

# AAE1001 Introduction to Artificial Intelligence and Data Analytics in Aerospace and Aviation Engineering

## Week 8 (Path Planning Programming Guide)

---

Dr Guohao Zhang, assisted by

Mr Zekun ZHANG, Mr Mingda YE, Ms Jingxiaotao FANG, Mr Chin Lok TSANG,  
Mr Di HAI, Mr Zhengdao LI

# Path Planning Example Code

# A-star Example

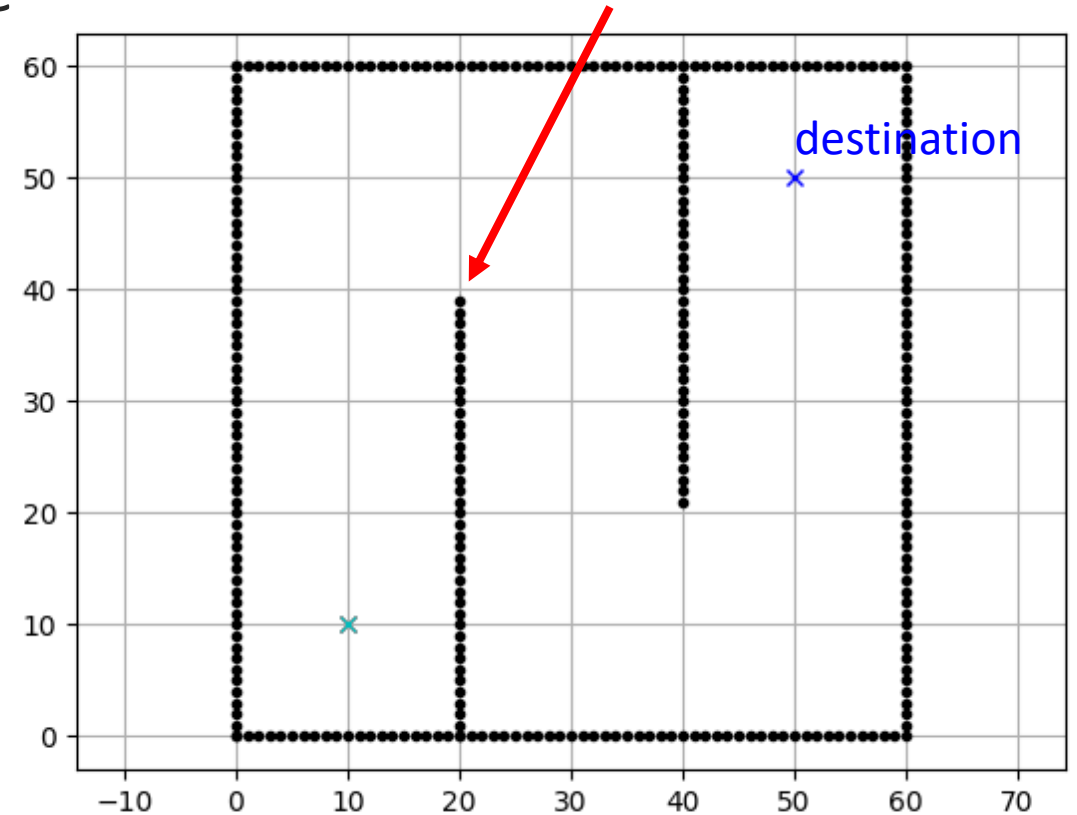
Each time A\* enters a node, it calculates the cost:  $f(n)$  -  $n$  being the neighboring node

It travel to all of the neighboring nodes, and then enters the node with the lowest value of  $f(n)$

