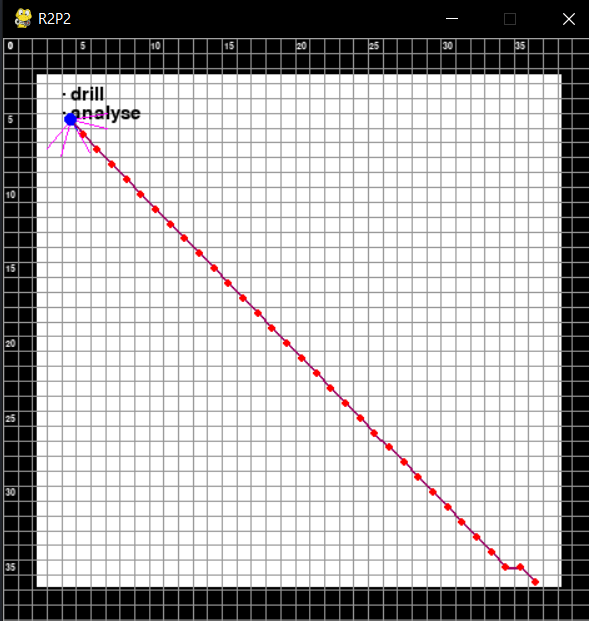
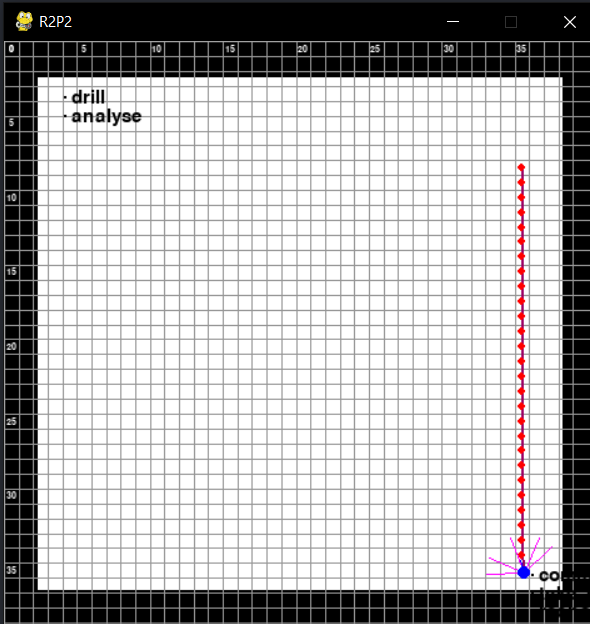
1. Run the new domain (if you have modelled it in a different way) and save the solution into a file called planning.txt. This file will be the input to the simulator that will call the path-planning algorithm. This file is located in ../res/ folder. Just overwrite it. Be aware that the initial position of the robot should be specified first.

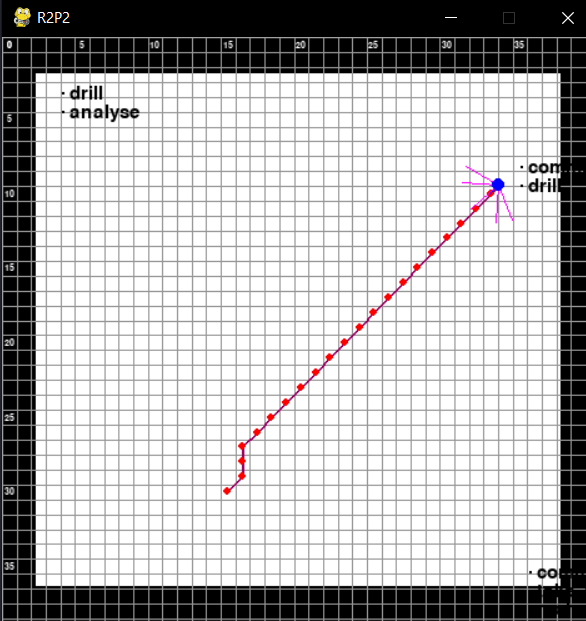
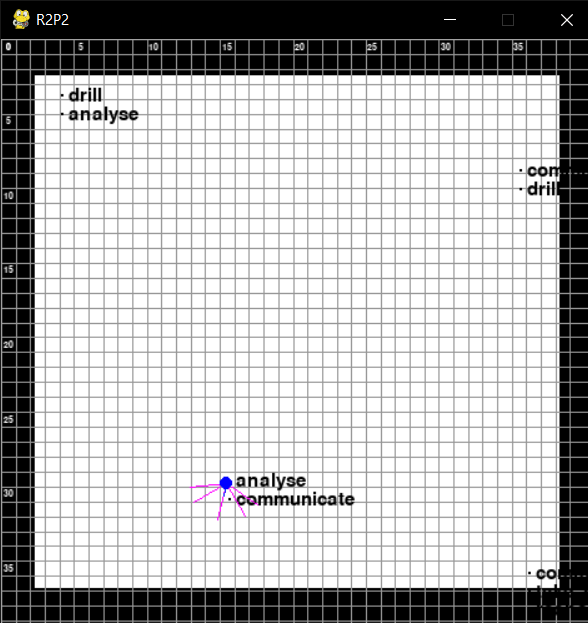
PLANNING

