zs2489 Zhou Shen
yx2725 YaoYao Xu

CAofR HW3

COMS 4733
November 10, 2021

Problem 1: 1-Dimensional Kalman Filter

(a) Write out the complete set of Kalman filter equations for the estimates of the state mean $\hat{x}$ and variance p.

$$\begin{aligned}
\hat{x}_{k|k-1} &= f_k \hat{x}_{k-1|k-1} + g_k u_k \\
p_{k|k-1} &= f_k p_{k-1|k-1} f_k^T + q_k \\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(z_k - H_k \hat{x}_{k|k-1}) \\
&= \hat{x}_{k|k-1} + p_{k|k-1} h_k^T (h_k p_{k|k-1} h_k^T + r_k)^{-1}(z_k - H_k \hat{x}_{k|k-1}) \\
p_{k|k} &= (I - p_{k|k-1} h_k^T (h_k p_{k|k-1} h_k^T + r_k)^{-1} h_k) p_{k|k-1}
\end{aligned} \tag{1}$$

(b) Examine the Kalman gain and the update equations that you derived.

We know that $var(h_k x_k) = h_k p_{k|k-1} h_k^T$ and $var(z_k) = r_k$. So the limiting behavior of the mean and variance updates in following two situations are (Refer to $equation_{[1]}$ above):

1)$var(h_k x_k) \ll var(z_k)$:
K_k will tend to be 0, which means the mean and the variance will be tend to be the predicted mean($h_k \hat{x}_k$) and predicted variance($h_k p_{x|x-1} h_k^T$). In other words, the model will trust the prediction more instead of measurement.

2)$var(h_k x_k) \gg var(z_k)$:
K_k will tend to be 1, which means the mean and the variance will be tend to be the measurement value(z_k) and 0. In other words, the model will trust the measurement more instead of prediction.

(c) Derive a quadratic equation for the state variance $p_{k|k}$ in the limit as $k \to \infty$, assuming that it converges to a fixed value.

If f,g,h,q,r are all constant over time, and we can assume that $p_{k|k-1}$ is 1 dimensional variance, so we get:

$$\begin{aligned}
p_{k|k-1} &= f^2 p_{k-1|k-1} + q \\
p_{k|k} &= p_{k|k-1} - \frac{p_{k|k-1} h^2}{h^2 p_{k|k-1} + r} \\
&= p_{k|k-1}(\frac{r}{p_{k|k-1} h^2 + r})
\end{aligned} \tag{2}$$

Then, put $P_{k|k-1}$ in $P_{k|k}$, we get:

$$p_{k|k} = (f^2 p_{k-1|k-1} + q)(\frac{r}{(f^2 p_{k-1|k-1} + q)h^2 + r}) \tag{3}$$

When $k \to \infty$, we can assume $p_{k|k-1} = p_{k|k}$, so the quadratic equation is:

$$f^2 h^2 p_{\infty|\infty}^2 + (qh^2 - f_2 r + r) p_{\infty|\infty} + qr = 0 \tag{4}$$

Problem 2: All Kinds of Noise

The properties we use: When computing $x_{k|k-1}$, we add the Expectation of each part and add them together. While, when we compute $P_{k|k-1}$, we use the identity $cov(Ax) = A\sum A^T$ to get the covariance and add them together.

(a) Find the new prediction equations for $\hat{x}_{k|K-1}$ and $P_{k|k-1}$, given $\hat{x}_{k-1|k-1}$ and $P_{k-1|k-1}$

$$\begin{aligned} \hat{x}_{k|k-1} &= F_k\hat{x}_{k-1|k-1} + G_k u_k + \bar{w}_k \\ P_{k|k-1} &= F_k P_{k-1|k-1} F_k^T + Q_k \end{aligned} \tag{5}$$

(b) Find the new prediction equations for $\hat{x}_{k|K-1}$ and $P_{k|k-1}$, given $\hat{x}_{k-1|k-1}$ and $P_{k-1|k-1}$

$$\begin{aligned} \hat{x}_{k|k-1} &= F_k\hat{x}_{k-1|k-1} + G_k \bar{u}_k \\ P_{k|k-1} &= F_k P_{k-1|k-1} F_k^T + G_k U_k G_k^T + Q_k \end{aligned} \tag{6}$$

(c) Propose a change to the definition of the innovation ˜y to account for this. Briefly explain if the update equations will change or stay $\tilde{y}$ the same as a result.

