XING WENYU 3035906071

1.

DATE / / NO

1.

$$\begin{pmatrix} \cos\theta_1 , & -\sin\theta_1 \\ \sin\theta_1 , & \cos\theta_1 \end{pmatrix} \times \begin{pmatrix} l_x \\ l_y \end{pmatrix} = \begin{pmatrix} \cos\theta_1 l_x - \sin\theta_1 l_y \\ \sin\theta_1 l_x + \cos\theta_1 l_y \end{pmatrix}.$$

$$l_{global} = \begin{pmatrix} \cos\theta_1 l_x - \sin\theta_1 l_y + x_1 \\ \sin\theta_1 l_x + \cos\theta_1 l_y + y_1 \end{pmatrix}.$$

2.

$$\begin{pmatrix} \cos(-\theta_1) , & -\sin(-\theta_1) \\ \sin(-\theta_1) , & \cos(-\theta_1) \end{pmatrix} \times \begin{pmatrix} l_x - x_1 \\ l_y - y_1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta_1 (l_x - x_1) + \sin\theta_1 (l_y - y_1) \\ -\sin\theta_1 (l_x - x_1) + \cos\theta_1 (l_y - y_1) \end{pmatrix}.$$

3.

$$\begin{pmatrix} \cos(\theta_2 - \theta_1) , & -\sin(\theta_2 - \theta_1) , & x_2 - x_1 \\ \sin(\theta_2 - \theta_1) , & \cos(\theta_2 - \theta_1) , & y_2 - y_1 \\ 0 , & 0 , & 1 \end{pmatrix}$$

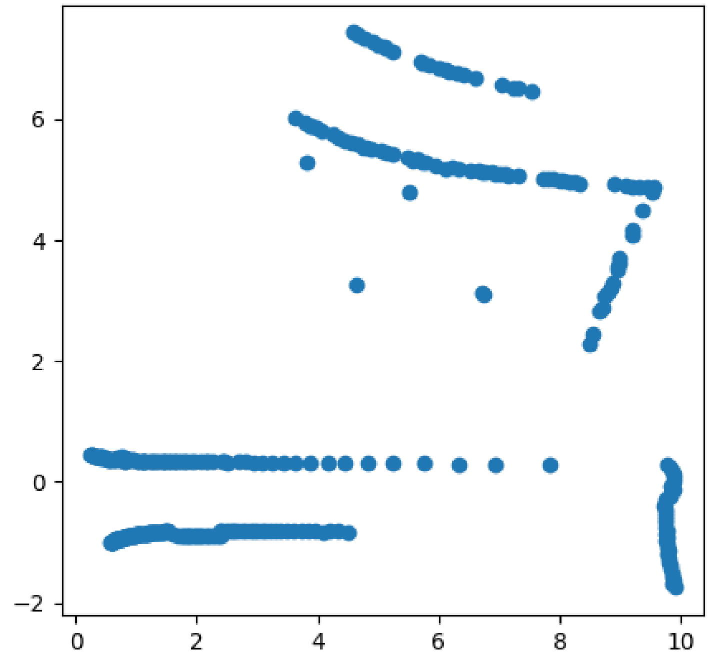
4.

$$\cancel{\begin{pmatrix} \cos\theta_2 , & -\sin\theta_2 \\ \sin\theta_2 , & \cos\theta_2 \end{pmatrix}} \times \cancel{\begin{pmatrix} \cos\theta_1 l_x - \sin\theta_1 l_y + x_1 \\ \sin\theta_1 l_x + \cos\theta_1 l_y + y_1 \end{pmatrix}}.$$

