

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC BÁCH KHOA**  
**KHOA ĐIỆN-ĐIỆN TỬ**  
**BỘ MÔN TỰ ĐỘNG**



**KỸ THUẬT ROBOT - EE3065**

---

**BÁO CÁO BÀI TẬP LỚN**  
**ARTICULATED ARM ROBOT**

---

GVHD: NGUYỄN HOÀNG GIÁP  
SV thực hiện: HOÀNG SỸ NHẤT  
MSSV: 2111915

Thành phố Hồ Chí Minh, Tháng 12/2024

## Mục lục

<b>1</b>	<b>GIỚI THIỆU</b>	<b>3</b>
1.1	Sơ lược về Articulated Arm Robot . . . . .	3
1.2	Giới thiệu về ngôn ngữ lập trình Python . . . . .	4
<b>2</b>	<b>ĐỘNG HỌC THUẬN</b>	<b>7</b>
2.1	Mô phỏng hình dạng Robot . . . . .	7
2.1.1	Xác định các thông số và lập bảng DH . . . . .	7
2.1.2	Xây dựng môi trường giả lập Robot . . . . .	8
2.1.3	Vẽ Robot . . . . .	10
2.2	Xây dựng UI và Animation cho bài toán động lực thuận . . . . .	11
2.2.1	Giao diện người dùng UI . . . . .	11
2.2.2	Tạo animation và di chuyển robot theo bài toán động lực thuận . . . . .	12
<b>3</b>	<b>ĐỘNG HỌC NGHỊCH</b>	<b>15</b>
3.1	DH-Table . . . . .	15
<b>4</b>	<b>QUY HOẠCH QUỸ ĐẠO TRAJECTORY PLANNING</b>	<b>16</b>
4.1	DH-Table . . . . .	16
<b>5</b>	<b>MA TRẬN JACOBI VÀ ĐIỂM KÌ DI</b>	<b>17</b>
5.1	DH-Table . . . . .	17
<b>6</b>	<b>Path Finding</b>	<b>18</b>
6.1	Sơ lược về bài toán . . . . .	18
6.2	Những đánh giá về game . . . . .	18
6.3	Giao diện trực quan hóa . . . . .	19
6.4	Xây dựng các cấu trúc để mô phỏng bài toán . . . . .	22
6.4.1	Cấu trúc file <code>astar.py</code> . . . . .	22
6.4.2	Cấu trúc file <code>map.py</code> . . . . .	22
6.5	Định nghĩa không gian trạng thái . . . . .	23
6.6	A* Search . . . . .	24
<b>7</b>	<b>ĐÁNH GIÁ CHUNG VÀ KẾT LUẬN</b>	<b>27</b>



<b>8 VIDEO DEMO VÀ CODE PYTHON</b>	<b>27</b>
<b>Tài liệu</b>	<b>27</b>

# 1 GIỚI THIỆU

## 1.1 Sơ lược về Articulated Arm Robot

Theo Robotic Institute of America (*RIA*): Robot là một hệ thống đa tác vụ có thể lập trình được thiết kế để di chuyển vật liệu, bộ phận, dụng cụ hoặc thiết bị chuyên dụng thông qua chuyển động được lập trình theo các biến số để thực hiện các nhiệm vụ cụ thể đó, ngoài ra còn có thể thu thập thông tin từ môi trường và di chuyển thông minh theo đáp ứng.

Nhìn chung, “*Robotic*” là thuật ngữ khoa học để định nghĩa lĩnh vực nghiên cứu khoa học về sự kết nối thông minh giữa việc ra quyết định và hành động. Do đó, có thể nhận định Robot là một chủ đề liên ngành liên quan đến các lĩnh vực cơ khí, điều khiển, máy tính và điện tử.

Articulated Arm là một loại cánh tay robot được thiết kế để mô phỏng chuyển động của cánh tay con người. Dưới đây là một số điểm nổi bật của Articulated Arm:

- **Cấu trúc:**

- Gồm nhiều khâu (*link*) được kết nối bằng các khớp (*joint*).
- Các khớp có thể là khớp quay (*cho phép xoay*) hoặc khớp dịch chuyển (*cho phép di chuyển tuyến tính*).

- **Bậc tự do:**

- Thường có nhiều bậc tự do, cho phép thực hiện các chuyển động và định vị phức tạp.

- **End Effectors (*Công cụ cuối*):**

- Có thể được trang bị nhiều công cụ hoặc kẹp ở cuối để thực hiện các nhiệm vụ cụ thể, như hàn, sơn, hoặc nhặt và đặt đồ vật.

- **Hệ thống điều khiển:**

- Thường được điều khiển bởi các thuật toán và phần mềm tinh vi, cho phép thực hiện các chuyển động chính xác và tự động hóa.

Articulated Arm được ứng dụng phổ biến trong các lĩnh vực:

- **Sản xuất:** Được sử dụng rộng rãi trong dây chuyền lắp ráp cho các nhiệm vụ như hàn, sơn và đóng gói.
- **Chăm sóc sức khỏe:** Sử dụng trong các robot phẫu thuật để hỗ trợ bác sĩ trong các nhiệm vụ chính xác.
- **Nghiên cứu và phát triển:** Được sử dụng trong các phòng thí nghiệm cho các thí nghiệm yêu cầu thao tác chính xác với các đối tượng.
- **Giải trí:** Được sử dụng trong animatronics và hiệu ứng đặc biệt trong phim.



**Hình 1:** Các mẫu cánh tay robot trong công nghiệp

## 1.2 Giới thiệu về ngôn ngữ lập trình Python

Python là một ngôn ngữ lập trình bậc cao, được phát triển bởi Guido van Rossum và lần đầu tiên được phát hành vào năm 1991. Nó nổi bật với cú pháp rõ ràng, dễ đọc và dễ học, giúp lập trình viên có thể phát triển ứng dụng một cách nhanh chóng và hiệu quả.

Các đặc điểm nổi bật của Python:

- **Cú pháp đơn giản:**

- Python có cú pháp gần gũi với ngôn ngữ tự nhiên, giúp người mới bắt đầu dễ dàng tiếp cận.

- **Đa năng:**

- Python có thể được sử dụng cho nhiều mục đích khác nhau, bao gồm phát triển web, phân tích dữ liệu, học máy, trí tuệ nhân tạo, tự động hóa, và nhiều lĩnh vực khác.

- **Thư viện phong phú:**

- Python có một hệ sinh thái thư viện phong phú, bao gồm các thư viện nổi tiếng như NumPy, Pandas, Matplotlib, TensorFlow và Django, giúp lập trình viên dễ dàng thực hiện các tác vụ phức tạp.

- **Hỗ trợ lập trình hướng đối tượng:**

- Python hỗ trợ lập trình hướng đối tượng, cho phép người lập trình tổ chức mã nguồn một cách hiệu quả hơn.

- **Cộng đồng lớn:**

- Python có một cộng đồng lập trình viên lớn và năng động, cung cấp nhiều tài nguyên, hướng dẫn và hỗ trợ.

