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

1 GIỚI THIỆU **2**

1.1 Sơ lược về Articulated Arm Robot 2

1.2 Giới thiệu về ngôn ngữ lập trình Python 3

2 ĐỘNG HỌC THUẬN **6**

2.1 Mô phỏng hình dạng Robot 6

2.1.1 Xác định các thông số và lập bảng DH 6

2.1.2 Xây dựng môi trường giả lập Robot 7

2.1.3 Vẽ Robot 9

2.2 Xây dựng UI và Animation cho bài toán động lực thuận 10

2.2.1 Giao diện người dùng UI 10

2.2.2 Tạo animation và di chuyển robot theo bài toán động lực thuận . . 11

3 ĐỘNG HỌC NGHỊCH **14**

3.1 Lập công thức giải bài toán động học nghịch 14

3.2 Xây dựng UI và Animation cho bài toán động lực nghịch 14

3.2.1 Giao diện người dùng UI 14

3.2.2 Tạo animation và di chuyển robot theo bài toán động lực nghịch . . 15

4 QUY HOẠCH QUỸ ĐẠO TRAJECTORY PLANNING **17**

4.1 Linear Segment with Parabolic Blend ($LSPB$) 17

4.2 Kết quả chạy được 18

5 ĐÁNH GIÁ CHUNG VÀ KẾT LUẬN **19**

6 VIDEO DEMO VÀ CODE PYTHON **19**

Tài liệu tham khảo **19**

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

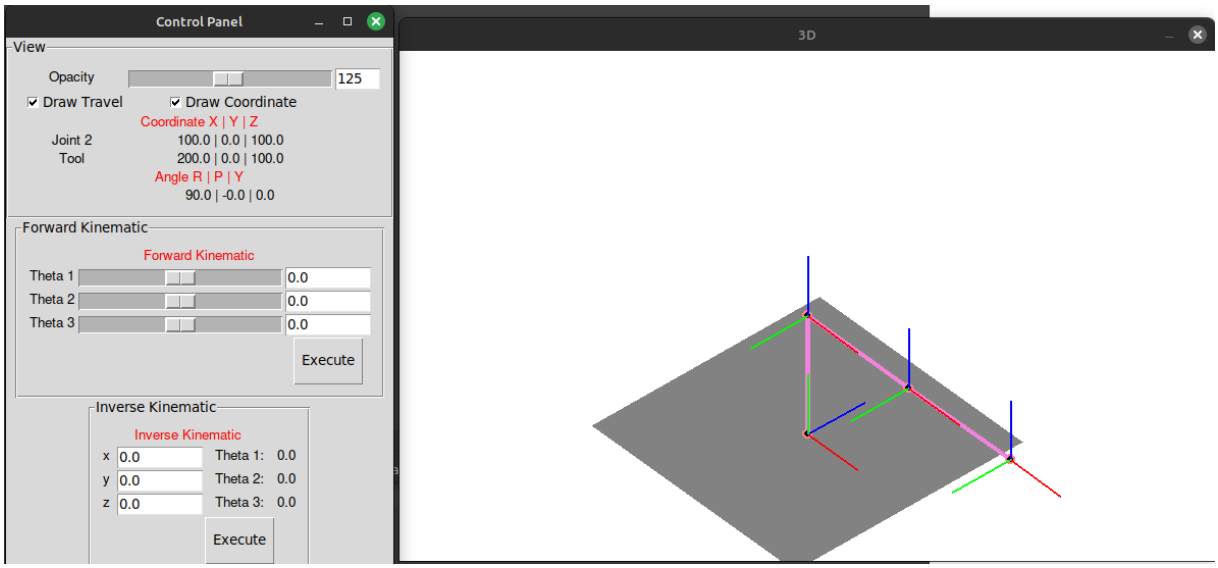
Python là một ngôn ngữ lập trình bậc cao, được phát triển bởi Guido van Rossum và lần đầu tiên được phát hành vào năm 1991. Nó nổi bật với cú pháp rõ ràng, dễ đọc và dễ học, giúp lập trình viên có thể phát triển ứng dụng một cách nhanh chóng và hiệu quả.
Các đặc điểm nổi bật của Python:

- **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 viên 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*

