AERO60492 – Autonomous Mobile Robots

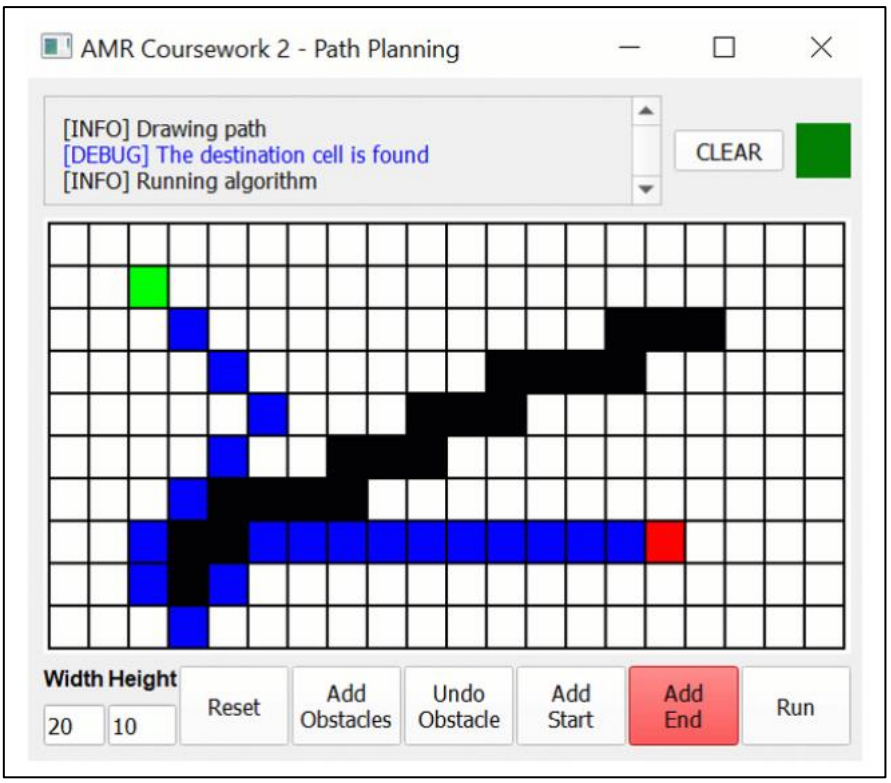
Coursework 2 – Path Planning

Section 1: aims, timeline, submission requirements, marking scheme.

Aims

The aim of this task is to implement a path planning algorithm for a generalised robotic system. The ability for a vehicle to automatically plan a route though terrain can significantly enhance the capabilities and reliability of mobile robots by reducing the operator workload and allowing the robot to make 'safe' or 'efficient' routing decisions based on its interpretation of the terrain or other hazards.

Using the Python programming language, you will create an implementation of the A* path planning algorithm. A graphical user interface (GUI) has been created with an empty function which must be edited. The image below shows an example of what a basic implementation of A* might achieve.



Timeline

Week 1 – coursework set with accompanying lecture, tutorial getting started lab session

Week 2 – continued self-work

Week 3 – deadline and submission (see BB page for deadline details)

Submission

The required submission a Python3 script function which implements the A* path planning algorithm as a function within the user interface program provided. Commenting should be used to explain the steps of your method and good coding standards should be followed.

IMPORTANT: This function must run! Any submissions which crash, have bugs, or otherwise cannot be downloaded and run within the GUI framework will not have marks assigned for a successful implementation or execution speed.

Please upload only your edited version of the `pathPlanner.py` script to Blackboard. If you decide to implement any advanced python functions and/or classes etc. They should all be contained within this single file.

Marking scheme

This coursework has a 15% contribution to the unit overall. Marks out of 20 will be awarded according to:

- Successful implementation - tested against idealised implementation and various test cases [10 marks]
- Explanation of the method used (in comments) [5 marks]
- Execution speed of code [2 marks]
- Good coding practices [3 marks]

Section 2: pre-requisites and background info.