Observed measurement is $(\mu_1, \Sigma_1) = (\bar{z}_k, R_k)$, now the $\bar{z}_k$ is $H_k x_k + \bar{v}_k$, so we get:

$$\tilde{y} = z_k - H_k\hat{x}_{k|k-1} = H_k x_k + \bar{v}_k - H_k \tag{7}$$

$\hat{x}_{k|k}$ will change because $\tilde{y}$ has changed, and there will always be a measurement noise here. Also, due to the noise, the measurement will be less accurate, which will cause the decrease of K_k, so that the model will trust the prediction more. While, $P_{k|K}$ won't change because only R_k is counted in the equation.

Problem 3: Kalman Filter Simulation

1. Question3 Code

```
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt

def KF(x, P, z):
# IMPLEMENT THIS
  xHat = F.dot(x)
  pHat =(F.dot(P)).dot(F.T) + Q
```

```
  S = (H.dot(pHat)).dot(H.T) + R
  K = (pHat.dot(H.T)).dot(inv(S))
  yhat = z - H.dot(xHat)

  x = xHat + K.dot(yhat)
  m = K.dot(H).shape[0]
  P = (np.eye(m) - K.dot(H)).dot(pHat)

  print("the Covariance Matrix is: ", P)
  print("the Kalman Gain is: ", K)
  print("the Innovation gain is: ", S)
  print("-------------------------------------")
  return x, P

F = np.array([[1, 0.5], [0, 1]])
H = np.array([[0, 1]])
#Q = np.array([[10, 1], [1, 50]])
Q=np.array([[0.01,0.001],[0.001,0.005]])
#Q = np.array([[0.1, 0.01], [0.01, 0.05]])
R = 0.1

x = np.array([[0.8, 2]])
xhat = np.array([[2, 4]])
P = np.array([[1,0], [0,2]])

for i in range(1000):
  w = np.random.multivariate_normal(np.zeros(2), Q)
  v = np.random.normal(0, R)
  x = np.vstack((x, F @ x[-1,:] + w))
  z = H @ x[-1,:] + v

  xnew, P = KF(xhat[-1,:], P, z)
  xhat = np.vstack((xhat, xnew))

plt.plot(x[:,0], x[:,1], label = "actual state")
plt.plot(xhat[:,0], xhat[:,1], label = "predicted state")
plt.legend()
plt.xlabel('position')
plt.ylabel('velocity')
plt.show()
```

(a) The code is attached to the appendix page. The two results are shown in figures below.