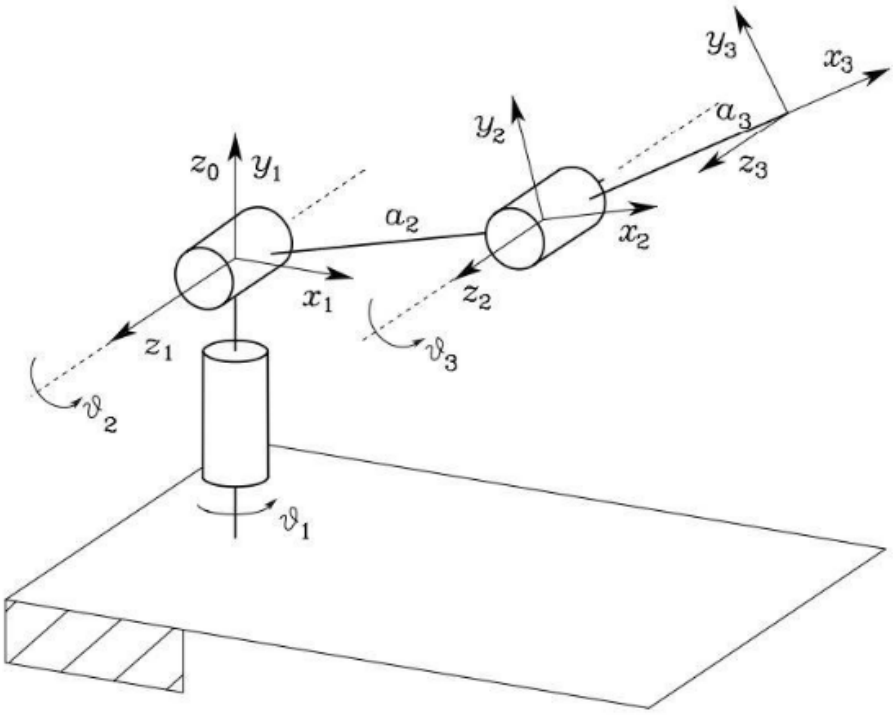
2 ĐỘNG HỌC THUẬN

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 θ_1 , θ_2 và θ_3

Link	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	d_1	θ_1^*
2	a_2	0	0	θ_2^*
3	a_3	0	0	θ_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.

```
# % Link | d    | theta         |    a    | alpha
# % 1       d1     theta_1           0       90
# % 2       0      theta_2          a2        0
# % 3       0      theta_3          a3        0
d = np.array([100, 0, 0])
a = np.array([0, 100, 100])
alpha = np.array([pi/2, 0, 0])
theta = np.array([0, 0, 0])
offsetTheta = np.array([0, 0, 0])

# Calculate DH matrix function
def DH_Matrix(d, theta, a, alpha):
c = cos(theta)
s = sin(theta)
ca = cos(alpha)
sa = sin(alpha)
return np.array([
[c, -ca*s, sa*s, a*c],
[s, c*ca, -sa*c, a*s],
[0, sa, ca, d],
[0, 0, 0, 1],
])

# Calculate Tranpose matrix T0, T1, T2, T3 function
def updateTranposeMatrix():
T0 = np.eye(4)
T1 = np.dot(T0, DH_Matrix(d[0], theta[0]+offsetTheta[0], a[0], alpha[0]))
T2 = np.dot(T1, DH_Matrix(d[1], theta[1]+offsetTheta[1], a[1], alpha[1]))
T3 = np.dot(T2, DH_Matrix(d[2], theta[2]+offsetTheta[2], a[2], alpha[2]))
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 viên 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).

```
scale = 1.5
angleX = -3*pi/4
angleY = 0
angleZ = -pi/4

def projective(pos):
pos = pos.reshape((3, 1))
rotation_z = np.array([
[cos(angleZ), -sin(angleZ), 0],
[sin(angleZ), cos(angleZ), 0],
[0, 0, -1],
])
rotation_y = np.array([
[cos(angleY), 0, sin(angleY)],
[0, 1, 0],
[-sin(angleY), 0, cos(angleY)],
])
rotation_x = np.array([
[1, 0, 0],
[0, cos(angleX), -sin(angleX)],
[0, sin(angleX), cos(angleX)],
])
pos = np.dot(rotation_z, pos)
pos = np.dot(rotation_y, pos)
pos = np.dot(rotation_x, pos)
x = pos[0][0]
y = pos[1][0]
z = pos[2][0]
xx = int(x * scale) + circle_pos[0]
yy = int(y * scale) + circle_pos[1]
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.