Trong bài tập lớn này sẽ sử dụng Python và các thư viện hỗ trợ sau sau để tạo giao diện và giả lập cánh tay robot:

- **Pygame:**

- Thư viện dùng để vẽ 2-D và tạo animation cũng như tương tác với bàn phím để xoay góc nhìn trong môi trường giả lập cánh tay robot.

- **Numpy:**

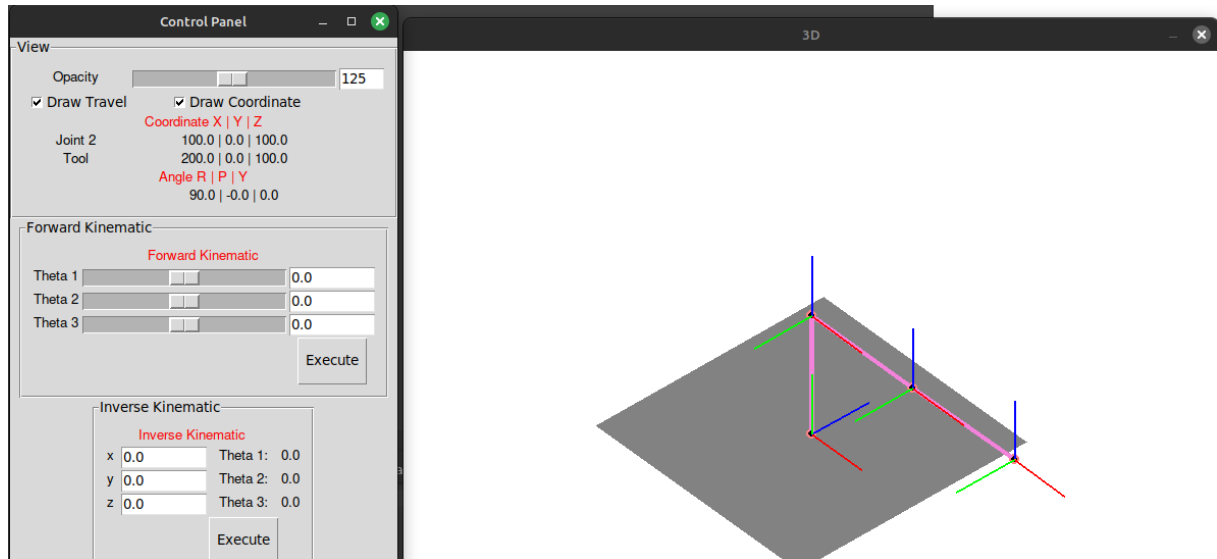
- Thư viện Numpy được sử dụng để tính toán các ma trận và một vài phép toán học cơ bản.

- **Matplotlib:**

- Dùng để vẽ đồ thị.

- Tkinter:

- Thư viện hỗ trợ tạo giao diện người dùng UI (*User Interface*).



Hình 2: Chương trình được viết bằng Python

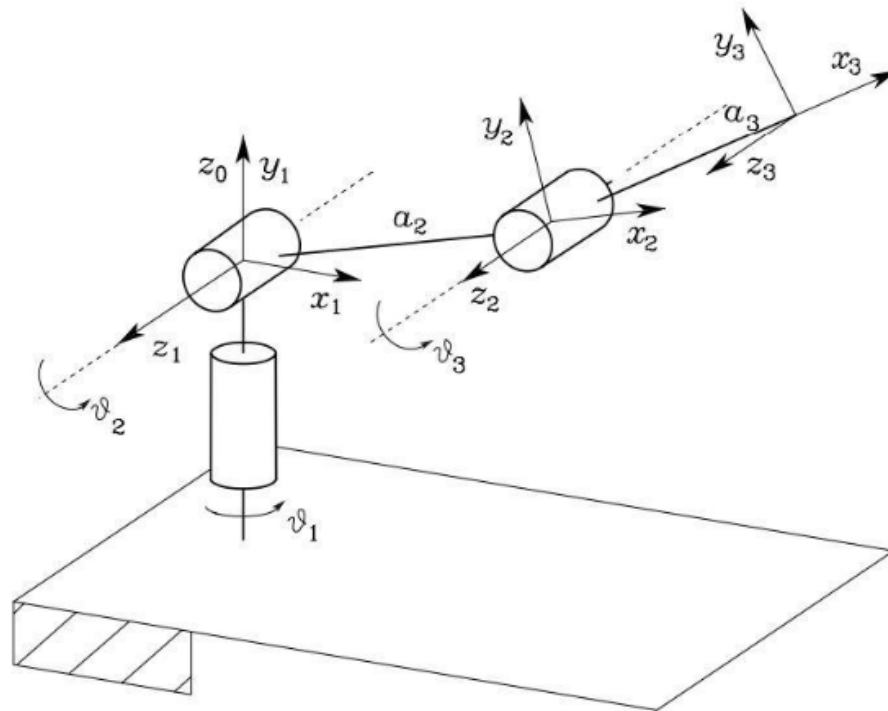
## 2 ĐỘNG HỌC THUẬN

### 2.1 Mô phỏng hình dạng Robot

#### 2.1.1 Xác định các thông số và lập bảng DH

Để có thể mô phỏng lại hình dạng robot, ta cần biết được vị trí của các khớp trong hệ toạ độ XYZ từ đó nối các điểm các khớp lại ta sẽ được hình dạng cơ bản của robot Articulated arm trong không gian.

Articulated arm là robot 3 bậc tự do, do đó, để có được vị trí của các khớp so với base, ta cần tính các ma trận biến đổi thuận nhất của các khớp so với base, từ đó lấy ra thông số vị trí và hướng của các khớp.



**Hình 3:** Articulated Arm

Đầu tiên ta cần thiết lập bảng DH, từ đó có thể thấy, các biến của robot Articulated arm là  $\theta_1$ ,  $\theta_2$  và  $\theta_3$

Link	$a_i$	$\alpha_i$	$d_i$	$\theta_i$
1	0	$\pi/2$	$d_1$	$\theta_1^*$
2	$a_2$	0	0	$\theta_2^*$
3	$a_3$	0	0	$\theta_3^*$



Sau khi có bảng DH, ta viết các hàm để tính toán ma trận biến đổi thuần nhất với input là các giá trị lấy từ ma trận DH.

```
1  # % Link | d | theta | a | alpha
2  # % 1     d1  theta_1  0   90
3  # % 2     0   theta_2  a2   0
4  # % 3     0   theta_3  a3   0
5
6  d = np.array([100, 0, 0])
7  a = np.array([0, 100, 100])
8  alpha = np.array([pi/2, 0, 0])
9  theta = np.array([0, 0, 0])
10 offsetTheta = np.array([0, 0, 0])
11
12 # Calculate DH matrix function
13 def DH_Matrix(d, theta, a, alpha):
14     c = cos(theta)
15     s = sin(theta)
16     ca = cos(alpha)
17     sa = sin(alpha)
18     return np.array([
19         [c, -ca*s, sa*s, a*c],
20         [s, c*ca, -sa*c, a*s],
21         [0, sa, ca, d],
22         [0, 0, 0, 1],
23     ])
24
25 # Calculate Tranpose matrix T0, T1, T2, T3 function
26 def updateTranposeMatrix():
27     T0 = np.eye(4)
28     T1 = np.dot(T0, DH_Matrix(d[0], theta[0]+offsetTheta[0], a[0], alpha[0]))
29     T2 = np.dot(T1, DH_Matrix(d[1], theta[1]+offsetTheta[1], a[1], alpha[1]))
30     T3 = np.dot(T2, DH_Matrix(d[2], theta[2]+offsetTheta[2], a[2], alpha[2]))
31     return [T0, T1, T2, T3]
```

### 2.1.2 Xây dựng môi trường giả lập Robot

Vì ở bài tập lớn này ta không sử dụng thư viện 3D mà sử dụng thư viện vẽ 2D là *Pygame* để vẽ robot nên ta phải dùng phép chiếu hình 3D lên màn hình 2D.