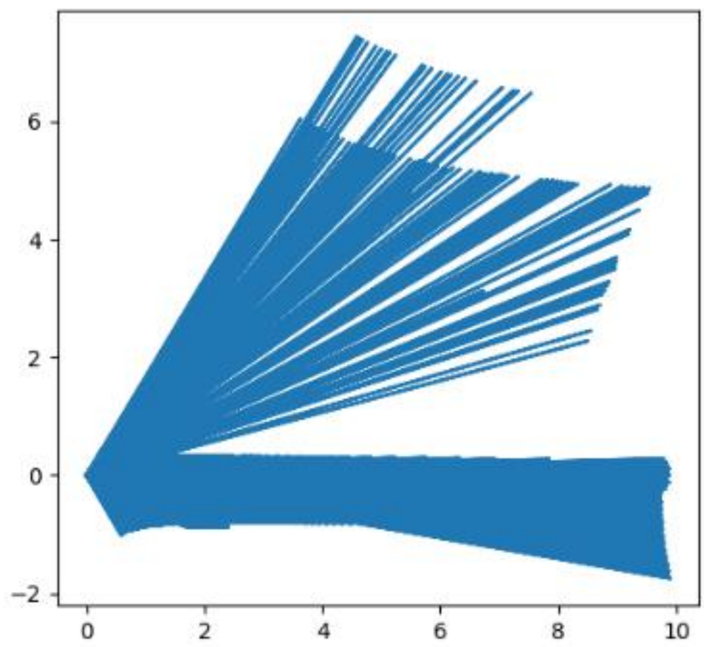
$$= \begin{pmatrix} \cos\theta_2 (\cos\theta_1 l_x - \sin\theta_1 l_y + x_1) - \sin\theta_2 (\sin\theta_1 l_x + \cos\theta_1 l_y + y_1) + x_2 \\ \sin\theta_2 (\cos\theta_1 l_x - \sin\theta_1 l_y + x_1) + \cos\theta_2 (\sin\theta_1 l_x + \cos\theta_1 l_y + y_1) + y_2 \end{pmatrix}.$$

2.

(a)

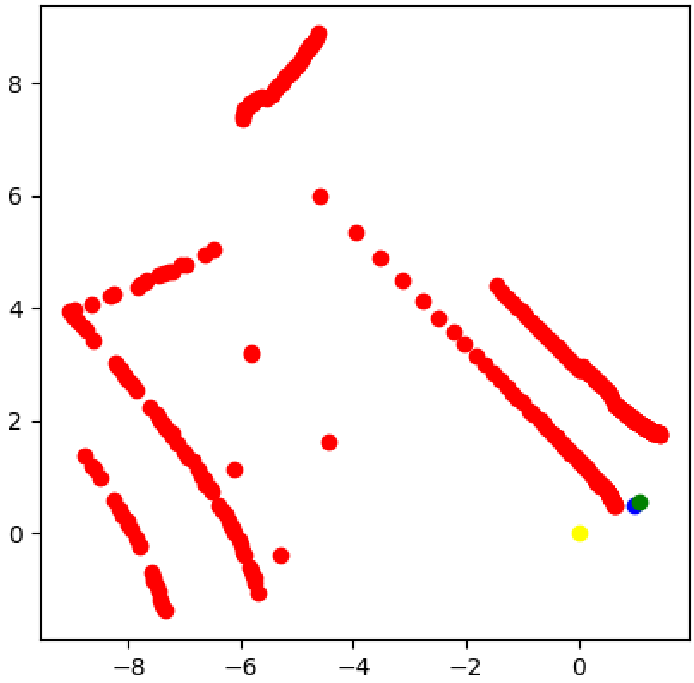


(b)



I draw the laser tracks of the data. As the figure shows above, in certain range of angle the laser got very close end-points, which means that maybe there an obstacle just in front of the robot.

(c)



center of the robot

the center of the lidar sensor

lidar end-points

world coordinate zero

3.

(a)

R = INF and the robot will go straight.

(b)

```
from math import sin, cos
import numpy as np

def diffdrive(x, y, theta, w1, w2, t, l):
    r = 1
    w = ((w1 - w2) * r) / (2 * l)
    if w1 == w2:
        return x + t * cos(theta) * w1 * r, y + t * sin(theta) * w1 * r, theta
    R = l * (w1 + w2) / (w2 - w1)
    icrx = x + R * sin(theta)
    icry = y - R * cos(theta)
    A = np.array(
        [[cos(w * t), -sin(w * t), 0],
         [sin(w * t), cos(w * t), 0],
         [0, 0, 1]]
    )
    B = np.array([[x - icrx], [y - icry], [theta]])
    C = np.array([[icrx], [icry], [w * t]])
    D = np.dot(A, B) + C
    xn = D[0][0]
    yn = D[1][0]
    thetan = D[2][0]

    return xn, yn, thetan
```

(c)

(1.9102065815032088, 2.738897253261369)

(d)

1.

