A dual Ackerman drive would steer both front and rear wheels using an Ackerman steering approach. What would the pros and cons for this system compared to a single Ackerman drive?

Pros:

-can still be used easily and intuitively with a conventional steering wheel

-gives steering over all 4 wheels

-allows for tighter turns about some pivot point not centered on the vehicle

-drive simillar to most newer lawn tractors

Cons:

-increased complexity means higher liklihood of failure

Real motion and measurement involves error and this problem will introduce the concepts. Assume that you have a differential drive robot with wheels that are 20cm in radius and L is 12cm. Using the differential drive code (forward kinematics) from the text, develop code to simulate the robot motion when the wheel velocities are *ϕ*˙1=0.25*t*2, *ϕ*˙2=0.5*t*. The starting location is [0,0] with *θ*=0

1. Plot the path of the robot on 0≤*t*≤5. It should end up somewhere near [50,60].
2. Assume that you have Gaussian noise added to the omegas each time you evaluate the velocity (each time step). Test with *μ*=0 and *σ*=0.3. Write the final location (x,y) to a file and repeat for 100 simulations. Hint:

mu, sigma = 0.0, 0.3

xerr = np.random.normal(mu,sigma, NumP)

yerr = np.random.normal(mu,sigma, NumP)

1. Generate a plot that includes the noise free robot path and the final locations for the simulations with noise. Hint:

**import** **numpy** **as** **np**

**import** **pylab** **as** **plt**

...

plt.plot(xpath,ypath, 'b-', x,y, 'r.')

plt.xlim(-10, 90)

plt.ylim(-20, 80)

plt.show()

1. Find the location means and 2x2 covariance matrix for this data set, and compute the eigenvalues and eigenvectors of the matrix. Find the ellipse that these generate. [The major and minor axes directions are given by the eigenvectors. Show the point cloud of final locations and the ellipse in a graphic (plot the data and the ellipse). Hint:

**from** **scipy** **import** linalg

**from** **matplotlib.patches** **import** Ellipse

*# assume final locations are in x & y*

mat = np.array([x,y])

*# find covariance matrix*

cmat = np.cov(mat)

*# compute eigenvals and eigenvects of covariance*

eval, evec = linalg.eigh(cmat)

*# find ellipse rotation angle*

angle = 180\*atan2(evec[0,1],evec[0,0])/np.pi

*# create ellipse*

ell = Ellipse((np.mean(x),np.mean(y)),

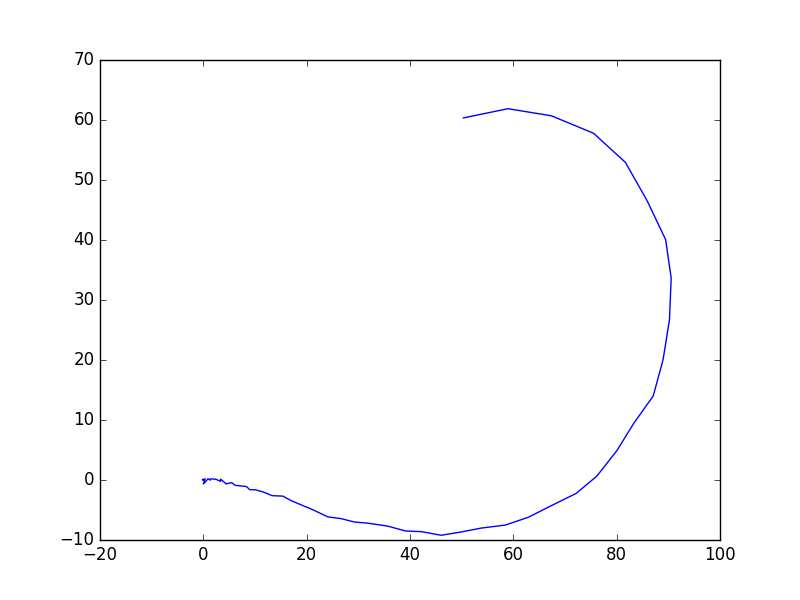
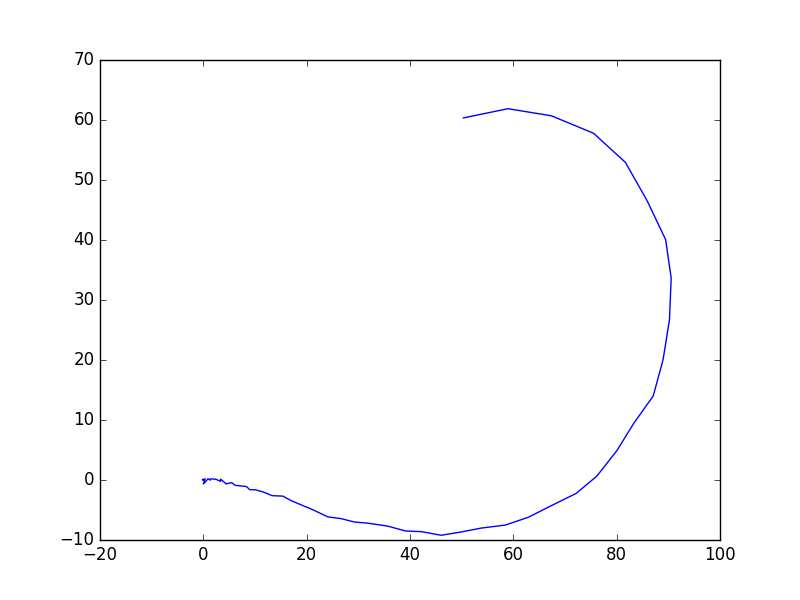
eval[0],eval[1],angle)