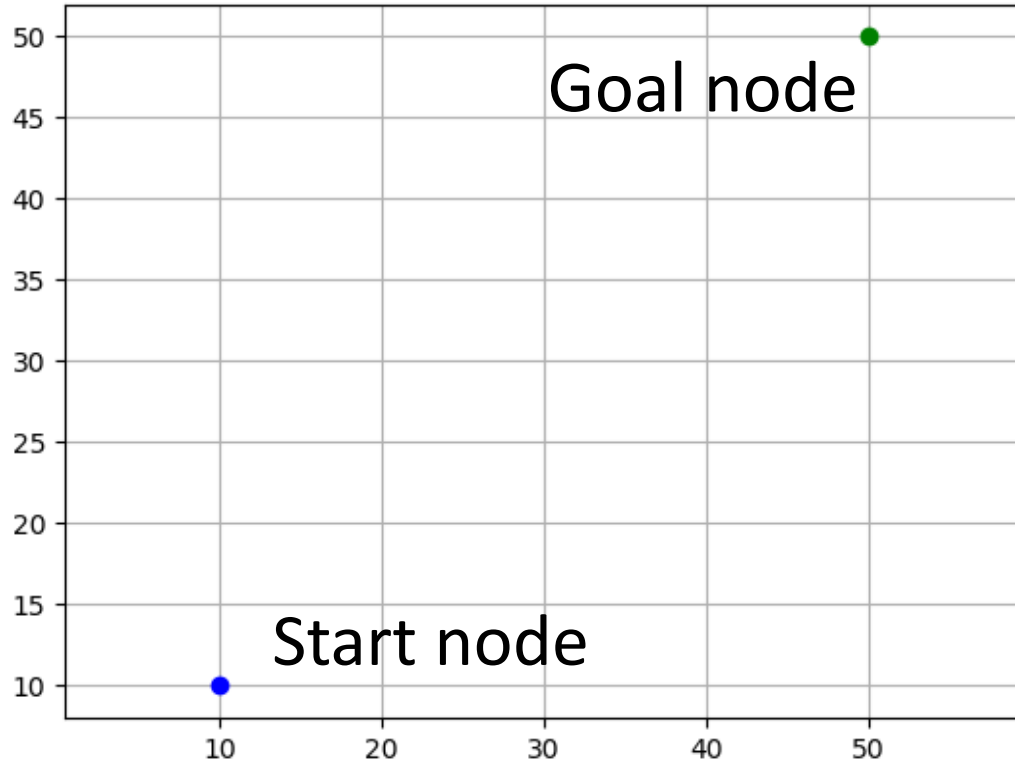
These values we calculate using the following formula:

$$f(x, y) = g(x, y) + h(x, y)$$

Wall (obstacles)  
cannot go through!



# Code Example (1): Set Up Start and Goal Node



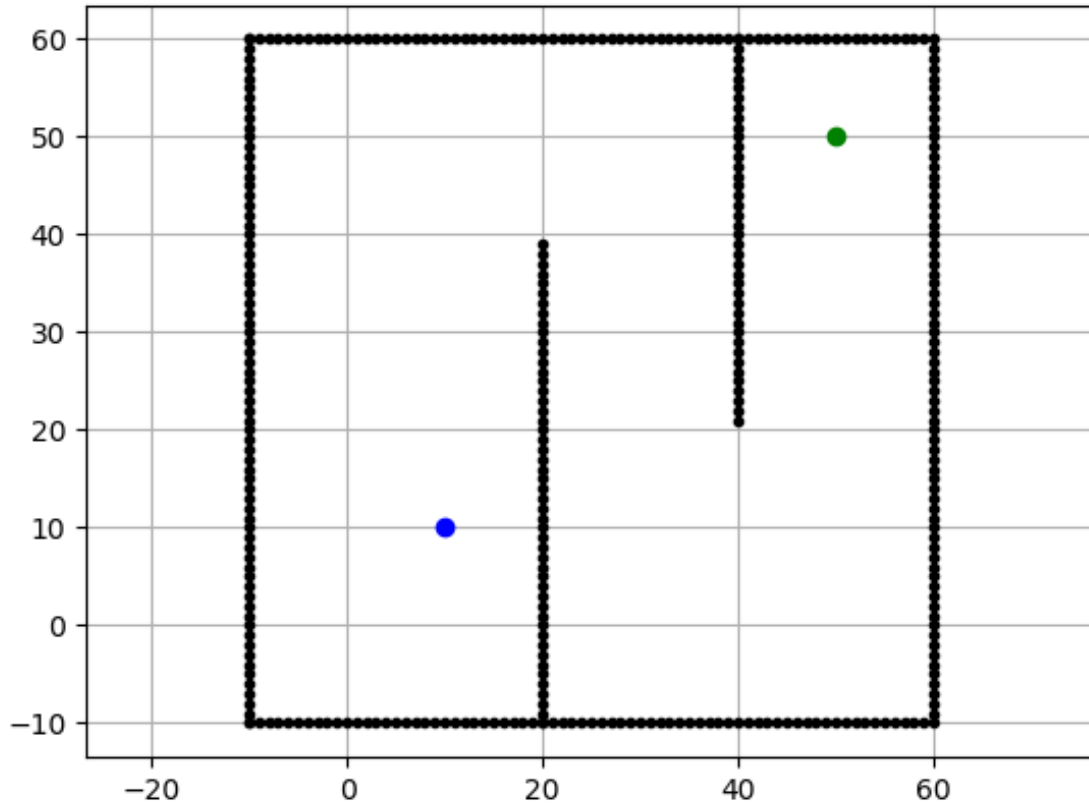
● Start node    ● Goal node

```
# start and goal position  
sx = 10.0 # [m]  
sy = 10.0 # [m]  
gx = 50.0 # [m]  
gy = 50.0 # [m]  
grid_size = 2 # [m]
```