We can't use 1 command to get the goal because if the robot ends in the same theta, it must go straight or do a 360 rotation. Neigh can get the goal.

2.

For two commands, consider:

R = l * (w1 + w2) / (w2 - w1)=0.125

w = ((w1 - w2) * r) / (2 * l) = pi

Then we can get (w1,w2,t).

This command will make robot to move to the position x=1.25m, y=2.0m, theta=-pi/2.

The robot can get the goal by doing this command twice.

(e)

0.25*pi

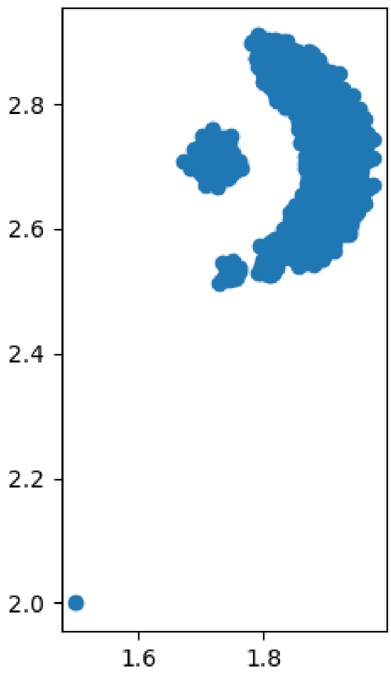
(f)

(i) additive Gaussian noise

```
def diffdrive_gauss(x, y, theta, w1, w2, t, l, n1, n2, n3):
    dot_number = 20
    r = 1
    w = ((w1 - w2) * r) / (2 * l)
    if w1 == w2:
        return x + t * cos(theta) * w1 * r, y + t * sin(theta) * w1 * r, theta
    R = l * (w1 + w2) / (w2 - w1)
    icrx = x + R * sin(theta)
    icry = y - R * cos(theta)
    A = np.array(
        [[cos(w * t), -sin(w * t), 0],
          [sin(w * t), cos(w * t), 0],
          [0, 0, 1]]
    )
    B = np.array([[x - icrx], [y - icry], [theta]])
    C = np.array([[icrx], [icry], [w * t]])
    D = np.dot(A, B) + C
    xn = D[0][0]
    yn = D[1][0]
    thetan = D[2][0]
    xn_list = []
    yn_list = []
    thetan_list = []
    for i in range(dot_number):
        xn_list.append(xn + np.random.normal(0, n1))
        yn_list.append(yn + np.random.normal(0, n2))
        thetan_list.append(thetan + np.random.normal(0, n3))
    return xn_list, yn_list, thetan_list
```

noise levels:n1 = n2 = 0.01 n3 = 0.3

result:

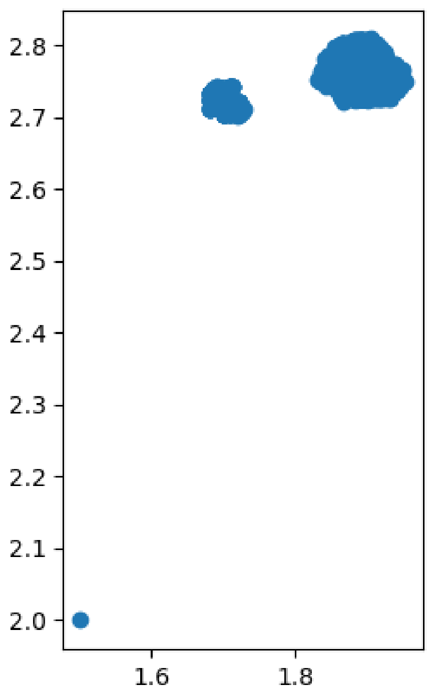


(ii) directly add the noise to the current state elements

```
def diffdrive_direct(x, y, theta, w1, w2, t, l, n):
    dot_number=30
    xn_list = []
    yn_list = []
    thetan_list = []
    for i in range(dot_number):
        w1+=np.random.normal(0,n)
        w2+=np.random.normal(0,n)
        r = 1
        w = ((w1 - w2) * r) / (2 * l)
        #w += np.random.normal(0, n)
        if w1 == w2:
            return x + t * cos(theta) * w1 * r, y + t * sin(theta) * w1 * r, theta
        R = l * (w1 + w2) / (w2 - w1)
        icrx = x + R * sin(theta)
        icry = y - R * cos(theta)
        A = np.array(
            [[cos(w * t), -sin(w * t), 0],
             [sin(w * t), cos(w * t), 0],
             [0, 0, 1]]
        )
        B = np.array([[x - icrx], [y - icry], [theta]])
        C = np.array([[icrx], [icry], [w * t]])
```

```
        D = np.dot(A, B) + C
        xn = D[0][0]
        yn = D[1][0]
        thetan = D[2][0]
        xn_list.append(xn)
        yn_list.append(yn)
        thetan_list.append(thetan)
    return xn_list, yn_list, thetan_list
```

noise levels: n=0.002
result:



The result of directly adding the noise to the current state elements is much similar to the Gaussian distribution than the first one. Because in the real two wheeled robot problem, the sensor reading of the wheels' angular velocity (w1, w2) is the main source of error. If we directly add the gaussian noise to the angular velocity, we will get the result that close to real. As for another method which we add gaussian noise to the states, this is not quite reasonable, because the 3 input parameters are chosen manually. In fact these noise level parameters should have some certain pattern that determined by the motion model. That's why we get a banana-shaped distribution.

4.

4. (a) $x_{k+1} = x_k + u_k + w_k \quad k=0,1,2,3 \quad w_k = 0$

cost : $\sum_{k=0}^{3} (x_k^2 + u_k^2) \quad x_0 = 5$

$V_t(x) = \min_{u_t} [c(x, u_t) + V_{t+1}(f(x, u_t))]$

① $k=3$

$V_3(x_3) = \min (x_3^2 + u_3^2 + V_4(x_3+u_3))$

$= \min (x_3^2 + u_3^2)$

$= x_3^2 \quad (u_3 = 0)$

② $k=2$

$V_2(x_2) = \min (x_2^2 + u_2^2 + (x_2+u_2)^2)$

$= \min (2x_2^2 + 2x_2u_2 + 2u_2^2)$

$x_2=0 \quad V_2(0) = \min (2u_2^2) = 0 \quad (u_2 = 0)$

$x_2=1 \quad V_2(1) = \min (2 + 2u_2 + 2u_2^2) = 2 \quad (u_2=0, u_2=-1)$

$x_2=2 \quad V_2(2) = \min (8 + 4u_2 + 2u_2^2) = 6 \quad (u_2 = -1)$

$x_2=3 \quad V_2(3) = \min (18 + 6u_2 + 2u_2^2) = 14 \quad (u_2=-1, u_2=-2)$

$x_2=4 \quad V_2(4) = \min (32 + 8u_2 + 2u_2^2) = 24 \quad (u_2 = -2)$

$x_2=5 \quad V_2(5) = \min (50 + 10u_2 + 2u_2^2) = 38 \quad (u_2=-3, u_2=-2).$

③ $k=1$ $V_1(x_1) = \min(x_1^2 + u_1^2 + V_2(x_1+u_1))$.

$x_1=0$ $V_1(0) = \min(u_1^2 + V_2(u_1)) = 0$ $(u_1=0)$.

$x_1=1$ $V_1(1) = \min(1 + u_1^2 + V_2(1+u_1)) = 2$ $(u_1=-1)$.

$x_1=2$ $V_1(2) = \min(4 + u_1^2 + V_2(2+u_1)) = 7$ $(u_1=-1)$

$x_1=3$ $V_1(3) = \min(9 + u_1^2 + V_2(3+u_1)) = 15$ $(u_1=-2)$

$x_1=4$ $V_1(4) = \min(16 + u_1^2 + V_2(4+u_1)) = 26$ $(u_1=-2)$

$x_1=5$ $V_1(5) = \min(25 + u_1^2 + V_2(5+u_1)) = 40$ $(u_1=-3)$.