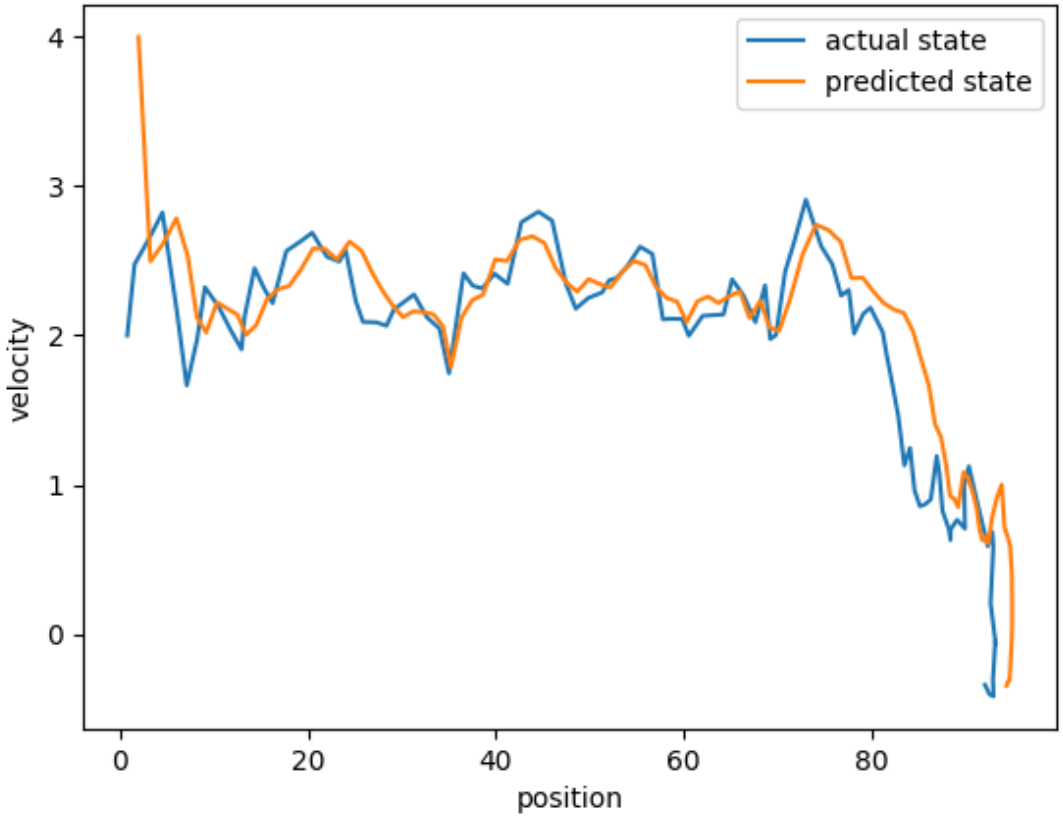

Figure 1: Problem 3a, Implement the KF procedure 1

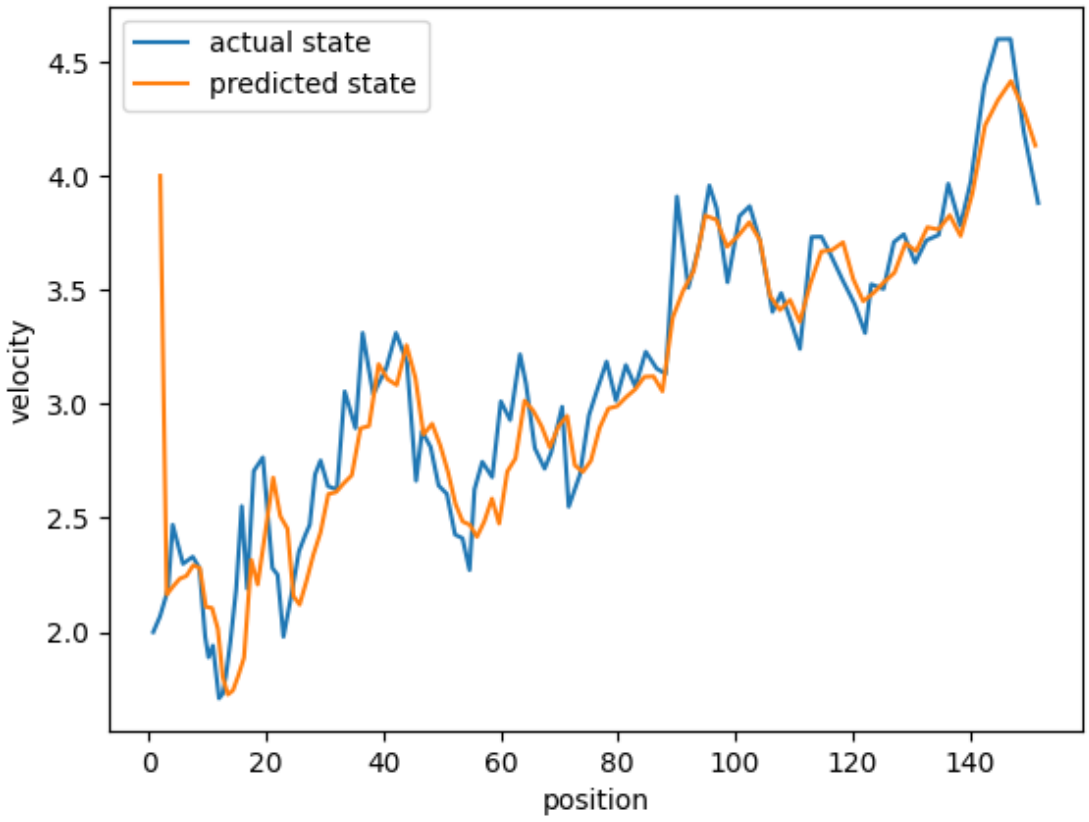

Figure 2: Problem 3a, Implement the KF procedure 2

(b) The state covariance, Kalman gain, and innovation covariance converge to a number after suitable iterations. This holds for all the simulations. Because when we do the multiplication of two Gaussians, it's variance will decrease, so the filter will have less uncertainty in suitable iterations, which means the state covariance, Kalman gain, and innovation covariance will converge to a number (as the figure below shows) and this number will not change much each time we run the script, since we did not change our input, such as Q and r.

```
-------------------------------------
the Covariance Matrix is:  [[13.18745238  0.035     ]
 [ 0.035       0.05      ]]
the Kalman Gain is:  [[0.35]
 [0.5 ]]
the Innovation gain is:  [[0.2]]
-------------------------------------
the Covariance Matrix is:  [[13.31045238  0.035     ]
 [ 0.035       0.05      ]]
the Kalman Gain is:  [[0.35]
 [0.5 ]]
the Innovation gain is:  [[0.2]]
```

Figure 3: Problem 3b, Converged Output

(c) When we use a larger Q, the total distance traveled will increase because of the Q causes large variance in the motion model. On the contrary, smaller Q will has less effects on the motion model. In an ideal cause, the noise Q should has effect on motion then inputs. Thus, a noise Q that less or equal than the effect of input is a reasonable choice.