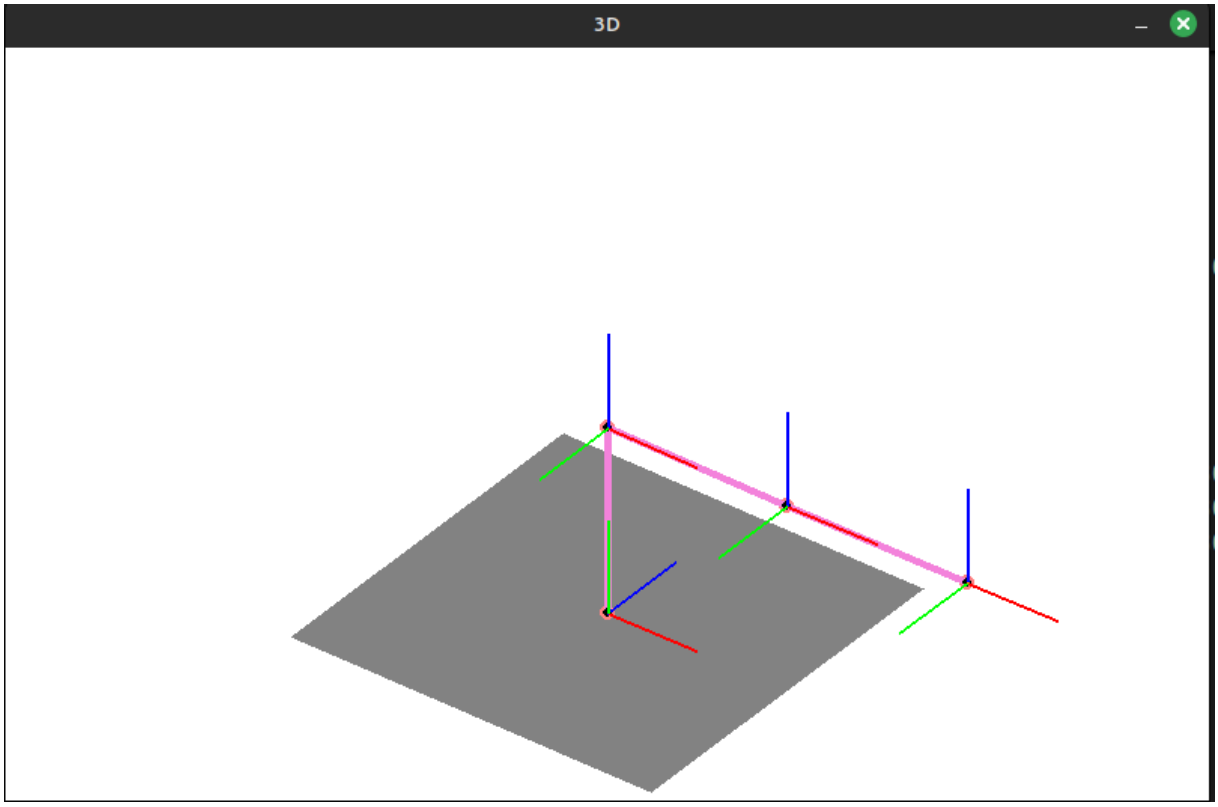
```
def drawCoor(screen, poso, posx, posy, posz):
o = projective(poso)
x = projective(posx)
y = projective(posy)
z = projective(posz)
pygame.draw.circle(screen, BLACK, o, 3)
pygame.draw.line(screen, RED, o, x, width=2)
pygame.draw.line(screen, BLUE, o, y, width=2)
pygame.draw.line(screen, GREEN, o, z, 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 và 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,
        0]])
4   drawPolygon(s, ground ,(0,0,0,opacity))
5   drawCircle(s, DHtranspose(base0, T[0]), color=(255,0,0,opacity), radius=5)
6   drawCircle(s, DHtranspose(base0, T[1]), color=(255,0,0,opacity), radius=5)
7   drawCircle(s, DHtranspose(base0, T[2]), color=(255,0,0,opacity), radius=5)
8   drawCircle(s, DHtranspose(base0, T[3]), color=(255,0,0,opacity), radius=5)
9   drawLine(s, DHtranspose(base0, T[0]), DHtranspose(base0, T[1]), color=(230,0,180,
        opacity), width=5)
10  drawLine(s, DHtranspose(base0, T[1]), DHtranspose(base0, T[2]), color=(230,0,180,
        opacity), width=5)
11  drawLine(s, DHtranspose(base0, T[2]), DHtranspose(base0, T[3]), color=(230,0,180,
        opacity), width=5)
12  screen.blit(s, (0,0))
```