Ta sẽ tạo các biến *angleX*, *angleY* và *angleZ* để lưu lại các góc xoay lần lượt theo trục *x*, *y* và *z* (hay còn gọi là góc *Roll-Pitch-Yaw*) của tọa độ chuẩn môi trường giả lập robot so với góc nhìn của chúng ta, các góc trên sẽ thay đổi dựa vào tương tác của người dùng trên UI hoặc qua bàn phím, chuột, ... . Ta viết được hàm chiếu 2D đơn giản với input là 1 điểm trong hệ *Oxyz* sang *Oxy* (Vì mô phỏng robot đơn giản chỉ có các điểm và đường

*nên ta có thể bỏ qua z).*

```
1  scale = 1.5
2  angleX = -3*pi/4
3  angleY = 0
4  angleZ = -pi/4
5
6  def projective(pos):
7      pos = pos.reshape((3, 1))
8      rotation_z = np.array([
9          [cos(angleZ), -sin(angleZ), 0],
10         [sin(angleZ), cos(angleZ), 0],
11         [0, 0, -1],
12     ])
13     rotation_y = np.array([
14         [cos(angleY), 0, sin(angleY)],
15         [0, 1, 0],
16         [-sin(angleY), 0, cos(angleY)],
17     ])
18     rotation_x = np.array([
19         [1, 0, 0],
20         [0, cos(angleX), -sin(angleX)],
21         [0, sin(angleX), cos(angleX)],
22     ])
23     pos = np.dot(rotation_z, pos)
24     pos = np.dot(rotation_y, pos)
25     pos = np.dot(rotation_x, pos)
26     x = pos[0][0]
27     y = pos[1][0]
28     z = pos[2][0]
29     xx = int(x * scale) + circle_pos[0]
30     yy = int(y * scale) + circle_pos[1]
31     return (xx, yy)
```

Tiếp tục ta sẽ viết các hàm vẽ các điểm, đường, polygon và trục tọa độ *Oxyz* để hỗ trợ vẽ robot sau này.

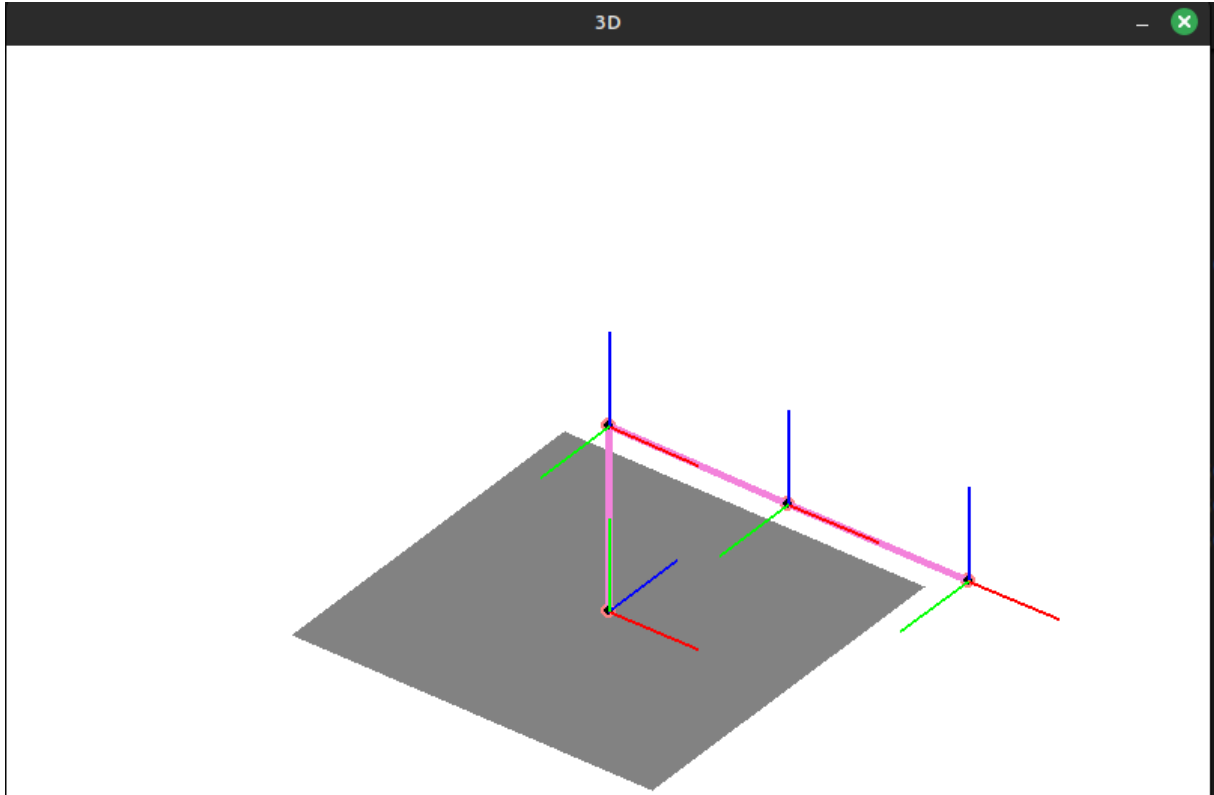
```
1  def drawCoor(screen, poso, posx, posy, posz):
2      o = projective(poso)
3      x = projective(posx)
4      y = projective(posy)
5      z = projective(posz)
6      pygame.draw.circle(screen, BLACK, o, 3)
7      pygame.draw.line(screen, RED, o, x, width=2)
8      pygame.draw.line(screen, BLUE, o, y, width=2)
9      pygame.draw.line(screen, GREEN, o, z, width=2)
```

```
10 def drawCircle(screen, pos, color=RED, radius=1, width=0):
11     pos = projective(pos)
12     pygame.draw.circle(screen, color, pos, radius, width)
13 def drawLine(screen, posx, posy, color=ORANGE, width=1):
14     x = projective(posx)
15     y = projective(posy)
16     pygame.draw.line(screen, color, x, y, width)
17 def drawPolygon(screen, posArray, color):
18     pygame.draw.polygon(screen, color, [projective(p) for p in posArray], 0)
```