Base code tutorial:

[https://www.youtube.com/watch?v=P\\_RKLhcG2kB0&ab\\_channel=POLYUIPNL](https://www.youtube.com/watch?v=P_RKLhcG2kB0&ab_channel=POLYUIPNL)

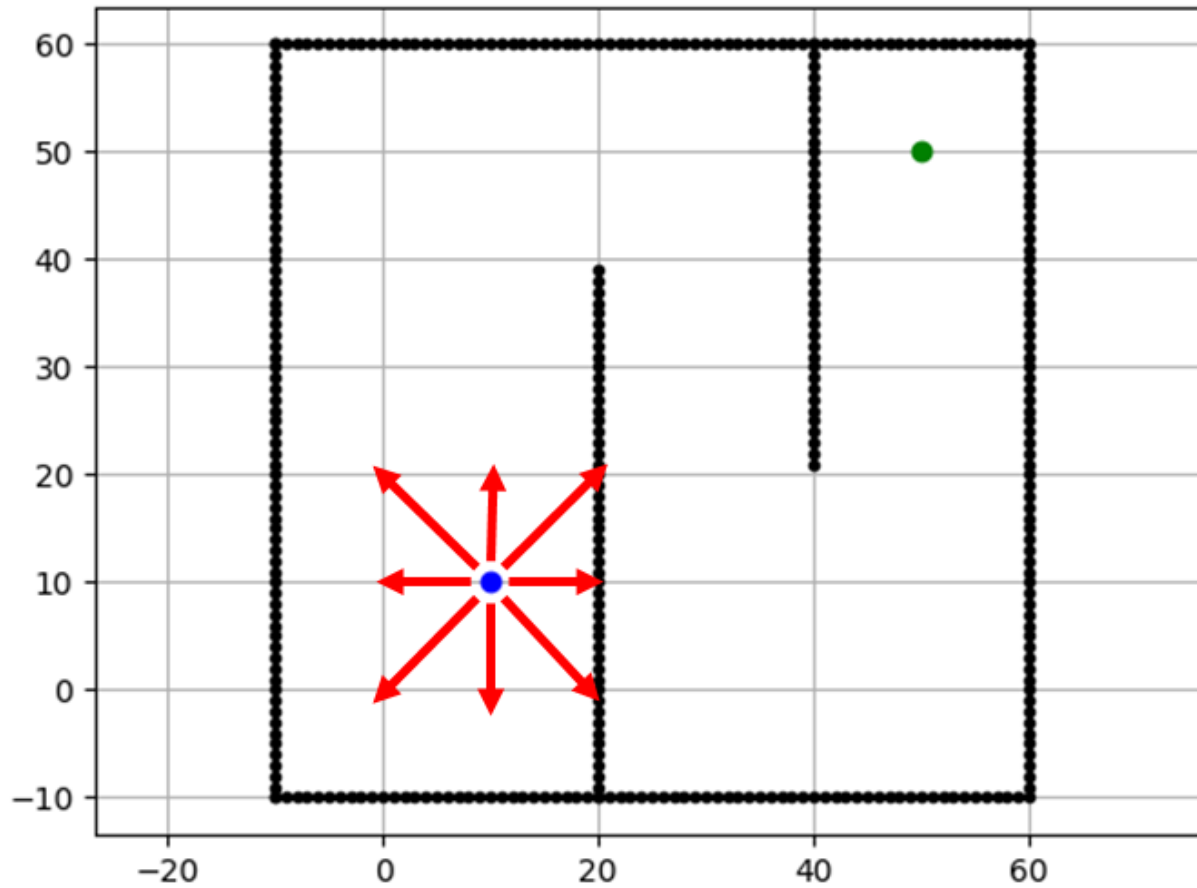
# Code Example (2): Set Up Obstacle



```
# set obstacle positions
ox, oy = [], []
for i in range(-10, 60): # draw the button border
    ox.append(i)
    oy.append(-10.0)
for i in range(-10, 60):
    ox.append(60.0)
    oy.append(i)
for i in range(-10, 61):
    ox.append(i)
    oy.append(60.0)
for i in range(-10, 61):
    ox.append(-10.0)
    oy.append(i)
for i in range(-10, 40):
    ox.append(20.0)
    oy.append(i)
for i in range(0, 40):
    ox.append(40.0)
    oy.append(60.0 - i)
```