Pre-requisites

It is expected that you already have a good understanding of the Python programming language. This work does not necessarily require the use of advanced programming methods or libraries. It can use basic math and array operations, flow control (i.e. if/else/while statements), and basic arithmetic value manipulations to achieve the required outcome. For a refresher on how to install and program using the Python3 language, please visit:

- https://docs.python.org/3/tutorial/index.html The official tutorial
- https://youtu.be/nLRL_NcnK-4?si=iw6mGQ3cqCkJgr1U Harvard CS50's Introduction to Programming with Python
- https://docs.anaconda.com/free/anaconda/install/index.html Installing Python using the Anaconda distribution.

There will be many ways to implement the underlying path planning algorithm. The exact programming style is left to you to explore as needed.

There are detailed instructions on installing and running the scripts and user interface below.

Section 3: task briefing and example code

Task - Implement A* algorithm

Your main task is to modify the path planning function to implement the A* algorithm to plot a route between the user-provided start and end locations while avoiding the user defined hazards.

(a) As part of this coursework, you are expected to perform independent research into the algorithm. There are many resources available in both texts, articles, and online. There have also been two lectures introducing the terminology and describing path planning methods.

The original description of the algorithm itself can be found in the original article:

> P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968, https://doi.org/10.1109/TSSC.1968.300136

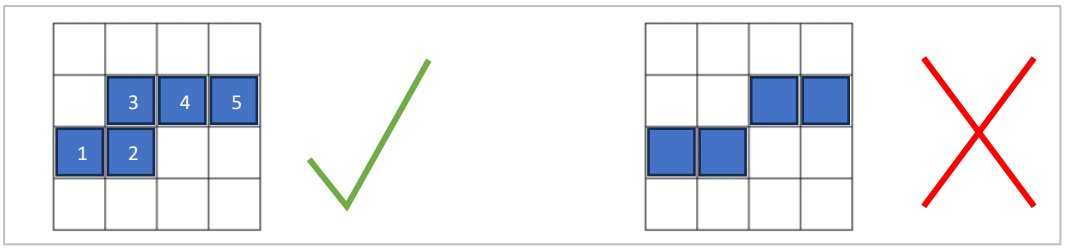
There is a copy hosted on Blackboard.

This is the method that your implementation will be evaluated against but there a numerous explanations of this first work available. Be careful to note the constraints listed below when reading other material as they might be implementing slight variations.

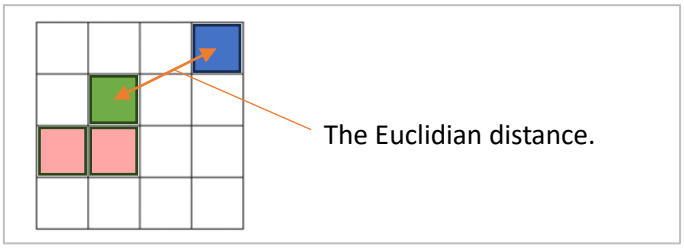
(b) Modify the `pathPlanner.py` Python3 script to implement the A* algorithm as originally described subject to the constraints listed. There is a function '`do_a_star()`' which is the main function called by the GUI and must remain; you can add additional functions and variables as needed, but they must reside within the `pathPlanner.py` file itself.

Constraints

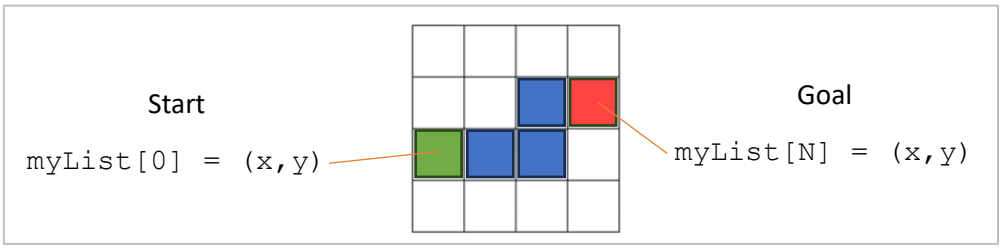
- The motion model must only allow four directions of movement. i.e. use moves in the up/down/left/right directions. No paths which move diagonally between cells are allowed. Each horizontal or vertical cell movement contributes a cost of '1' to the path length ($g(n)$).