### 2.1.3 Vẽ Robot

Dựa vào vị trí gốc (base) của robot và các ma trận biến đổi thuần nhất ta tính toán ra được các vị trí của các khớp xoay (joint) trong không gian và dùng hàm vẽ có được ở trên để vẽ ra được robot, các trục tọa độ và cả mặt đất bên dưới robot.

```
1 def drawMachine(opacity, T):
2     s = pygame.Surface((WIDTH, HEIGHT), pygame.SRCALPHA)
3     ground = np.array([[-100, -100, 0], [-100, 100, 0], [100, 100, 0], [100, -100,
4         0]])
5     drawPolygon(s, ground, (0, 0, 0, opacity))
6     drawCircle(s, DHtranspose(base0, T[0]), color=(255, 0, 0, opacity), radius=5)
7     drawCircle(s, DHtranspose(base0, T[1]), color=(255, 0, 0, opacity), radius=5)
8     drawCircle(s, DHtranspose(base0, T[2]), color=(255, 0, 0, opacity), radius=5)
9     drawCircle(s, DHtranspose(base0, T[3]), color=(255, 0, 0, opacity), radius=5)
10    drawLine(s, DHtranspose(base0, T[0]), DHtranspose(base0, T[1]), color=(230, 0, 180,
11        opacity), width=5)
12    drawLine(s, DHtranspose(base0, T[1]), DHtranspose(base0, T[2]), color=(230, 0, 180,
13        opacity), width=5)
14    drawLine(s, DHtranspose(base0, T[2]), DHtranspose(base0, T[3]), color=(230, 0, 180,
15        opacity), width=5)
16    screen.blit(s, (0, 0))
```



Hình 4: Vẽ robot

## 2.2 Xây dựng UI và Animation cho bài toán động lực thuận

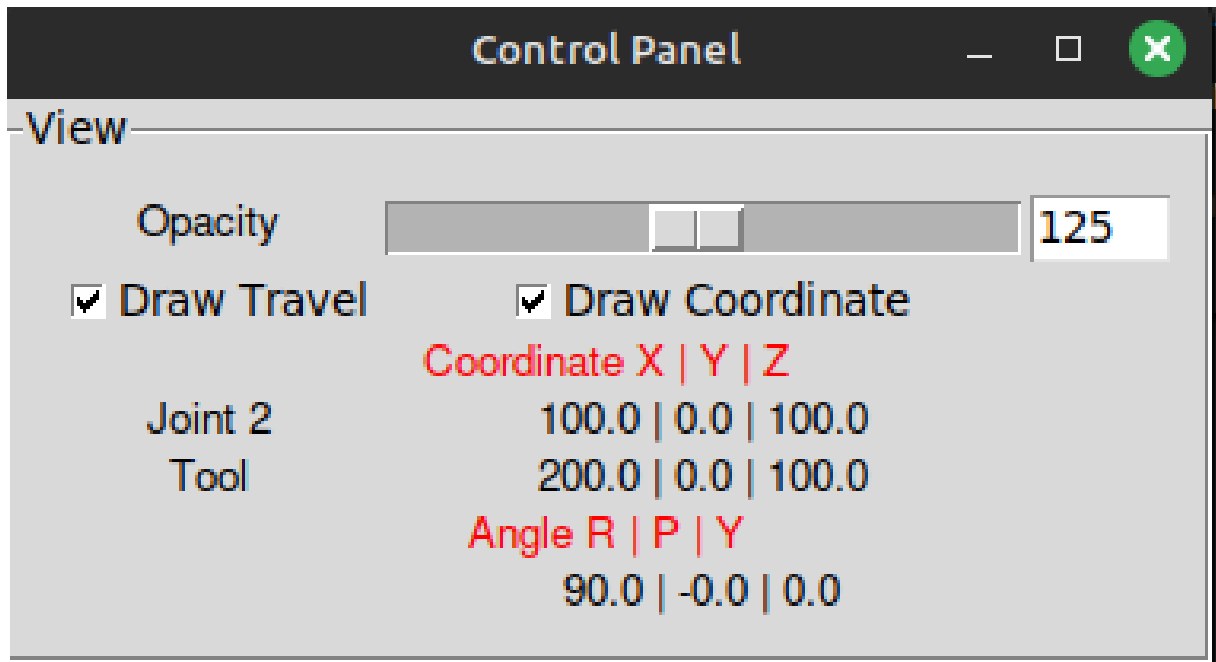
### 2.2.1 Giao diện người dùng UI

Sử dụng thư viện *Tkinter* để xây dựng giao diện người dùng. Xây dựng một cửa sổ để tùy chỉnh các thông số để vẽ robot và in các thông số cơ bản của robot như vị trí và góc xoay RPY của Tool (*End Effector*).

```

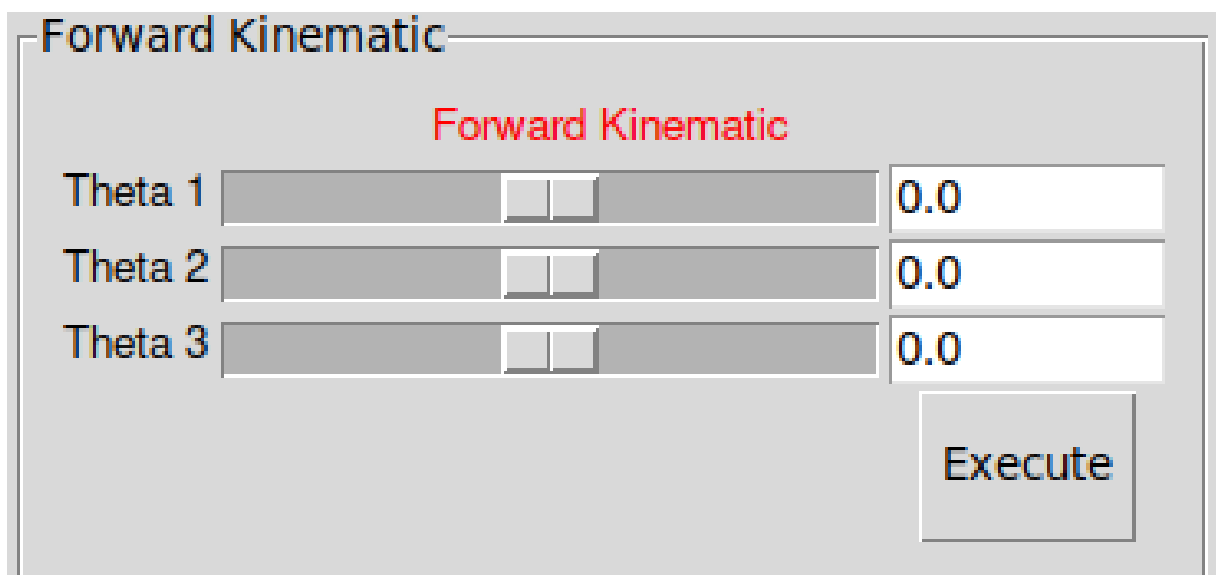
1  def calRPY(T):
2      pitch = atan2(-T[2][0], sqrt(T[2][1]**2 + T[2][2]**2))
3      if pitch == pi/2:
4          yaw = 0
5          rall = atan2(T[0][1], T[1][1])
6      elif pitch == -pi/2:
7          yaw = 0
8          rall = -atan2(T[0][1], T[1][1])
9      else:
10         yaw = atan2(T[1][0]/cos(pitch), T[0][0]/cos(pitch))
11         rall = atan2(T[2][1]/cos(pitch), T[2][2]/cos(pitch))
12     return str(round(rall*180/pi, 3)) + " | " + str(round(pitch*180/pi, 3)) + " | " +
           str(round(yaw*180/pi, 3))

```



Hình 5: Giao diện View

Xây dựng giao diện để người dùng nhập các góc  $\theta_1$ ,  $\theta_2$  và  $\theta_3$ , sau khi bấm nút *Execute* robot sẽ tiến hành di chuyển theo các góc  $\theta$  trên.