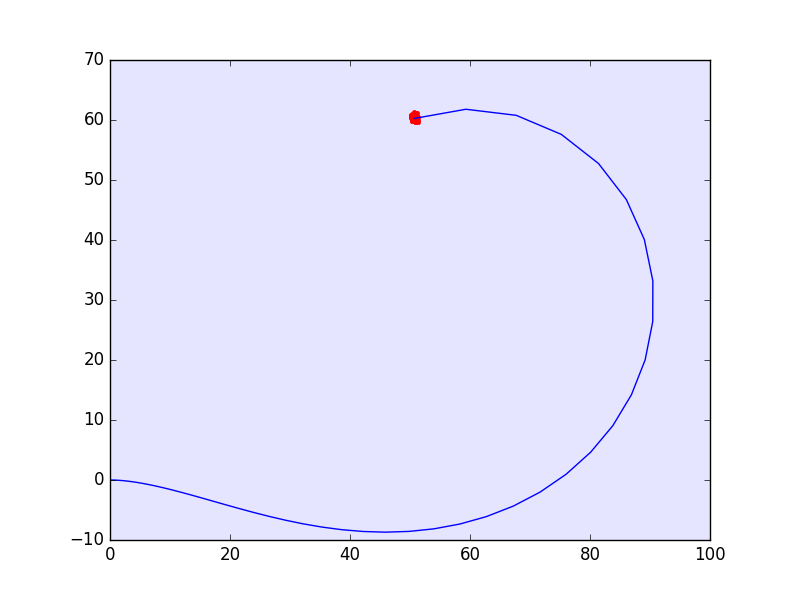
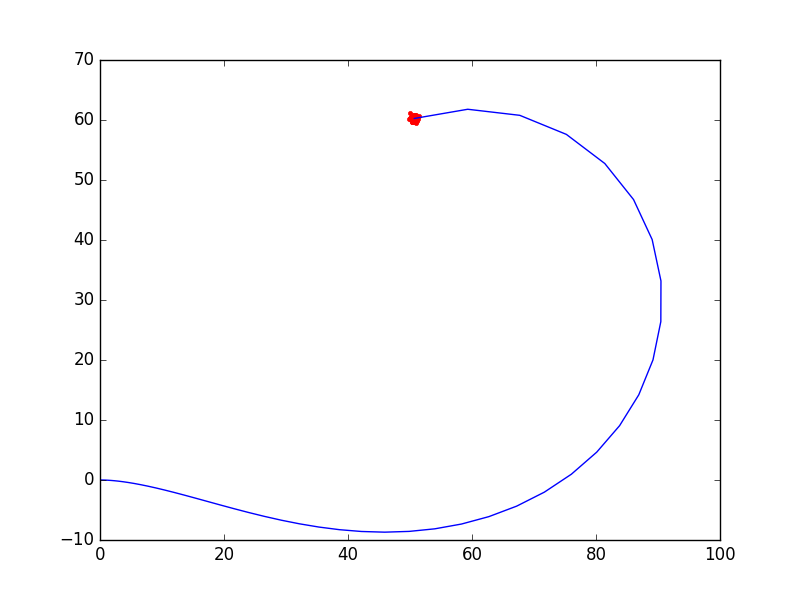
*# make the ellipse subplot*