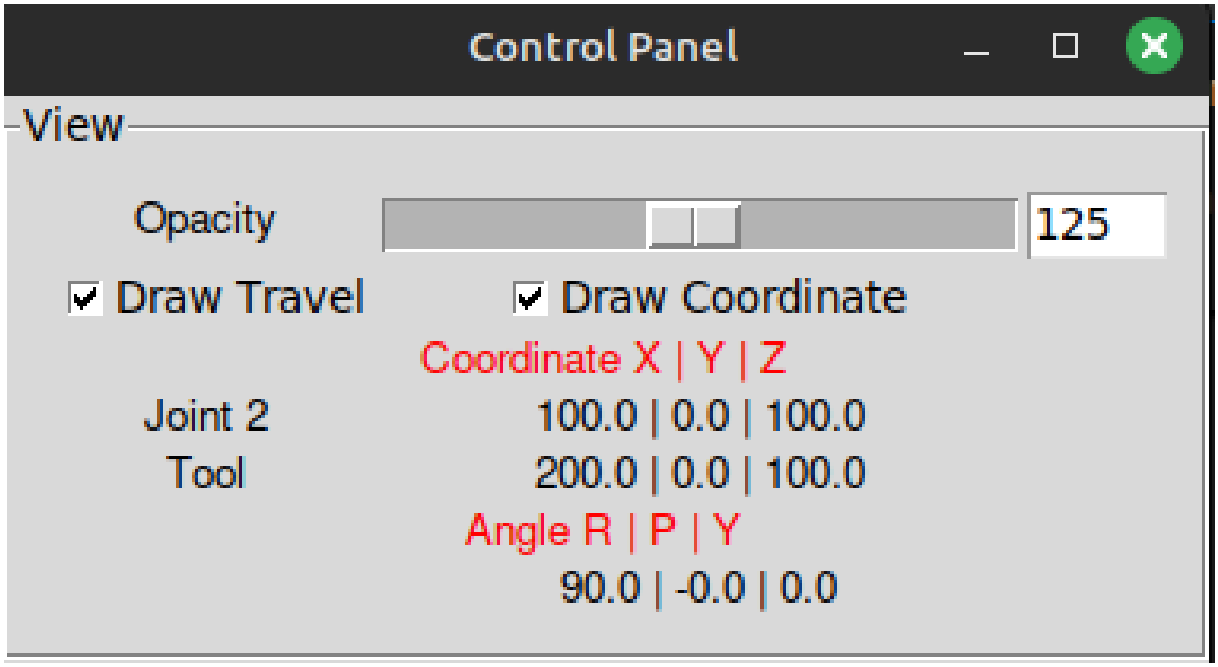
Hình 4: *Vẽ robot*

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

2.2.1 Giao diện người dùng UI

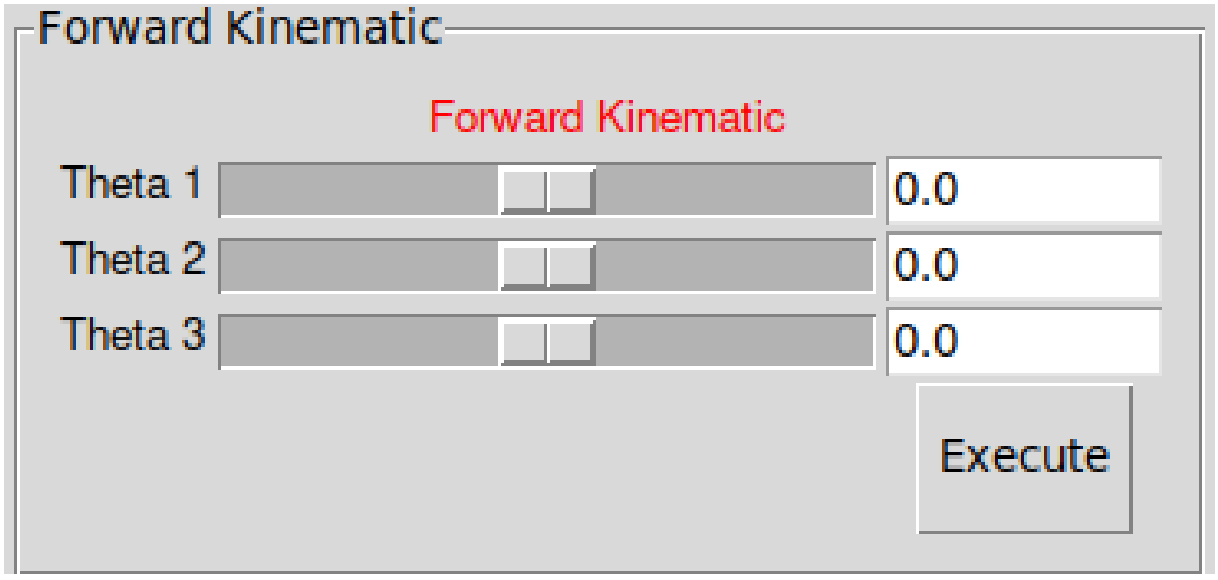
Sử dụng thư viên *Tkinter* để xây dựng giao diện người dùng. Xây dựng một cửa sổ để tùy chỉnh các thông số đẽ vẽ robot và in các thông số cơ bản của robot như vị trí và góc xoay RPY của Tool (*End Effector*).

```
def calRPY(T):
pitch = atan2(-T[2][0], sqrt(T[2][1]**2 + T[2][2]**2))
if pitch == pi/2:
yaw = 0
rall = atan2(T[0][1], T[1][1])
elif pitch == -pi/2:
yaw = 0
rall = -atan2(T[0][1], T[1][1])