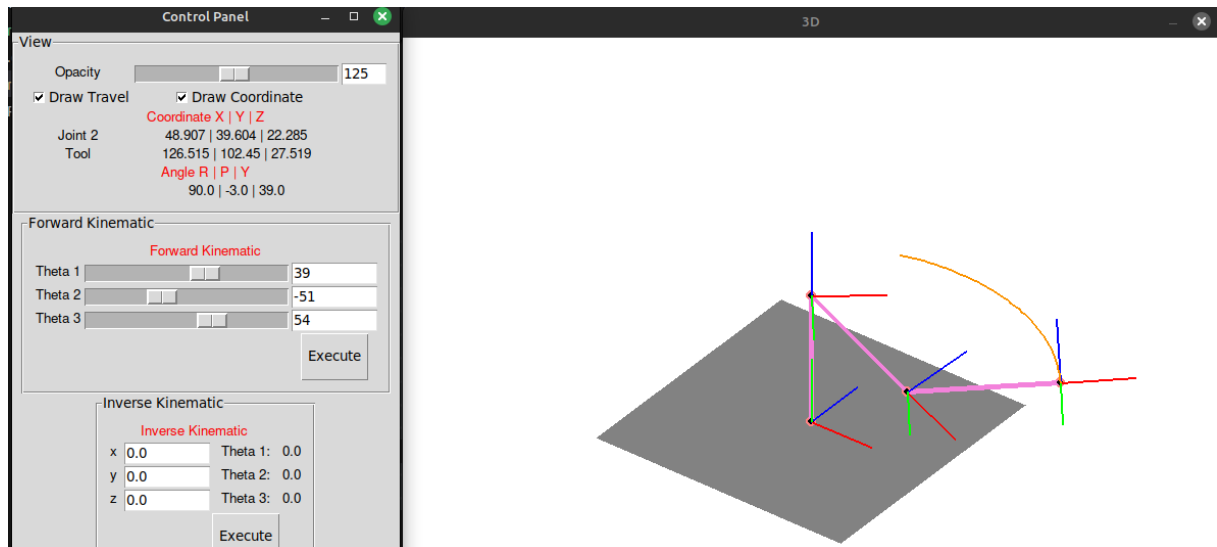
Hình 6: Giao diện Forward Kinematic

### 2.2.2 Tạo animation và di chuyển robot theo bài toán động lực thuận

Để tạo chuyển động cho robot ta sẽ lấy các giá trị góc  $\theta$  mới trên UI và trừ đi góc  $\theta$  hiện tại chia cho số bước ta muốn cho robot chuyển động sẽ ra được góc  $\delta\theta$  cho mỗi bước di chuyển. Lưu tất cả các giá trị này vào một mảng các góc  $\theta$  của robot sau mỗi bước di

chuyển và tiến hành vẽ lại robot sau mỗi bước ta sẽ tạo được chuyển động cho robot.

```
1  def forwardKine():
2      deltaTime = 60
3      nextTheta = np.array([guiTheta[0].get()*pi/180, guiTheta[1].get()*pi/180,
4                           guiTheta[2].get()*pi/180])
5      stepAngle = np.array([(nextTheta[0]-theta[0])/deltaTime, (nextTheta[1]-theta[1])/
6                           deltaTime, (nextTheta[2]-theta[2])/deltaTime])
7      animateTranform(nextTheta, stepAngle, deltaTime)
8
9  def animateTranform(nextTheta, stepAngle, deltaTime):
10     global animateArray, travelArray
11     temp = np.array([0, 0, 0])
12     for j in range(deltaTime):
13         temp = temp+stepAngle
14         animateArray = np.append(animateArray, theta+temp)
15         animateArray = np.append(animateArray, nextTheta)
16         animateArray = np.reshape(animateArray, (-1,3))
17         travelArray = np.array([])
18
19     def updateTheta():
20         global theta, animateArray, travelArray
21         if animateArray.size > 0:
22             theta = animateArray[0]
23             animateArray = animateArray[1:]
24             T = updateTranposeMatrix()
25             for i in range(2):
26                 guiCoor[i].set(coor2str(DHtranspose(base0, T[i+2])))
27                 guiRPY.set(calRPY(T[3]))
28
29             travelArray = np.append(travelArray, DHtranspose(base0, T[3]))
30     return T
```



Hình 7: Forward Kinematic



## 3 ĐỘNG HỌC NGHỊCH

### 3.1 DH-Table





## 4 QUY HOẠCH QUỶ ĐẠO TRAJECTORY PLANNING

### 4.1 DH-Table



## 5 MA TRẬN JACOBI VÀ ĐIỂM KÌ DỊ

### 5.1 DH-Table

## 6 Path Finding

### 6.1 Sơ lược về bài toán

Bài toán Path Finding là một trong những bài toán cơ bản trong lĩnh vực Trí tuệ Nhân tạo, với mục tiêu tìm đường đi ngắn nhất từ một điểm khởi đầu (*start*) đến một điểm kết thúc (*goal*) trên một bản đồ hoặc đồ thị.

Giải thuật A\* (*A-star*) được lựa chọn để giải quyết bài toán này nhờ vào tính hiệu quả cao trong việc kết hợp giữa hai phương pháp: tìm kiếm tham lam (*Greedy Search*) và tìm kiếm chi phí đều (*Uniform Cost Search*). Giải thuật này sử dụng công thức:

$$f(n) = g(n) + h(n)$$

Trong đó:

- $f(n)$ : Hàm lượng giá tổng hợp.
- $g(n)$ : Chi phí từ điểm bắt đầu đến trạng thái  $n$ .
- $h(n)$ : Heuristic (ước lượng chi phí từ trạng thái  $n$  đến đích).

Trong bài toán này, nhóm lựa chọn sẽ áp dụng giải thuật A\* để tìm đường đi ngắn nhất từ điểm bắt đầu đến điểm đích trên một bản đồ lưới 2D.

### 6.2 Những đánh giá về game

Path Finding là một bài toán không chỉ mang tính thực tiễn cao mà còn chứa đựng nhiều khía cạnh học thuật quan trọng trong lĩnh vực Trí tuệ Nhân tạo. Đây là nền tảng của nhiều ứng dụng thực tế như:

- Định tuyến trong giao thông vận tải, các ứng dụng bản đồ.
- Điều hướng cho các nhân vật trong các trò chơi điện tử.
- Điều hướng robot trong lĩnh vực Robotics.