- Start node
- Goal node
- █ Obstacle (wall)

# Code Example (3): Neighboring Node Search



```
def get_neighbouring_node(): # the cost of the surrounding 8 points
    # dx, dy, cost
    motion = [[1, 0, 1],
              [0, 1, 1],
              [-1, 0, 1],
              [0, -1, 1],
              [-1, -1, math.sqrt(2)],
              [-1, 1, math.sqrt(2)],
              [1, -1, math.sqrt(2)],
              [1, 1, math.sqrt(2)]]

    return motion
```

- Start node
- Goal node
- █ Obstacle (wall)

# Code Example (4): Cost Calculation

Exact cost  $g(x, y)$  calculation

```
node = self.Node(current.x + self.motion[i][0],  
                 current.y + self.motion[i][1],  
                 current.cost + self.motion[i][2], c_id)
```

Heuristic cost  $h(x, y)$  calculation

```
def calc_heuristic(n1, n2):  
    w = 1.0 # weight of heuristic  
    d = w * math.hypot(n1.x - n2.x, n1.y - n2.y)  
    return d
```

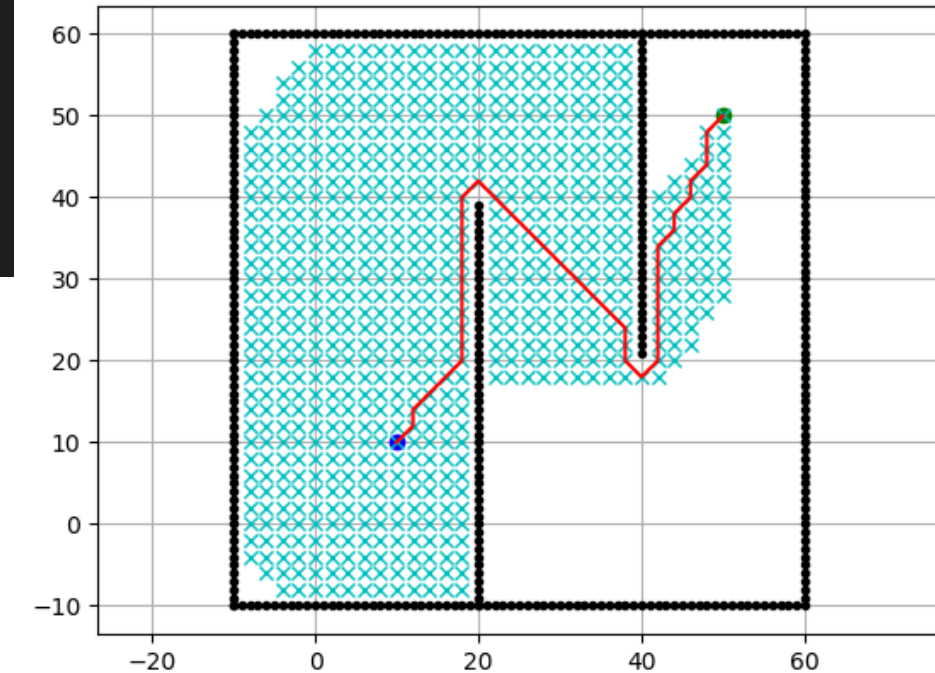


# Code Example (5): Calculation of Final Path

```
def calc_final_path(self, goal_node, closed_set):
    # generate final course
    rx, ry = [self.calc_grid_position(goal_node.x, self.min_x)], [
        self.calc_grid_position(goal_node.y, self.min_y)] # save the goal node as the first point
    parent_index = goal_node.parent_index
    while parent_index != -1:
        n = closed_set[parent_index]
        rx.append(self.calc_grid_position(n.x, self.min_x))
        ry.append(self.calc_grid_position(n.y, self.min_y))
        parent_index = n.parent_index

    return rx, ry
```

Retrieve the optimal path from all passing through nodes (once being the current node)





# Path Planning Programming Guide

# The Path Planning Code

- You can find the path planning code inside the course GitHub repository
- There are 2 set of codes:
  - A default one
  - A noted one