else:
yaw = atan2(T[1][0]/cos(pitch), T[0][0]/cos(pitch))
rall = atan2(T[2][1]/cos(pitch), T[2][2]/cos(pitch))
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 θ_1 , θ_2 và θ_3, sau khi bấm nút *Execute* robot sẽ tiến hành di chuyển theo các góc θ trên.



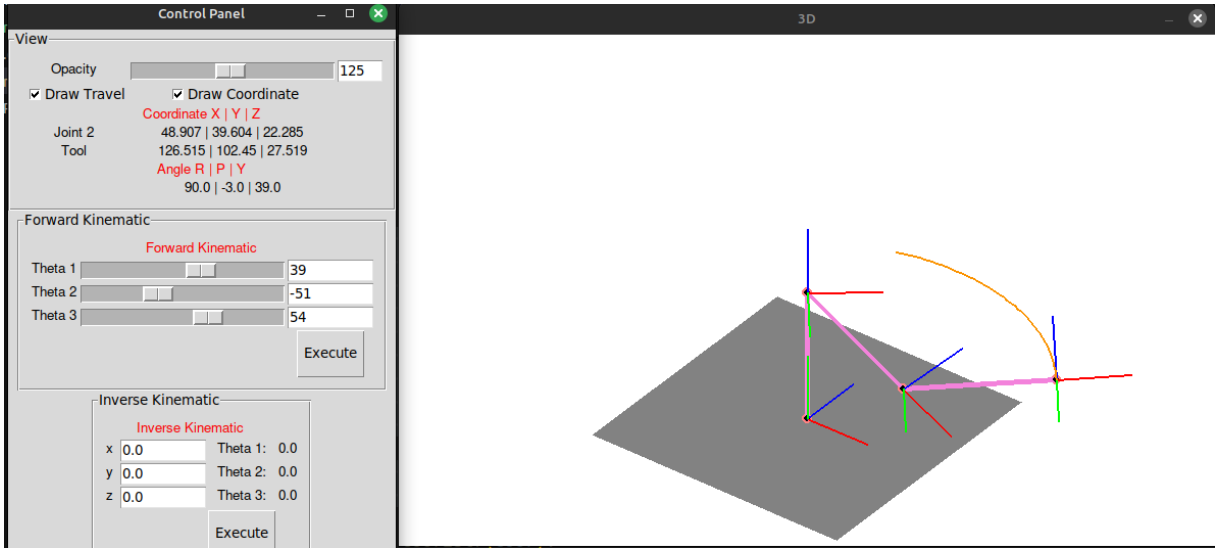
Hình 6: *Giao diện Forward Kinematic*

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 θ mới trên UI và trừ đi góc θ 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 θ của robot sau mỗi bước di

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