Tuy nhiên, việc giải quyết bài toán này có thể bị ảnh hưởng bởi độ lớn của không gian trạng thái, độ phức tạp của các phép di chuyển, và chất lượng của hàm lượng giá. Một heuristic không phù hợp có thể dẫn đến kết quả kém tối ưu, hoặc khiến thuật toán phải duyệt qua quá nhiều trạng thái.

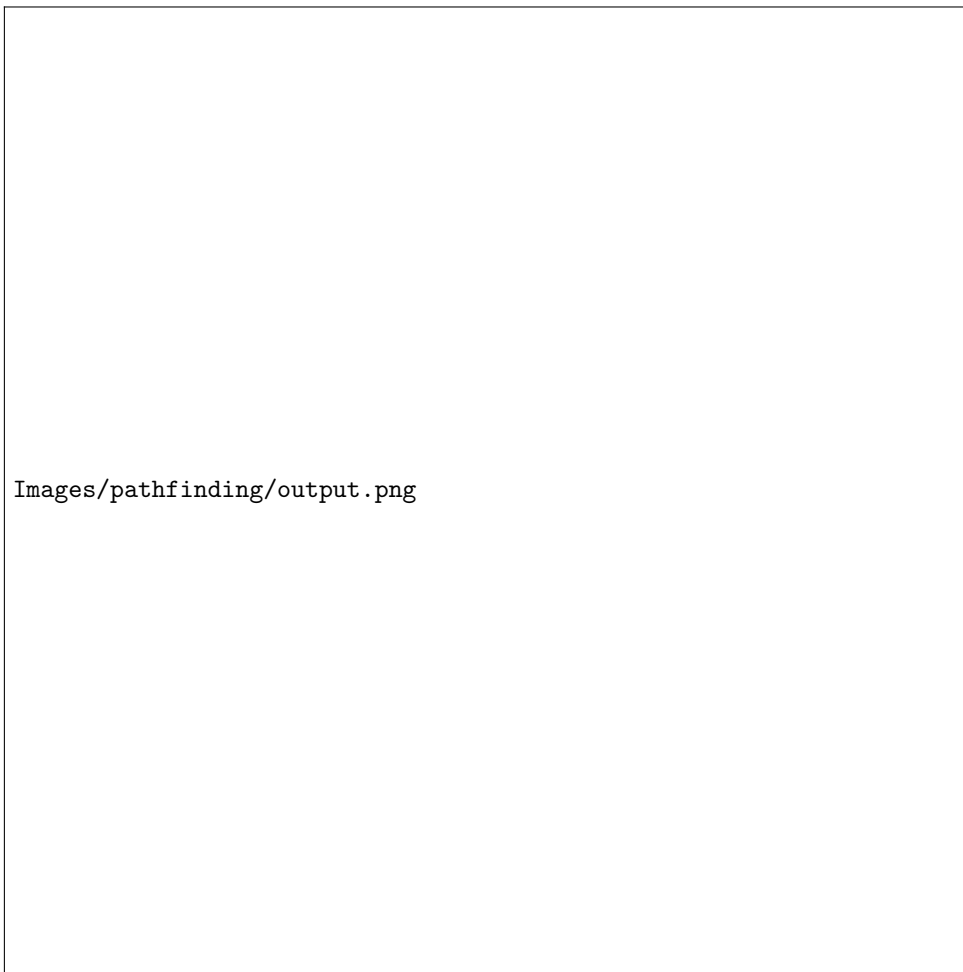
Giải thuật  $A^*$  với cơ chế tối ưu hoá chi phí bằng cách kết hợp hàm  $g(n)$  (chi phí đã đi) và hàm  $h(n)$  (ước lượng chi phí còn lại) là một giải pháp mạnh mẽ. Việc đánh giá và cải thiện heuristic phù hợp là yếu tố then chốt trong việc nâng cao hiệu quả của thuật toán.

### 6.3 Giao diện trực quan hóa

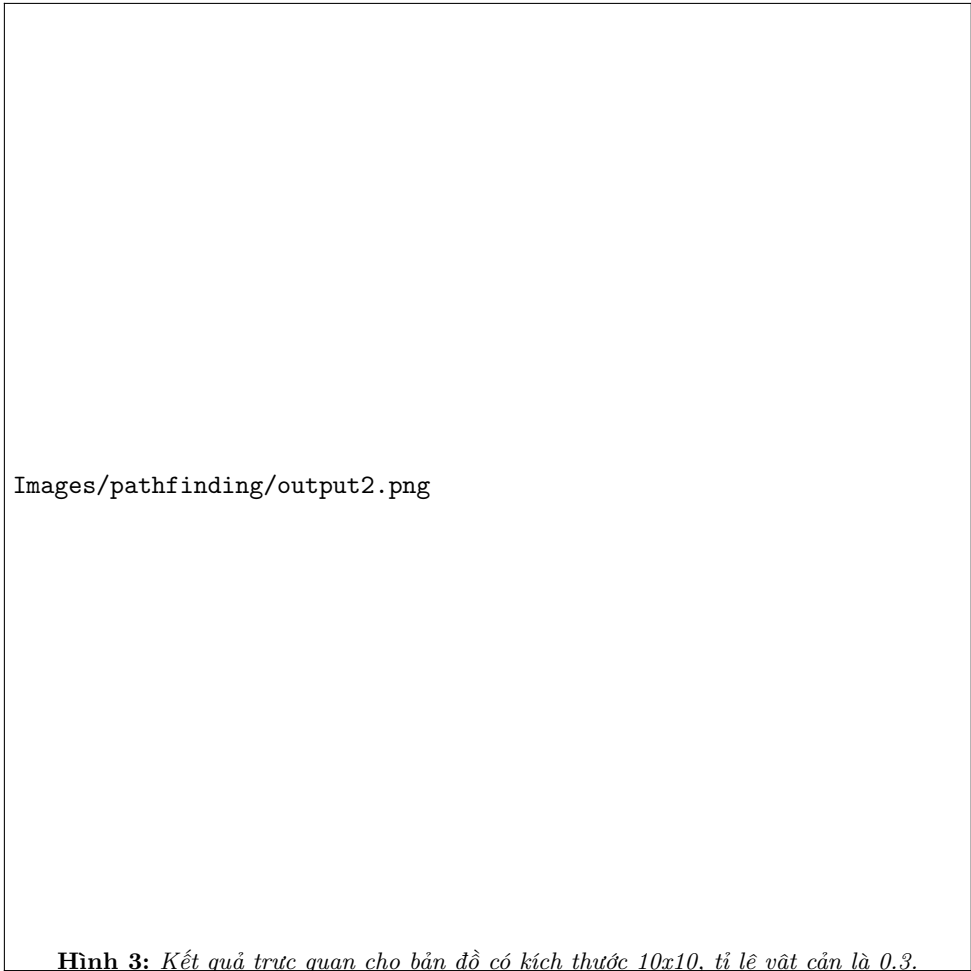
Để trực quan hóa kết quả của thuật toán  $A^*$ , nhóm em đã xây dựng một giao diện minh họa bằng Python, sử dụng thư viện `matplotlib` để vẽ bản đồ và đường đi. Dưới đây là chi tiết về cách thức hoạt động của giao diện:

- Mỗi ô trên bản đồ được hiển thị bằng một màu sắc khác nhau, tương ứng với các trạng thái:
  - **0: Passage (đường đi trống)** - màu đen.
  - **1: Wall (vật cản)** - màu tím đậm.
  - **2: Start (điểm bắt đầu)** - màu đỏ.
  - **3: Destination (điểm đích)** - màu cam.
  - **4: Path (đường đi tìm được)** - màu vàng nhạt.
- Giao diện có thể tạo tùy chỉnh một bản đồ kích thước  $width \times height$ , với tỷ lệ vật cản  $wallrate$ . Với điểm bắt đầu và điểm đích nếu không được tùy chỉnh thì sẽ được sinh ngẫu nhiên.
- Các đường đi được vẽ bằng cách tô màu các ô thuộc đường đi ngắn nhất từ điểm bắt đầu đến điểm đích.