- The heuristic function ($h(n)$) must use the Euclidean distance to destination.



- The function must return a list of node/cell coordinates with the start of the path at the top and the goal at the end.



- **No additional libraries.** It is possible to fully implement the algorithm without using any additional libraries. That said, efficient coding will allow a higher score on the execution speed evaluation.

Skeleton/GUI Code

Since this task introduces a more advanced python program (with a GUI rather than only command line), a few additional libraries might be required to get started. Since it is possible to cause system instability and version conflicts when installing python packages, it is recommended that the work is carried out in a 'Virtual Environment' (venv) - this effectively makes a separate set of local installations and settings, hence if a mistake is made, the virtual environment can be erased without leaving issues to the host system. A guide to setting up the environment with Anaconda is provided in the **Installation Instructions** document on blackboard.

Running the program

There are two files for this coursework.

- `gui.py` – the master program which controls creating a user interface and calls the path planning algorithm. DO NOT EDIT THIS
- `pathPlanner.py` – the path planning algorithm implementation. EDIT THIS ONE

The gui script will perform a number of useful tasks beyond just displaying a graphical input. When running a test of your path planner, if any mistakes have been made with bugs or syntax, the gui script will not crash. This is useful because edits can then be made to the path planner and retried without having to start from scratch each time.

To make a start, open a terminal, navigate to the project directory, activate the environment, then start the gui script. A user interface titled "AMR Coursework 2" should appear with a grid of squares and some buttons. The appearance and purpose of the buttons is given below:

Message outputs. Messages can start with different prefixes.

[INFO] it is an message from the gui/main control program.

[DEBUG] messages that the user can add within their own path planning implementation to aid development.

[ERROR] a copy of any error messages due to bugs in code/syntax.

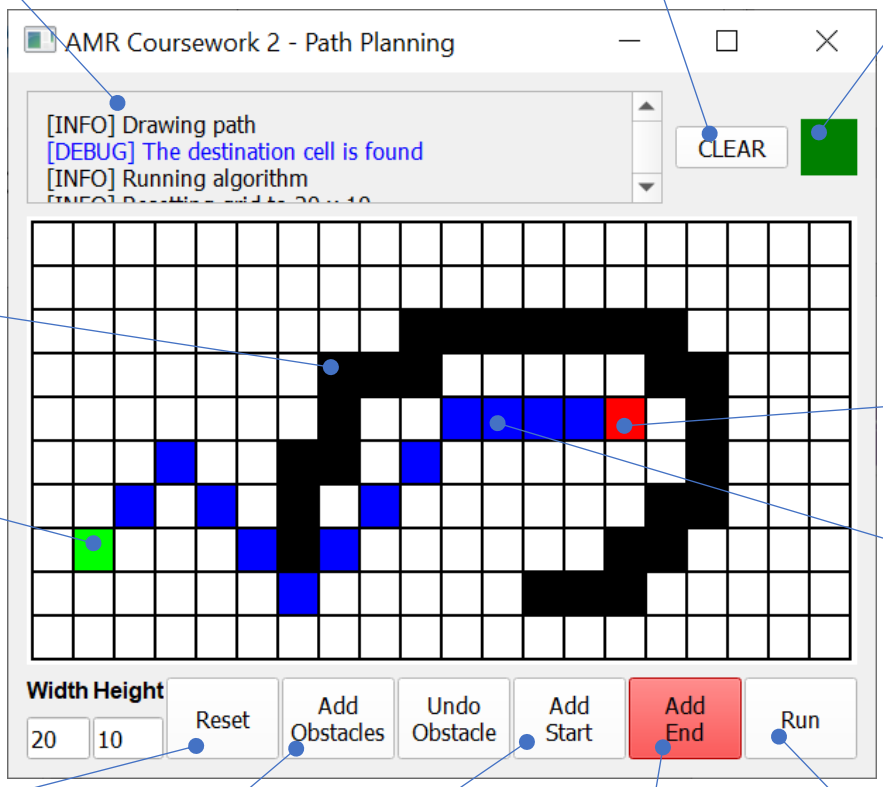
[WARN] code run was successful but might not have planned a valid path.