④ $k=0$. $x_0=5$

$V_0(x_0) = \min(x_0^2 + u_0^2 + V_1(x_0+u_0))$

$V_0(5) = \min(25 + u_0^2 + V_1(u_0+5)) = 41$ $(u_0=-3)$.

optimal x_k, u_k sequence:

$x_0 = 5$ $u_0 = -3$ cost $= 34$.

$x_1 = 2$ $u_1 = -1$ cost $= 5$

$x_2 = 1$ $u_2 = 0, -1$ cost $= 1, 2$

$x_3 = 0, 1$ $u_3 = 0$ cost $= 0, 1$

(b)

① $k=3$

$V_3(x_3) = \min_{\mathbb{W}} (x_3^2 + u_3^2 + V_4(x_3+u_3+w_3))$

$= \min(x_3^2 + u_3^2) = x_3^2$ $(u_3=0)$

② $k=2$

$V_2(x_2) = \min(x_2^2 + u_2^2 + V_3(x_2+u_2+w_2))$.

if $\begin{pmatrix} u_2+x_2=5 \\ \text{or } u_2+x_2=0 \end{pmatrix}$ $V_2(x_2) = \min(x_2^2 + u_2^2 + V_3(x_2+u_2))$.

$\cancel{x}$ $V_2(x_2) = \min(2x_2^2 + 2u_2^2 + 2x_2u_2)$

$x_2 = 0$ $V_2(0) = \min(2u_2^2) = 0$ $u_2 = 0$

$x_2 = 1$ $V_2(1) = \min(2 + 2u_2^2 + 2u_2) = 2$ $u_2 = -1$

$x_2 = 2$ $V_2(2) = \min(8 + 2u_2^2 + 4u_2) = 8$ $u_2 = -2$

$x_2 = 3$ $V_2(3) = \min(18 + 2u_2^2 + 6u_2) = 18$ $u_2 = -3$

$x_2 = 4$ $V_2(4) = \min(32 + 2u_2^2 + 8u_2) = 32$ $u_2 = -4$

$x_2 = 5$ $V_2(5) = \min(50 + 2u_2^2 + 10u_2) = 50$ $u_2 = -5$, $u_2 = 0$.

else:

$V_2(x_2) = \min(x_2^2 + u_2^2 + \frac{1}{2}V_3(x_2 + u_2 + 1) + \frac{1}{2}V_3(x_2 + u_2 - 1))$

$= \min(x_2^2 + u_2^2 + \frac{1}{2}x_2^2 + \frac{1}{2}u_2^2 + \frac{1}{2} + x_2u_2 + \cancel{\frac{1}{2}x_2} + \frac{1}{2}u_2$

$+ \frac{1}{2}x_2^2 + \frac{1}{2}u_2^2 + \frac{1}{2} + x_2u_2 - \frac{1}{2}u_2 - \cancel{\frac{1}{2}x_2})$

$= \min(2x_2^2 + 2u_2^2 + 2x_2u_2 + 1)$

$x_2 = 0$ $V_2(0) = 0$ $u_2 = 0$.

$x_2 = 1$ $V_2(1) = 2$ $u_2 = -1$

$x_2 = 2$ $V_2(2) = 7$ $u_2 = -1$

$x_2 = 3$ $V_2(3) = 15$ $u_2 = -2$, $u_2 = -1$

$x_2 = 4$ $V_2(4) = 25$ $u_2 = -2$

$x_2 = 5$ $V_2(5) = 39$ $u_2 = -2$, $u_2 = -3$.

③ $k = 1$

if $u_1 + x_1 = 5$ or $u_1 + x_1 = 0$.

$V_1(x_1) = \min(x_1^2 + u_1^2 + V_2(x_1 + u_1))$.