Dưới đây là một vài kết quả trực quan hóa:



**Hình 8:** Kết quả trực quan cho bản đồ có kích thước  $20 \times 20$ , tỉ lệ vật cản là 0.3.



Images/pathfinding/output2.png

**Hình 9:** Kết quả trực quan cho bản đồ có kích thước  $10 \times 10$ , tỉ lệ vật cản là 0.3.

## 6.4 Xây dựng các cấu trúc để mô phỏng bài toán

Ở trò chơi này, nhóm em tổ chức các đoạn mã Python trên các tệp tin gồm Map.py, Astar.py, visualize.ipynb và có một file map1.txt dành cho trường hợp muốn tùy chỉnh sâu cấu trúc bản đồ lưới 2D: Để giải quyết bài toán tìm đường (Path Finding) bằng giải thuật A\*, nhóm em đã triển khai hai file Python chính là `astar.py` và `map.py`.

### 6.4.1 Cấu trúc file `astar.py`

Listing 1: Định nghĩa các lớp trong file `astar.py`

```
1 class Cell:
2     def __init__(self):
3 class AStar:
4     def __init__(self, map, movedir=4):
5     def Search(self):
6     def PrintPath(self):
7     def UpdateGrid(self):
```

#### Giải thích:

- `class Cell`: Đại diện cho từng ô trong bản đồ.
  - Thuộc tính:
    - \* `parent_x`, `parent_y`: Tọa độ của ô cha.
    - \* `f`, `g`, `h`: Các giá trị tổng chi phí, chi phí đến ô, và ước lượng đến đích.
- `class AStar`: Thực hiện thuật toán A\*.
  - Phương thức:
    - \* `Search`: Triển khai thuật toán A\* để tìm đường đi.
    - \* `PrintPath`: In đường đi từ điểm đầu đến điểm đích.
    - \* `UpdateGrid`: Cập nhật bản đồ với đường đi tìm được.

### 6.4.2 Cấu trúc file `map.py`

Listing 2: Định nghĩa các lớp trong file `map.py`

```
1 class Map:
2     def CreateWall(self, wall_rate=0.3):
3     def IsValid(self, x, y):
4     def IsBlock(self, x, y):
5     def CalculateH(self, x, y, algorithm):
```

### Giải thích:

- **class Map:** Đại diện cho bản đồ trong bài toán.
  - **Thuộc tính:**
    - \* **width, height:** Kích thước bản đồ.
    - \* **grid:** Ma trận biểu diễn bản đồ.
    - \* **src\_point, des\_point:** Tọa độ điểm bắt đầu và điểm kết thúc.
  - **Phương thức:**
    - \* **CreateWall:** Tạo các vật cản ngẫu nhiên trên bản đồ.
    - \* **IsValid:** Kiểm tra xem tọa độ có hợp lệ không.
    - \* **IsBlock:** Kiểm tra xem tọa độ có bị chặn bởi vật cản (tường) không.
    - \* **IsDestination:** Kiểm tra xem tọa độ có phải điểm đích không.
    - \* **CalculateH:** Tính toán heuristic theo thuật toán Manhattan hoặc Euclidean.

## 6.5 Định nghĩa không gian trạng thái

Trong bài toán Path Finding, không gian trạng thái được định nghĩa như sau:

- **Trạng thái:** Mỗi trạng thái trong không gian trạng thái là một tọa độ  $(x, y)$  trên bản đồ lưới 2D kích thước  $n \times m$ , trong đó:
  - $x$ : Chỉ số hàng (row index).
  - $y$ : Chỉ số cột (column index).
- **Trạng thái khởi đầu (Start State):** Là tọa độ ban đầu  $S = (x_s, y_s)$ , nơi bắt đầu tìm kiếm đường đi.
- **Trạng thái kết thúc (Goal State):** Là tọa độ đích  $G = (x_g, y_g)$ , nơi cần tìm đến.
- **Trạng thái cản trở (Obstacle):** Các ô không thể đi qua được trên bản đồ, ký hiệu là  $W$  (Wall).
- **Trạng thái trung gian (Intermediate State):** Các ô có thể đi qua trong quá trình tìm kiếm, ký hiệu là  $P$  (Passage).



**Không gian trạng thái:** Không gian trạng thái là tập hợp tất cả các tọa độ  $(x, y)$  khả thi trên bản đồ, được định nghĩa là:

$$\mathcal{S} = \{(x, y) \mid 0 \leq x < n, 0 \leq y < m, (x, y) \notin W\}$$

**Hàm chuyển trạng thái (State Transition Function):** Các trạng thái chuyển đổi được định nghĩa bởi các quy tắc di chuyển:

- Di chuyển sang trái:  $(x, y) \rightarrow (x, y - 1)$
- Di chuyển sang phải:  $(x, y) \rightarrow (x, y + 1)$
- Di chuyển lên:  $(x, y) \rightarrow (x - 1, y)$
- Di chuyển xuống:  $(x, y) \rightarrow (x + 1, y)$

Các phép di chuyển này chỉ khả thi nếu tọa độ mới  $(x', y')$  thỏa mãn:

$$(x', y') \in \mathcal{S}.$$

**Hàm lượng giá (Heuristic Function):** Để tối ưu hóa quá trình tìm kiếm, nhóm em sử dụng hàm lượng giá  $h(n)$  để ước lượng chi phí từ trạng thái hiện tại đến trạng thái đích:

**Manhattan Distance:**

$$h(x, y) = |x - x_g| + |y - y_g|$$

Ngoài ra để mở rộng bài toán cho trường hợp Agent di chuyển được cả hướng chéo thì sẽ dùng hàm lượng giá:

**Euclidean Distance:**

$$h(x, y) = \sqrt{(x - x_g)^2 + (y - y_g)^2}$$

Với định nghĩa trên, không gian trạng thái và các phép chuyển đổi đã được thiết lập để áp dụng giải thuật A\*.