a = plt.subplot(111, aspect='equal')

ell.set\_alpha(0.1) *# make the ellipse lighter*

a.add\_artist(ell) *# add this to the plot*





Top Left: 6.5.a

Top Right: 6.5.b

Bottom Left: 6.5.c

Bottom Right: 6.5.d

Code Base for problem 6.5:

a.)

import pylab as plt

import numpy as np

from math import \*

N=52

x = np.zeros(N)

y = np.zeros(N)

q = np.zeros(N)

x[0] = 0; y[0] = 0; q[0] = 0.0

t = 0; dt = 0.1

def ddstep(xc, yc, qc,r,l,dt,w1,w2):

xn = xc + (r\*dt/2.0)\*(w1+w2)\*cos(qc)

yn = yc + (r\*dt/2.0)\*(w1+w2)\*sin(qc)

qn = qc + (r\*dt/(2.0\*l))\*(w1-w2)

return (xn,yn,qn)

for i in range(N-1):

w1 = 0.25\*t\*t

w2 = 0.5\*t

x[i+1], y[i+1], q[i+1] = ddstep(x[i], y[i], q[i],20,12.0,dt,w1,w2)

t = t + dt

print(t)

plt.plot(x,y,'b')

plt.show()

b.)

import pylab as plt

import numpy as np

from math import \*

N=52

mu, sigma = 0.0, 0.3

filename = "output.txt"

file = open(filename, "w")

for k in range(100):

x = np.zeros(N)

y = np.zeros(N)

x\_with\_error = np.zeros(N)

y\_with\_error = np.zeros(N)

q = np.zeros(N)

xerr = np.random.normal(mu,sigma, 100)

yerr = np.random.normal(mu,sigma, 100)

x[0] = 0; y[0] = 0; q[0] = 0.0

t = 0; dt = 0.1

def ddstep(xc, yc, qc,r,l,dt,w1,w2):

xn = xc + (r\*dt/2.0)\*(w1+w2)\*cos(qc)

yn = yc + (r\*dt/2.0)\*(w1+w2)\*sin(qc)

qn = qc + (r\*dt/(2.0\*l))\*(w1-w2)

return (xn,yn,qn)

for i in range(N-1):

w1 = 0.25\*t\*t

w2 = 0.5\*t

x\_with\_error[i] = xerr[i] + x[i]

y\_with\_error[i] = yerr[i] + y[i]

x[i+1], y[i+1], q[i+1] = ddstep(x[i], y[i], q[i],20,12.0,dt,w1,w2)

x\_with\_error[i+1], y\_with\_error[i+1], q[i+1] = ddstep(x\_with\_error[i], y\_with\_error[i], q[i],20,12.0,dt,w1,w2)

t = t + dt

print(t)

file.write('{0} {1}\n'.format(x[51], y[51]))

k+=1

file.close()

plt.plot(x\_with\_error,y\_with\_error,'b')

plt.show()

c.)

import pylab as plt

import numpy as np

from math import \*

N=52

mu, sigma = 0.0, 0.3

filename = "output.txt"

file = open(filename, "w")

for k in range(100):

x = np.zeros(N)

y = np.zeros(N)

x\_with\_error = np.zeros(N)

y\_with\_error = np.zeros(N)

q = np.zeros(N)

xerr = np.random.normal(mu,sigma, 100)

yerr = np.random.normal(mu,sigma, 100)

x[0] = 0; y[0] = 0; q[0] = 0.0

t = 0; dt = 0.1

def ddstep(xc, yc, qc,r,l,dt,w1,w2):

xn = xc + (r\*dt/2.0)\*(w1+w2)\*cos(qc)

yn = yc + (r\*dt/2.0)\*(w1+w2)\*sin(qc)