The plan is in the planning.txt file

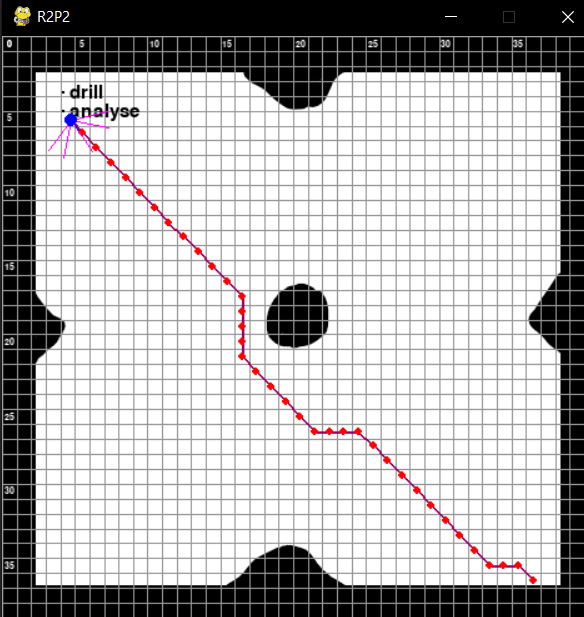
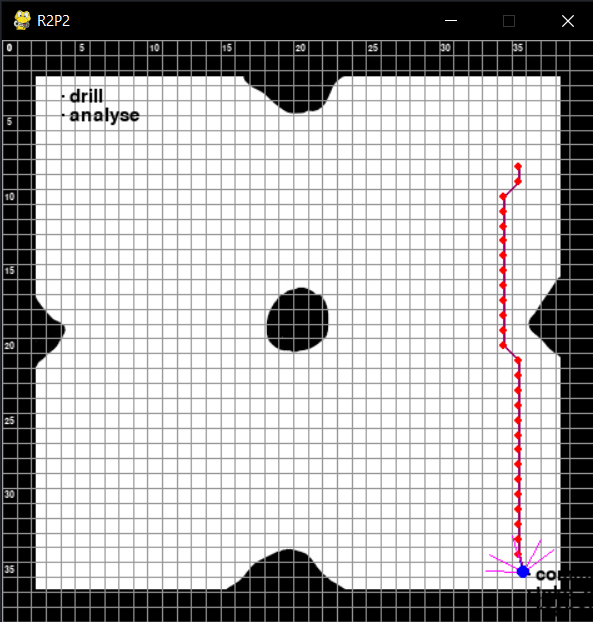
1. The code provided in the following link is placed in r2p2/controllers/pddl\_executor.py. It reads the PDDL output, and each time there is a move action with a coordinate, it calls the path-planning algorithm with the destination given by the action. This code is integrated in the simulator.
2. Change the heuristics and the algorithms in the appropriate function.
3. Run the planning domain scenario using the simulator.

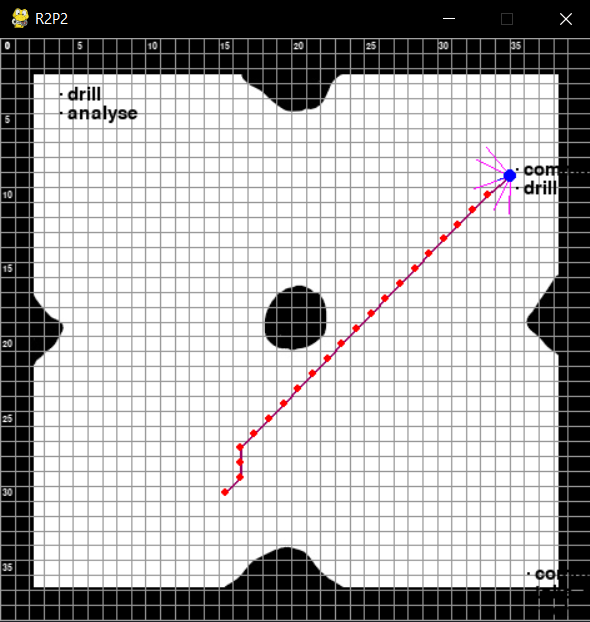
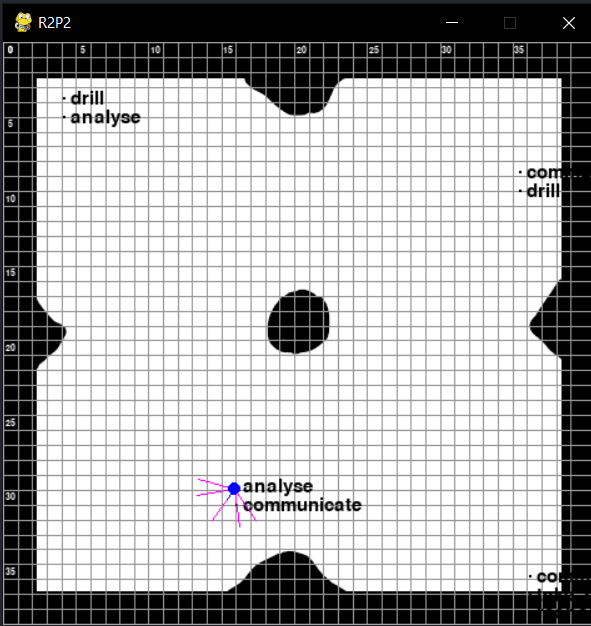
WITHOUT OBSTACLES

WITH OBSTACLES

WHAT ELSE?

Para controlar al robot de manera autónoma en un entorno tan cerrado y observable sí que sería suficiente. Sin embargo, en situaciones inesperadas, o que no se hayan establecido en el planificador, el robot no tendría como reaccionar. Por poner un ejemplo, si en este planificador incluyésemos colores para determinar la pendiente de una zona, el robot debería tener en cuenta que pendientes es capaz de subir, y no únicamente una reducción de velocidad según el color. Para ello, debería implementarse algún sistema que permitiese la replanificación con nueva información del entorno.