## 6.6 A\* Search

- **Pseudocode:**

Thuật toán A\* được triển khai như sau:

function AStarSearch(start, goal, map):

1. Khởi tạo danh sách mở (Open List) chứa ô bắt đầu, với  $f = 0$ .
2. Khởi tạo danh sách đóng (Closed List) rỗng.

While Open List không rỗng:

3. Lấy ô trong danh sách mở có  $f$  nhỏ nhất làm ô hiện tại.
4. Nếu ô hiện tại là đích, dừng thuật toán và trả về đường đi.
5. Duyệt qua các ô lân cận của ô hiện tại:
  - a. Nếu ô không hợp lệ hoặc là vật cản, bỏ qua.
  - b. Nếu ô đã được xét trong danh sách đóng, bỏ qua.
  - c. Tính giá trị  $g$ ,  $h$ , và  $f$  cho ô lân cận:
    - $g$  = chi phí từ điểm bắt đầu đến ô lân cận.
    - $h$  = ước lượng chi phí từ ô lân cận đến đích (heuristic).
    - $f = g + h$ .
  - d. Nếu ô lân cận chưa có trong danh sách mở hoặc có  $f$  nhỏ hơn, cập nhật danh sách mở.
6. Thêm ô hiện tại vào danh sách đóng.
7. Nếu danh sách mở rỗng nhưng chưa tìm được đích, trả về thất bại.

### • Áp dụng vào bài toán và kết quả:

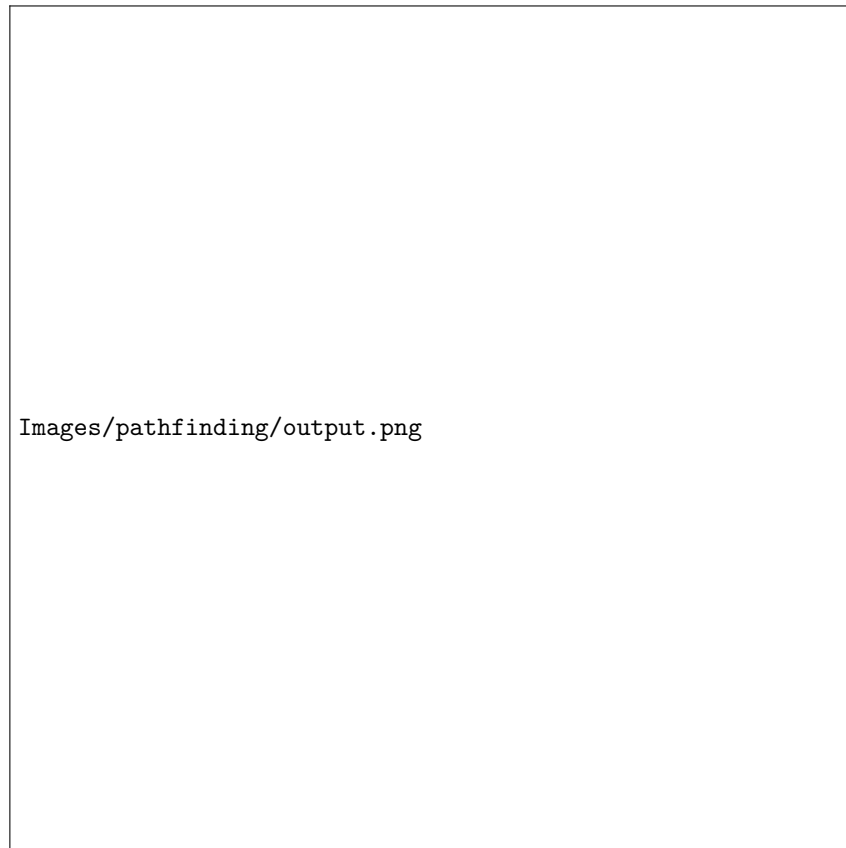
Thuật toán A\* được áp dụng để giải bài toán tìm đường trên bản đồ lưới  $20 \times 20$  với tỷ lệ vật cản là 30%.

#### – Các bước chính:

1. **Khởi tạo danh sách mở và danh sách đóng:** - Danh sách mở (*Open List*) chứa các ô cần được xét. - Danh sách đóng (*Closed List*) chứa các ô đã xét.
2. **Tính toán các giá trị  $f, g, h$ :**
  - $g(n)$ : Chi phí từ điểm bắt đầu đến ô hiện tại.
  - $h(n)$ : Heuristic dựa trên khoảng cách Manhattan hoặc Euclidean.
  - $f(n) = g(n) + h(n)$ .
3. **Tìm ô có giá trị  $f$  nhỏ nhất:** Duyệt các ô xung quanh (theo 4 hoặc 8 hướng) và cập nhật danh sách mở.

4. **Lưu lại đường đi tối ưu:** Truy xuất từ điểm đích ngược về điểm bắt đầu bằng cách sử dụng `parent_x` và `parent_y`.

– **Minh họa kết quả:** Sau khi áp dụng thuật toán, đường đi tối ưu được đánh dấu trên bản đồ với màu vàng.



**Hình 10:** Kết quả đường đi tìm được bằng thuật toán A\*.

– **Đánh giá:**

- \* Thuật toán tìm được đường đi ngắn nhất với chi phí tối ưu.
- \* Áp dụng heuristic Manhattan giúp giảm số ô cần xét trong danh sách mở.
- \* Với bản đồ có kích thước lớn, bộ nhớ sử dụng cho danh sách mở và đóng có thể tăng đáng kể.

## 7 ĐÁNH GIÁ CHUNG VÀ KẾT LUẬN

Sau thời gian thực hiện bài tập lớn số 1 môn Nhập môn Trí tuệ nhân tạo, nhóm có những đánh giá và kết luận chung sau khi hoàn thành bài tập lớn này như sau :

- Về những kết quả thu được:
  - Biết cách mô phỏng bài toán cũng như kết quả dưới dạng các ứng dụng đơn giản trên nền tảng ngôn ngữ Python 3 bằng cách áp dụng linh hoạt các module cũng như sử dụng các framework hiệu quả.
  - Hiểu được cấu trúc và những thành phần cơ bản của các bài toán để từ đó xây dựng được các cấu trúc để mô phỏng bài toán.
  - Đã hiểu rõ cũng như biết cách áp dụng 4 giải thuật tìm kiếm cơ bản đó là Depth-First Search, Genetic, A\* Search vào để tìm lời giải cho bài toán.
  - Cải thiện được khả năng lập trình bằng ngôn ngữ Python 3, tận dụng các module sẵn có vô cùng hữu ích của ngôn ngữ này.
  - Nâng cao khả năng làm việc theo nhóm cũng như khả năng hoàn thành công việc độc lập và đúng thời gian của từng thành viên trong nhóm.
- Về những hạn chế còn tồn đọng:
  - Cải thiện được khả năng lập trình bằng ngôn ngữ Python 3, tận dụng các module sẵn có vô cùng hữu ích của ngôn ngữ này.

Qua những đánh giá và kết luận chung như trên, các thành viên của nhóm đều nhận thấy rằng bản thân đã học thêm được khá nhiều điều mới lạ và đã rèn dũa thêm được những kỹ năng sẵn có. Tuy vậy thì nhóm cũng nhận ra được vẫn còn khá nhiều khuyết điểm quan trọng cần được khắc phục ngay để có thể thực hiện những dự án khác trong tương lai một cách đầy đủ và hoàn thiện hơn.

## 8 VIDEO DEMO VÀ CODE PYTHON

Link video thuyết trình: [Nền tảng Youtube](#)

Link slide thuyết trình: [Nền tảng Canva](#)



## Tài liệu