qn = qc + (r\*dt/(2.0\*l))\*(w1-w2)

return (xn,yn,qn)

for i in range(N-1):

w1 = 0.25\*t\*t

w2 = 0.5\*t

x\_with\_error[i] = xerr[i] + x[i]

y\_with\_error[i] = yerr[i] + y[i]

x[i+1], y[i+1], q[i+1] = ddstep(x[i], y[i], q[i],20,12.0,dt,w1,w2)

x\_with\_error[i+1], y\_with\_error[i+1], q[i+1] = ddstep(x\_with\_error[i], y\_with\_error[i], q[i],20,12.0,dt,w1,w2)

t = t + dt

plt.plot(x\_with\_error[51],y\_with\_error[51],'r.')

k+=1

plt.plot(x,y,'b-')

plt.show()

d.)

import pylab as plt

import numpy as np

from scipy import linalg

from matplotlib.patches import Ellipse

from math import \*

N=52

mu, sigma = 0.0, 0.3

filename = "output.txt"

file = open(filename, "w")

x\_final = np.zeros(100)

y\_final = np.zeros(100)

for k in range(100):

x = np.zeros(N)

y = np.zeros(N)

x\_with\_error = np.zeros(N)

y\_with\_error = np.zeros(N)

q = np.zeros(N)

xerr = np.random.normal(mu,sigma, 100)

yerr = np.random.normal(mu,sigma, 100)

x\_mean = 0

y\_mean = 0

xerr\_mean =0

yerr\_mean =0

x[0] = 0; y[0] = 0; q[0] = 0.0

t = 0; dt = 0.1

def ddstep(xc, yc, qc,r,l,dt,w1,w2):

xn = xc + (r\*dt/2.0)\*(w1+w2)\*cos(qc)

yn = yc + (r\*dt/2.0)\*(w1+w2)\*sin(qc)

qn = qc + (r\*dt/(2.0\*l))\*(w1-w2)

return (xn,yn,qn)

for i in range(N-1):

w1 = 0.25\*t\*t

w2 = 0.5\*t

x\_with\_error[i] = xerr[i] + x[i]

y\_with\_error[i] = yerr[i] + y[i]

x\_mean += x[i]

y\_mean += y[i]

xerr\_mean += x\_with\_error[i]

yerr\_mean += y\_with\_error[i]

x\_with\_error[i+1], y\_with\_error[i+1], q[i+1] = ddstep(x\_with\_error[i], y\_with\_error[i], q[i],20,12.0,dt,w1,w2)

x[i+1], y[i+1], q[i+1] = ddstep(x[i], y[i], q[i],20,12.0,dt,w1,w2)

t = t + dt

x\_final[k] = x\_with\_error[51]

y\_final[k] = y\_with\_error[51]

x\_mean /= N

y\_mean /= N

xerr\_mean /= N

yerr\_mean /= N

print(x\_mean)

print(y\_mean)

tempx = (x\_mean+xerr\_mean)/2

tempy = (y\_mean+yerr\_mean)/2

varx= (x\_mean - tempx)\*(x\_mean - tempx) + (xerr\_mean - tempx)\*(xerr\_mean - tempx)

vary= (y\_mean - tempy)\*(y\_mean - tempy) + (yerr\_mean - tempy)\*(yerr\_mean - tempy)

covxy= (x\_mean - tempx)\*(y\_mean - tempy) + (xerr\_mean - tempx)\*(yerr\_mean - tempy)

# assume final locations are in x & y

mat = np.array([x,y])

# find covariance matrix

cmat = np.cov(mat)

# compute eigenvals and eigenvects of covariance

eval, evec = linalg.eigh(cmat)

# find ellipse rotation angle

angle = 180\*atan2(evec[0,1],evec[0,0])/np.pi

# create ellipse

ell = Ellipse((np.mean(x),np.mean(y)),

eval[0],eval[1],angle)

# make the ellipse subplot

a = plt.subplot(111, aspect='equal')

ell.set\_alpha(0.1) # make the ellipse lighter

a.add\_artist(ell) # add this to the plot

print(eval)

for j in range(100):

plt.plot(x\_final[j], y\_final[j],'r.')

plt.plot(x,y,'b-')

plt.show()