Clears message box

Status indicator.

GREEN: successful path planner code run.

RED: unsuccessful path planner execution



BLACK – walls/obstacles

RED – end point

GREEN – start point

BLUE – the algorithm output path

Size of cell grid

Remove all paths, 'walls', start, stop locations

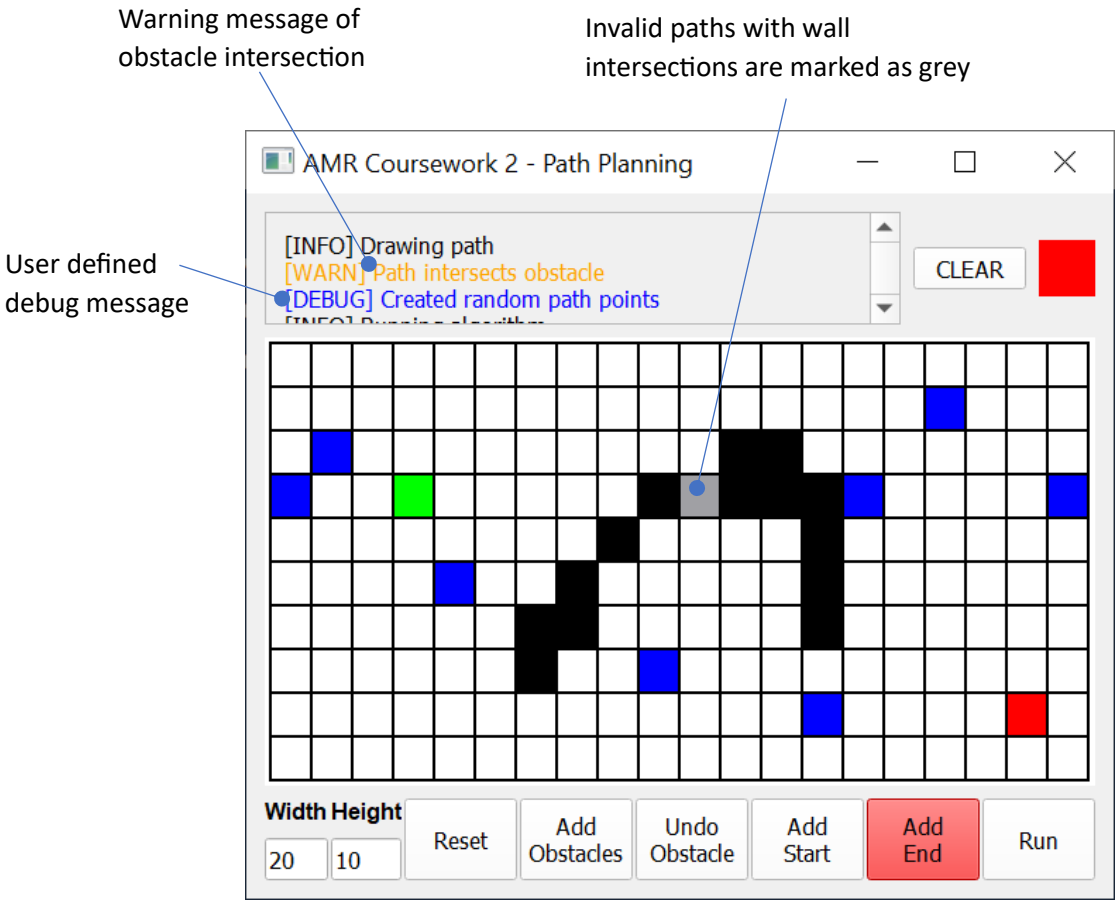
This mode enables the addition of blocked cells ('walls'). Use left click to add on grid.