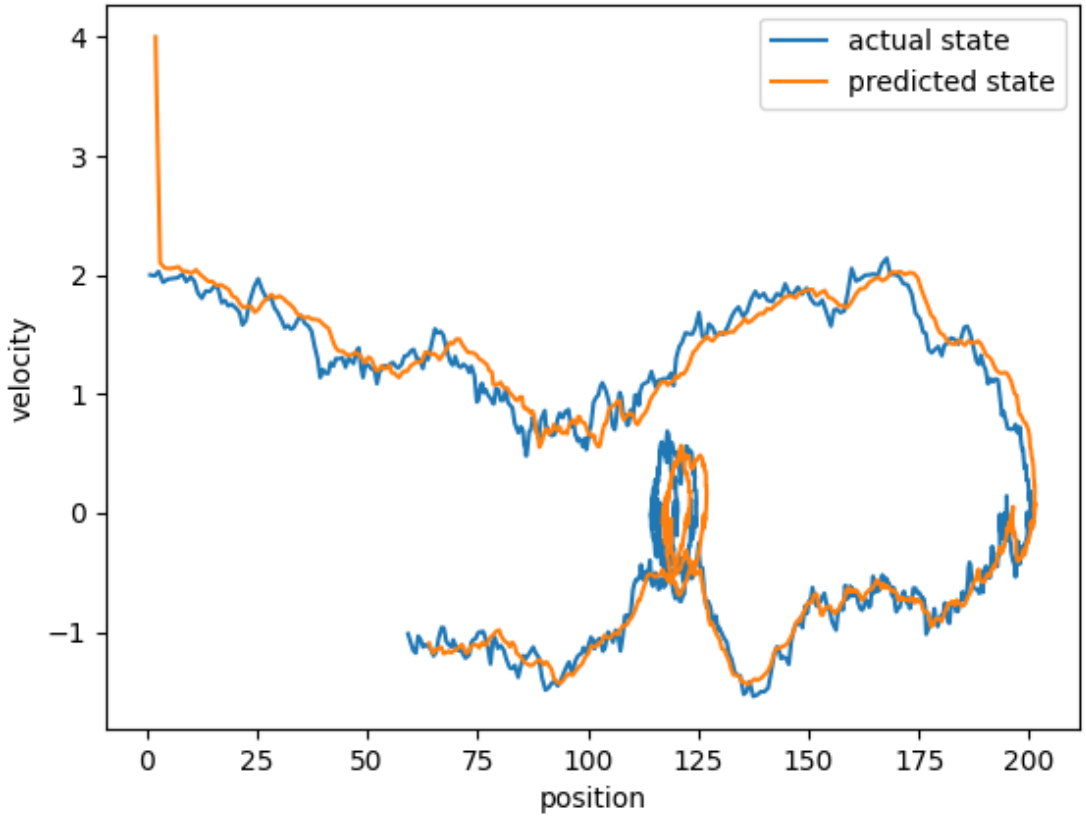

Figure 4: Problem 3c, Output using small Q

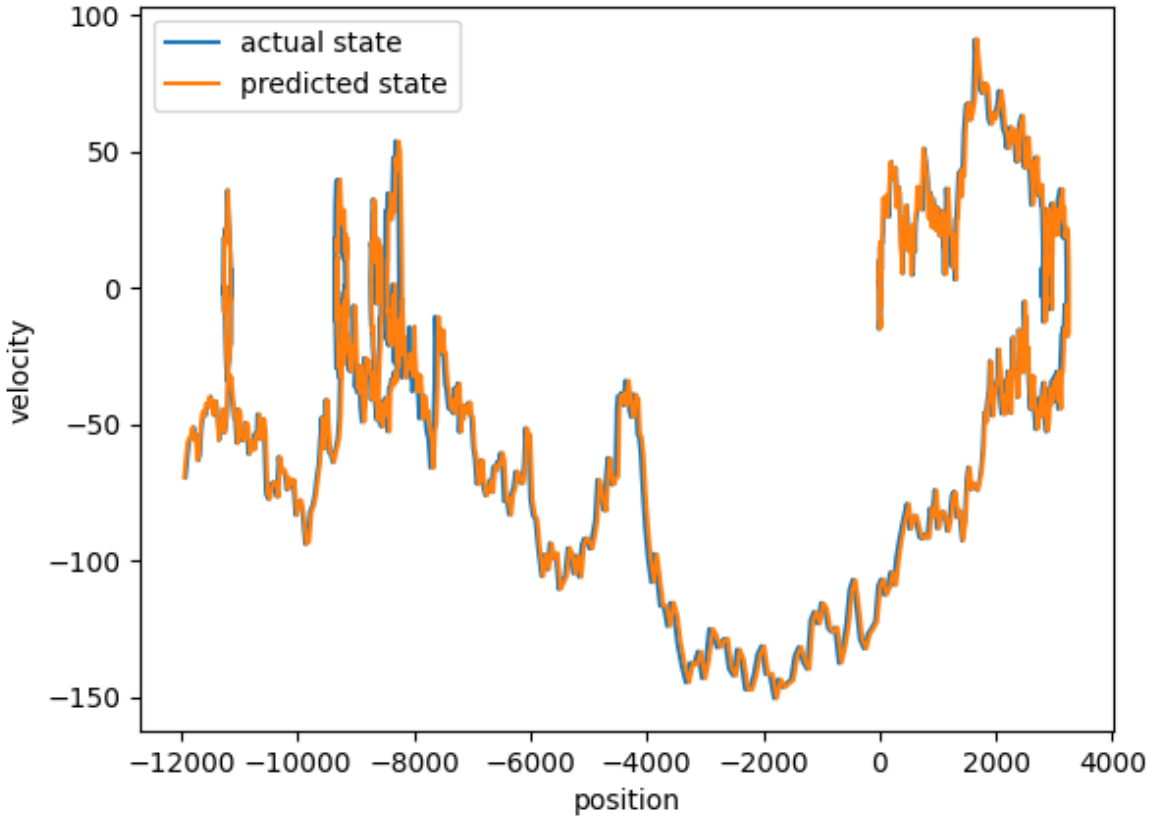

Figure 5: Problem 3c, Output using large Q

(d) We tried the r to be very large, which is 100, there is a big difference between our prediction and the actual robot's states. In contrast, when we tried a very small r, which is 0.001, the plot shows that the prediction is very close to the actual