- The default one is a basic A\* path planning code without any extra information and features
- The noted one provides an example of what your code should look like after modifications (**Remember each group should complete a different set of obstacles and requirements**)
- Repository link: [https://github.com/IPNL-POLYU/PolyU\\_AAE1001\\_Github\\_Project](https://github.com/IPNL-POLYU/PolyU_AAE1001_Github_Project)

# Where you can find the code

The screenshot shows the GitHub interface for the repository `PolyU_AAE1001_Github_Project`. The `Files` tab is selected, displaying the file structure. The `Sample Codes` directory is expanded, showing a list of files. A red circle highlights the repository path `PolyU_AAE1001_Github_Project / Sample Codes /` in the breadcrumb navigation. Another red circle highlights the file `a_star_noted.py` in the file list.

Name	Last commit message	Last commit date
..		
Tutorial 1 Sample.py	add AAE1001	2 months ago
a_star_noted.py	add AAE1001	2 months ago
a_star_original.py	add AAE1001	2 months ago
animation.gif	add AAE1001	2 months ago
readme.md	add AAE1001	2 months ago

# Noted Version Guide

- Line 50,51: Declaration of cost intensive area cost modifier
- Line 53: Declare cost per grid

```
34
35     self.resolution = resolution # get resolution of the grid
36     self.rr = rr # robot radius
37     self.min_x, self.min_y = 0, 0
38     self.max_x, self.max_y = 0, 0
39     self.obstacle_map = None
40     self.x_width, self.y_width = 0, 0
41     self.motion = self.get_motion_model() # motion model for grid search expansion
42     self.calc_obstacle_map(ox, oy)
43
44     self.fc_x = fc_x
45     self.fc_y = fc_y
46     self.tc_x = tc_x
47     self.tc_y = tc_y
48
49
50     self.Delta_C1 = 0.2 # cost intensive area 1 modifier
51     self.Delta_C2 = 0.4 # cost intensive area 2 modifier
52
53     self.costPerGrid = 1
54
```