```
def forwardKine():
deltaTime = 60
nextTheta = np.array([guiTheta[0].get()*pi/180, guiTheta[1].get()*pi/180,
    guiTheta[2].get()*pi/180])
stepAngle = np.array([(nextTheta[0]-theta[0])/deltaTime, (nextTheta[1]-theta[1])/
    deltaTime, (nextTheta[2]-theta[2])/deltaTime])
animateTranform(nextTheta, stepAngle, deltaTime)

def animateTranform(nextTheta, stepAngle, deltaTime):
global animateArray, travelArray
temp = np.array([0, 0, 0])
for j in range(deltaTime):
temp = temp+stepAngle
animateArray = np.append(animateArray, theta+temp)
animateArray = np.append(animateArray, nextTheta)
animateArray = np.reshape(animateArray, (-1,3))
travelArray = np.array([])

def updateTheta():
global theta, animateArray, travelArray
if animateArray.size > 0:
theta = animateArray[0]
animateArray = animateArray[1:]
T = updateTranposeMatrix()
for i in range(2):
guiCoor[i].set(coor2str(DHtranspose(base0, T[i+2])))
guiRPY.set(calRPY(T[3]))

travelArray = np.append(travelArray, DHtranspose(base0, T[3]))
return T
```



Hình 7: *Forward Kinematic*

3 ĐỘNG HỌC NGHỊCH

3.1 Lập công thức giải bài toán động học nghịch

Ta sẽ tính toán các công thức động học ngược robot từ bài toán động lực thuận.

Link	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	d_1	θ_1^*
2	a_2	0	0	θ_2^*
3	a_3	0	0	θ_3^*