Add start point cell

Add destination cell

Run the path panning algorithm

The skeleton code is provided to only demonstrate the interplay of the functions between the gui.py script and the pathPlanner.py script. It only produces a set of random points as the path and outputs an example debug message. It is not expected that this random example code should exist in your final submission. An example of the functionality of the skeleton code is shown below.

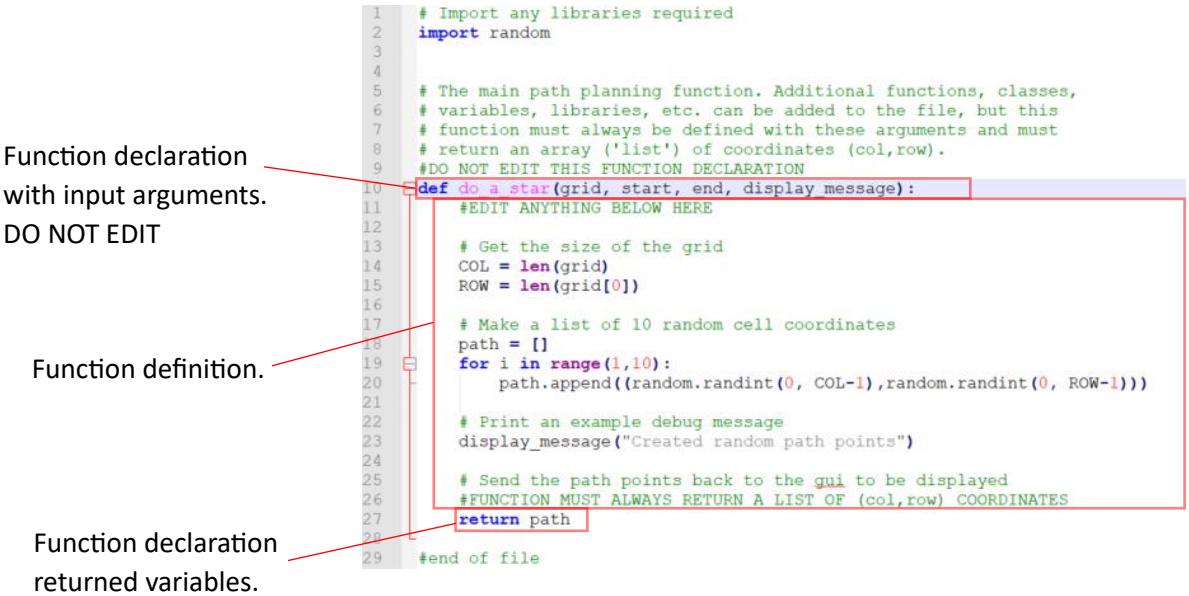


The pathPlanner.py file can be edited however is needed to implement the algorithm. You can add functions, variables, classes, however the `do_a_star()` function definition must not be changed.

THE FUNCTION NEEDS TO BE IMPLEMENTED FROM BASIC PROGRAMMING FUNCTIONS ONLY. Do not import any additional libraries. When the cost is tested, it will fail to run if you have added any additional `#inport libraryName` in your code. Creating your own sub functions is allowed using the `def functionName` type commands.

Path Planner Function Definition

Your task is to edit the path planner function definition code to perform the A* algorithm. The function declaration should not be changed. The declaration is the line which defines the inputs (arguments) and expected outputs (returns).



The arguments/return are:

grid – the occupancy grid. An array of 1's or 0's. If the value equals 1, it represents an empty cell which can become part of the path. If the value equals 0, then that cell represents a wall/obstacle and should not become part of the path.

start – the grid coordinate of the start location (col,row)

end – the grid coordinate of the end location (col,row)

display_messages – a function required to allow debug messages to be printed in the gui.

Return – a Python list of coordinates, e.g. path = [(1,2), (3,4), (9,8)]

Hint – don't forget Python arrays are 0 based (i.e. the first value is in the [0] index location.