# Noted Version Guide

- Line 115: Showing the final calculation of total trip time
- Line 135-144: Adding additional cost during cost intensive area

```

103  if show_animation: # pragma: no cover
104      plt.plot(self.calc_grid_position(current.x, self.min_x),
105               self.calc_grid_position(current.y, self.min_y), "xc")
106      # for stopping simulation with the esc key.
107      plt.gcf().canvas.mpl_connect('key_release_event',
108                                   lambda event: [exit(
109                                       0) if event.key == 'escape' else None])
110      if len(closed_set.keys()) % 10 == 0:
111          plt.pause(0.001)
112
113      # reaching goal
114      if current.x == goal_node.x and current.y == goal_node.y:
115          print("Total Trip time required -> ",current.cost )
116          goal_node.parent_index = current.parent_index
117          goal_node.cost = current.cost
118          break
119
120      # Remove the item from the open set
121      del open_set[c_id]
122
123      # Add it to the closed set
124      closed_set[c_id] = current
125
126      # print(len(closed_set))
127
128      # expand_grid search grid based on motion model
129      for i, _ in enumerate(self.motion): # tranverse the motion matrix
130          node = self.Node(current.x + self.motion[i][0],
131                           current.y + self.motion[i][1],
132                           current.cost + self.motion[i][2] * self.costPerGrid, c_id)
133
134          ## add more cost in cost intensive area 1
135          if self.calc_grid_position(node.x, self.min_x) in self.tc_x:
136              if self.calc_grid_position(node.y, self.min_y) in self.tc_y:
137                  # print("cost intensive area!!")
138                  node.cost = node.cost + self.Delta_C1 * self.motion[i][2]
139
140          # add more cost in cost intensive area 2
141          if self.calc_grid_position(node.x, self.min_x) in self.fc_x:
142              if self.calc_grid_position(node.y, self.min_y) in self.fc_y:
143                  # print("cost intensive area!!")
144                  node.cost = node.cost + self.Delta_C2 * self.motion[i][2]
145          # print()
146

```

# Noted Version Guide

- Line 263-270: Declaring motions for the aircraft
- Line 279-284: Declaring starting point and end point

```
260 @staticmethod
261 def get_motion_model(): # the cost of the surrounding 8 points
262     # dx, dy, cost
263     motion = [[1, 0, 1],
264               [0, 1, 1],
265               [-1, 0, 1],
266               [0, -1, 1],
267               [-1, -1, math.sqrt(2)],
268               [-1, 1, math.sqrt(2)],
269               [1, -1, math.sqrt(2)],
270               [1, 1, math.sqrt(2)]]
271
272     return motion
273
274
275 def main():
276     print(__file__ + " start the A star algorithm demo !!") # print simple notes
277
278     # start and goal position
279     sx = 0.0 # [m]
280     sy = 0.0 # [m]
281     gx = 50.0 # [m]
282     gy = 0.0 # [m]
283     grid_size = 1 # [m]
284     robot_radius = 1.0 # [m]
```



# Noted Version Guide

- Line 309-329: Adding obstacles
- Line 337-348, Adding cost intensive areas  
(Hint: Refer to this part for your task 2!)

```
308 # set obstacle positions for group 9
309 ox, oy = [], []
310 for i in range(-10, 60): # draw the button border
311     ox.append(i)
312     oy.append(-10.0)
313 for i in range(-10, 60): # draw the right border
314     ox.append(60.0)
315     oy.append(i)
316 for i in range(-10, 60): # draw the top border
317     ox.append(i)
318     oy.append(60.0)
319 for i in range(-10, 60): # draw the left border
320     ox.append(-10.0)
321     oy.append(i)
322
323 for i in range(-10, 30): # draw the free border
324     ox.append(20.0)
325     oy.append(i)
326
327 for i in range(0, 20):
328     ox.append(i)
329     oy.append(-1 * i + 10)
330
331 # for i in range(40, 45): # draw the button border
332 #     ox.append(i)
333 #     oy.append(30.0)
334
335
336 # set cost intensive area 1
337 fc_x, fc_y = [], []
338 for i in range(30, 40):
339     for j in range(0, 40):
340         fc_x.append(i)
341         fc_y.append(j)
342
343 # set cost intensive area 1
344 tc_x, tc_y = [], []
345 for i in range(10, 20):
346     for j in range(20, 50):
347         tc_x.append(i)
348         tc_y.append(j)
```

# Bonus !

- If you wish to do the trip cost (in terms of expense) calculation using the program, you should add the calculation function under line 117, inside the reaching goal condition
- It would be even better if the program could distinguish viable and non-viable aircraft types!
- Use the noted version as your example to modify your own code!

# Bonus Showcase

When you add in a cost calculation function, the output should look something like this, it should be able to:

1. Calculate and show each aircraft types' operating costs
2. Mention which type might not be viable for certain scenarios

```
min_x: -10
min_y: -10
max_x: 60
max_y: 60
x_width: 70
y_width: 70
Total travelling time -> 93.35575746753788
A321 not viable!
Total cost of operating A330 in this scenario: 27360.167918740684
Total cost of operating A350 in this scenario: 30752.648960130347
```