$$^0T_3 = {}^0T_1\,{}^1T_2\,{}^2T_3 = \begin{bmatrix} c_1c_{23} & -c_1s_{23} & s_1 & c_1(a_2c_2 + a_3c_{23}) \\ s_1c_{23} & -s_1s_{23} & -c_1 & s_1(a_2c_2 + a_3c_{23}) \\ s_{23} & c_{23} & 0 & d_1 + a_2s_2 + a_3s_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

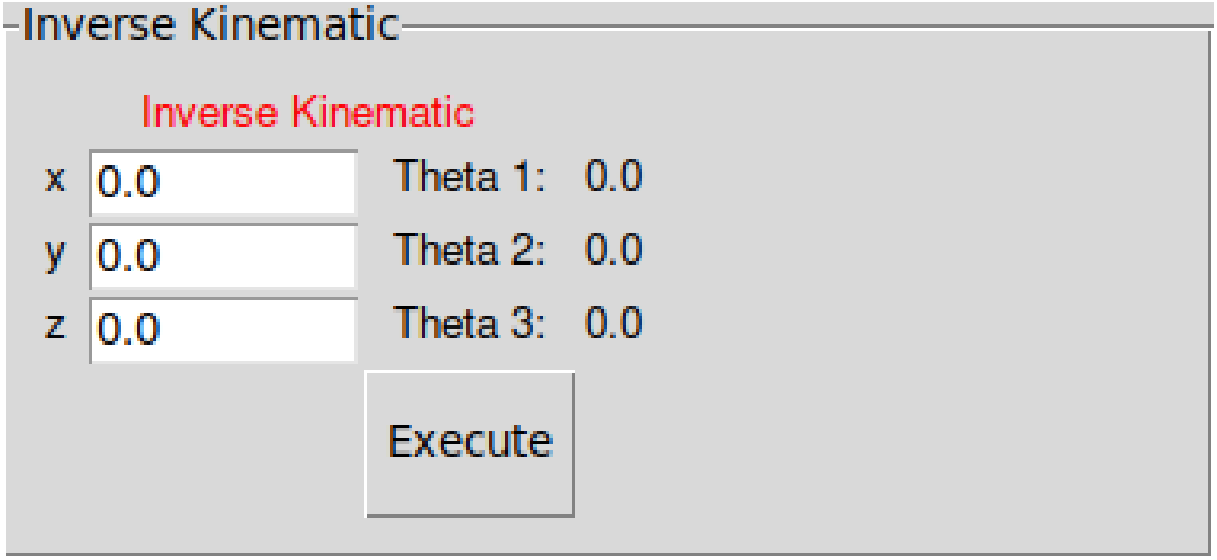
Với input Pw_x, Pw_y, Pw_z ta lập được công thức tính các góc θ_1 , θ_2 và θ_3:

- $\theta_3 = atan2(s_3, c_3)$
  - $c_3 = \frac{{Pw_x}^2 + {Pw_y}^2 + {Pw_z}^2 - {a_1}^2 - {a_2}^2}{2a_1a_2}$
  - $s_3 = \sqrt{1 - {c_3}^2}$
- $\theta_2 = \alpha - \beta$ với:
  - $\alpha = atan2(Pw_z, \sqrt{{Pw_x}^2 + {Pw_y}^2})$
  - $\beta = cos^{-1}(\frac{{Pw_x}^2 + {Pw_y}^2 + {Pw_z}^2 + {a_1}^2 - {a_2}^2}{2a_1\sqrt{{Pw_x}^2 + {Pw_y}^2 + {Pw_z}^2}})$
- $\theta_1 = atan2(Pw_y, Pw_x)$

3.2 Xây dựng UI và Animation cho bài toán động lực nghịch

3.2.1 Giao điện người dùng UI

Xây dựng giao diện để người dùng nhập tọa độ vị trí của End effector, sau khi bấm nút *Execute* robot sẽ tiến hành di chuyển đến vị trí trên.



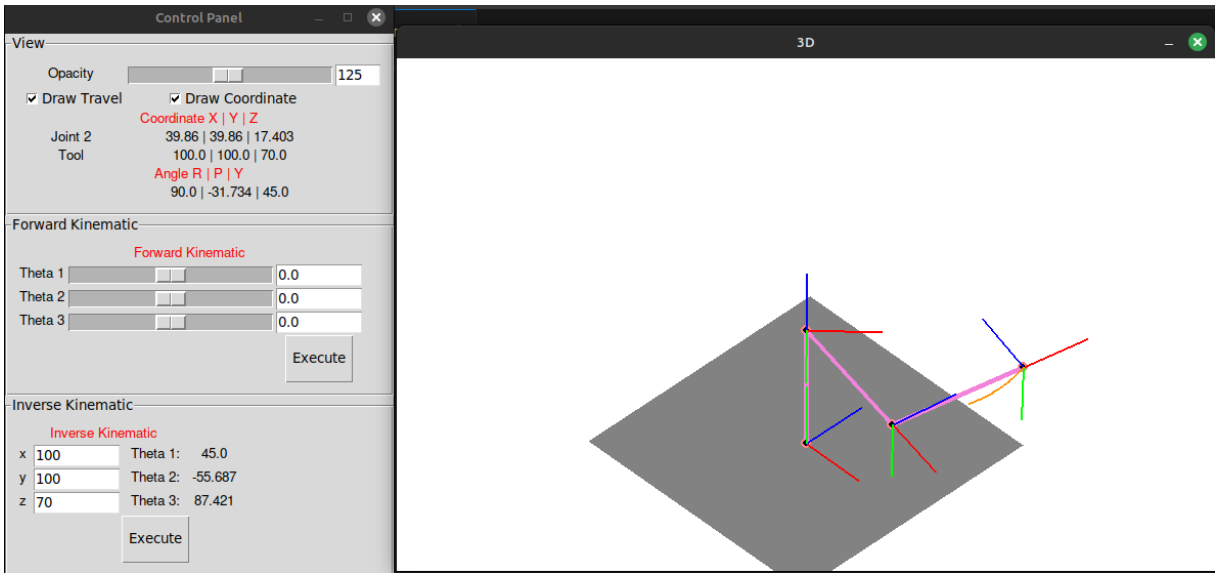
Hình 8: *Giao diện Inverse Kinematic*

3.2.2 Tạo animation và di chuyển robot theo bài toán động lực nghịch

Dựa vào các công thức đã lập được ở trên và các giá trị x, y và z do người dùng nhập ta tính ngược lại giá trị các góc θ_1 , θ_2 và θ_3, sau khi kiểm tra các góc xoay có nằm trong phạm vi giới hạn quya của từng khớp ta tiến hành cho robot chuyển động giống như bài toán thuận theo các giá trị θ vừa tính được ở trên.