$\cancel{if}$ else

$V_1(x_1) = \min(x_1^2 + u_1^2 + \frac{1}{2}V_2(x_1 + u_1 + 1) + \frac{1}{2}V_2(x_1 + u_1 - 1))$

$\cancel{x_2 = 0}$

$x_1=0$ $V_1=0$ $u_1=0$

$x_1=1$ $V_1=2$ $u_1=-1$

$x_1=2$ $V_1=8$ $u_1=-2$

$x_1=3$ $V_1=16.5$ $u_1=-2$

$x_1=4$ $V_1=28.5$ $u_1=-3$, $u_1=-2$.

$x_1=5$ $V_1=42.5$ $u_1=-3$

$k=0$. $x_0=5$.

$\cancel{V_0(x_0)}$ if $u_0=-5$ or $u_0=0$.

$V_0(5) = \min(25+u_0^2 + V_1(u_1+5))$

else

$V_0(5) = \min(25+u_0^2+V_1(u_1+6)+V_1(u_1+4))$.

$x_0=5$ $V_0=42.25$ $u_0=-3$.

We can't compute the optimal action sequence and state sequence because for each Uk, we can't get the exact state Xk+1, which means the state sequence in this MDP is a tree structure. We can only tell the high probability optimal action in each state.

python code in solving question(b):

```
def v2(x, u):
    if u + x == 5 or u + x == 0:
        return 2 * x ** 2 + 2 * u ** 2 + 2 * x * u
    else:
        return 2 * x ** 2 + 2 * u ** 2 + 2 * x * u + 1

for x in range(0, 6):
    v_list = []
    for u in range(-5, 6):
        if ((x + u) < 0) or ((x + u) > 5): continue
        print(x, u, v2(x, u))
        v_list.append(v2(x, u))
    print('v2---min:---', min(v_list))

def v2_res(x):
    if x==0: return 0
    if x == 1: return 2
    if x == 2: return 7
    if x == 3: return 15
    if x == 4: return 25
    if x == 5: return 39
```

```
def v1(x, u):
        if u + x == 5 or u + x == 0:
                return x ** 2 + u ** 2 + v2_res(x+u)
        else:
                return x** 2 + u ** 2 + 0.5*v2_res(x+u+1)+0.5*v2_res(x+u-1)

for x in range(0, 6):
        v_list = []
        for u in range(-5, 6):
                if ((x + u) < 0) or ((x + u) > 5): continue
                print(x, u, v1(x, u))
                v_list.append(v1(x, u))
        print('v1---min:---', min(v_list))

def v1_res(x):
        if x==0: return 0
        if x == 1: return 2
        if x == 2: return 8
        if x == 3: return 16.5
        if x == 4: return 28.5
        if x == 5: return 42.5

def v0(x, u):

        if u + x == 5 or u + x == 0:
                return x ** 2 + u ** 2 + v1_res(x+u)
        else:
                return x** 2 + u ** 2 + 0.5*v1_res(x+u+1)+0.5*v1_res(x+u-1)

v_list = []
x=5
for u in range(-5, 6):
        if ((x + u) < 0) or ((x + u) > 5): continue
        print(x, u, v0(x, u))
        v_list.append(v0(x, u))
print('v0---min:---', min(v_list))
```

5.

$\dot{x} = Ax + Bu \qquad y = Cx.$

consider $\tilde{x} = \begin{bmatrix} x \\ 1 \end{bmatrix} \qquad \tilde{u} = \ddot{\tilde{x}} = \begin{bmatrix} \ddot{x} \\ 0 \end{bmatrix} = \begin{bmatrix} u \\ 0 \end{bmatrix}$

then $\dot{\tilde{x}} = \begin{bmatrix} A & \Delta \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} + \begin{bmatrix} B & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ 0 \end{bmatrix}$

$y = \begin{bmatrix} C & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} X \\ 1 \end{bmatrix}$

6.

(a)

Present an optimization method to solve coupled redundant inverse kinematics problems and generate trajectories for humanoid robot full-body manipulation.

(b)

inverse kinematics

gradient-based optimization

Sequential Quadratic Programming

Constrained Bidirectional RRT

2-link planar system

(c)

In this implement, it generates randomly one task, which contain 3 key pose and 1 goal (the star mark). The arm structure (i.e. the length) is randomly set and it's start pose is the red one. For each step, I try to use a simple inverse kinematics function to figure out the robot arm joint coordinate, and it should be optimal.