```
def inverseKine():
global guiThetaInv
px = guiCoorInv[0].get()
py = guiCoorInv[1].get()
pz = guiCoorInv[2].get()-d[0]
# Check valid
valid = 0
tmp = (px**2+py**2+pz**2)**0.5
if (tmp <= a[1]+a[2]) and (tmp >= abs(a[1]-a[2])) and (pz+d[0] >= 0):
# calculate
c3 = (px**2+py**2+pz**2-a[1]**2-a[2]**2)/(2*a[1]*a[2])
s3 = (1-c3**2)**0.5
theta3 = atan2(s3, c3)

theta2 = atan2(pz, (px**2+py**2)**0.5) - acos((px**2+py**2+pz**2+a[1]**2-a[2]**2)
    /(2*a[1]*(px**2+py**2+pz**2)**0.5))

theta1 = atan2(py, px)

valid = 1
for i, thetaN in enumerate([theta1, theta2, theta3]):
```

```
21    if thetaN < limitTheta[i][0] or thetaN > limitTheta[i][1]:
22    valid = 0
23    else: guiThetaInv[i].set(round(thetaN*180/pi, 3))
24
25    if valid:
26    deltaTime = 60
27    nextTheta = np.array([theta1, theta2, theta3])
28    stepAngle = np.array([(nextTheta[0]-theta[0])/deltaTime, (nextTheta[1]-theta[1])/
          deltaTime, (nextTheta[2]-theta[2])/deltaTime])
29    animateTranform(nextTheta, stepAngle, deltaTime)
30    else:
31    tkinter.messagebox.showwarning("Inverse Kinematic.", "Out of workspace")
```

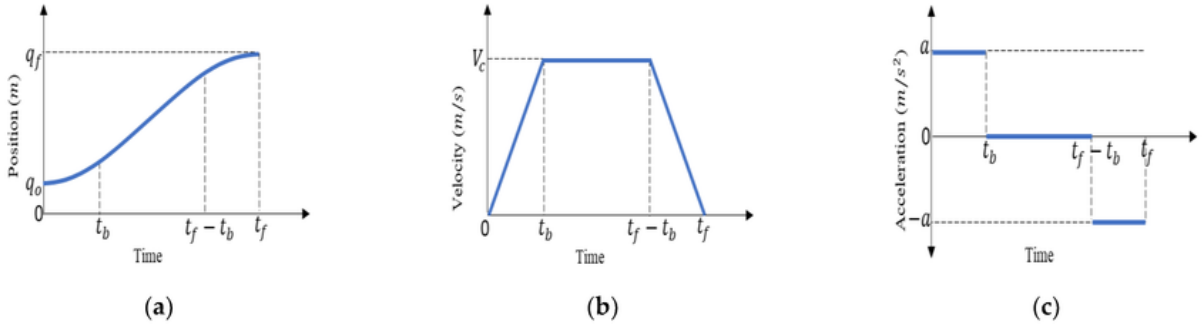


Hình 9: *Inverse Kinematic*

4 QUY HOẠCH QUỸ ĐẠO TRAJECTORY PLANNING

4.1 Linear Segment with Parabolic Blend (*LSPB*)

Xây dựng giải thuật LSPB

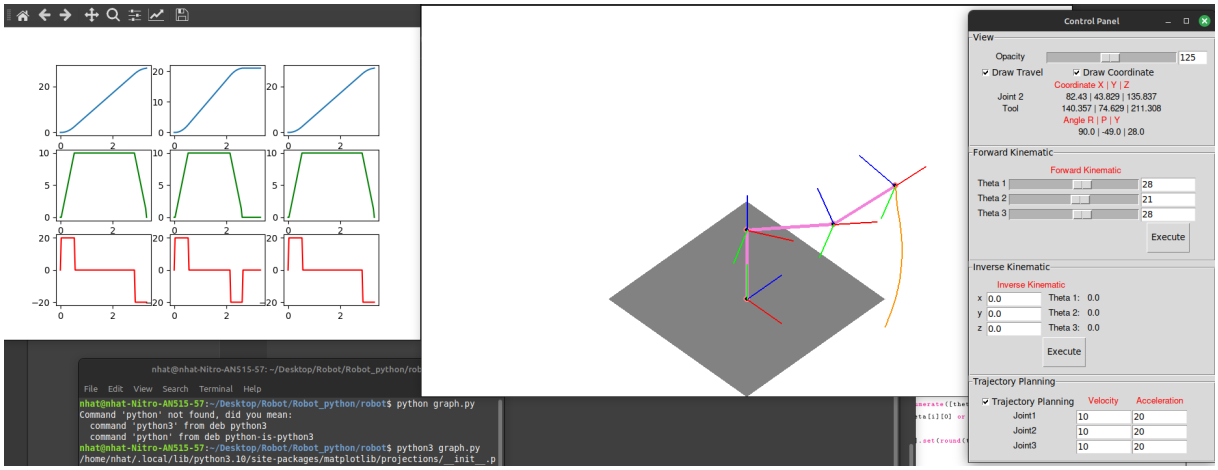


Hình 10: *Linear Segment with Parabolic Blend*

```
def lspb(q0, qn, vmax, amax):
  if q0 == qn:
  return np.zeros(1), np.array([q0]), np.zeros(1), np.zeros(1)
  q0 = q0*180/pi
  qn = qn*180/pi
  qmax = qn-q0
  if qmax < 0:
  amax = -amax
  vmax = -vmax
  t1 = round((vmax/amax)/0.0333)*0.0333

  if abs(qmax) < abs(amax*t1**2):
  t1 = round(np.sqrt(0.333*qmax*2/amax)/0.0333)*0.0333
  t2 = round(((qmax-amax*t1**2)/(amax*t1))/0.0333)*0.0333 + t1
  t3 = t2+t1

  else:
  t2 = round(((qmax-amax*t1**2)/vmax)/0.0333)*0.0333 + t1
  t3 = t2+t1

  t = np.arange(0, t3, 0.0333)

  qt = np.zeros(len(t), dtype=float)
  vt = np.zeros(len(t), dtype=float)
  at = np.zeros(len(t), dtype=float)

  for i, tt in enumerate(t):
  if tt <= t1:
  at[i] = amax
```

```
30      vt[i] = at[i]*tt
31      qt[i] = q0 + 0.5*at[i]*tt**2
32      elif tt <= t2:
33      at[i] = 0
34      vt[i] = amax*t1
35      qt[i] = q0 + 0.5*amax*t1**2 + vt[i]*(tt-t1)
36      else:
37      at[i] = -amax
38      vt[i] = amax*t1 + at[i]*(tt-t2)
39      qt[i] = q0 + 0.5*amax*t1**2 + amax*t1*(t2-t1) + amax*t1*(tt-t2) - 0.5*amax*(tt-
            t2)**2
40      qt[-1] = qn
41      vt[-1] = 0
42      return t, qt*pi/180, vt, at
```

4.2 Kết quả chạy được



Hình 11: *Linear Segment with Parabolic Blend*

5 ĐÁNH GIÁ CHUNG VÀ KẾT LUẬN

Sau thời gian thực hiện bài tập lớn môn Nhập môn Kỹ thuật Robot, em 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 bài toán động học thuận/nghịch, jacobi vào để giả lập chuyển động robot trong thực tế.
  - 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.
- 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, em 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 em 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.

6 VIDEO DEMO VÀ CODE PYTHON

Link video demo: Nền tảng Youtube

Tài liệu

[1] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. 2008. Robotics: Modelling, Planning and Control (1st. ed.). Springer Publishing Company, Incorporated.

[2] Mark Lutz. 2009. Learning Python: Powerful Object-Oriented Programming (4th. ed.). O'Reilly Media, Inc.