robots' states. R represents the noises from the measurement equation, the variance of our prediction will increase gradually, if the variance of our measurement is large as well, the product of two Gaussian distributions will increase more, which will lead to the big difference between our prediction and the actual robot's state.

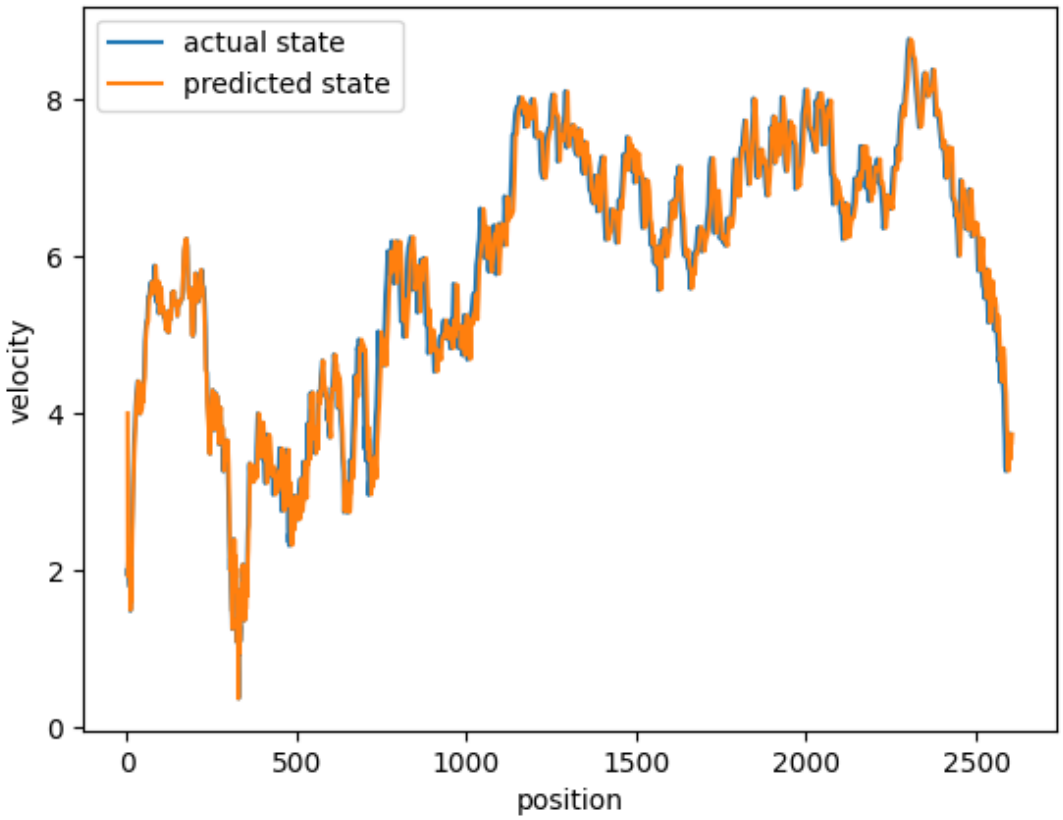

Figure 6: Problem 3d, Output using small r

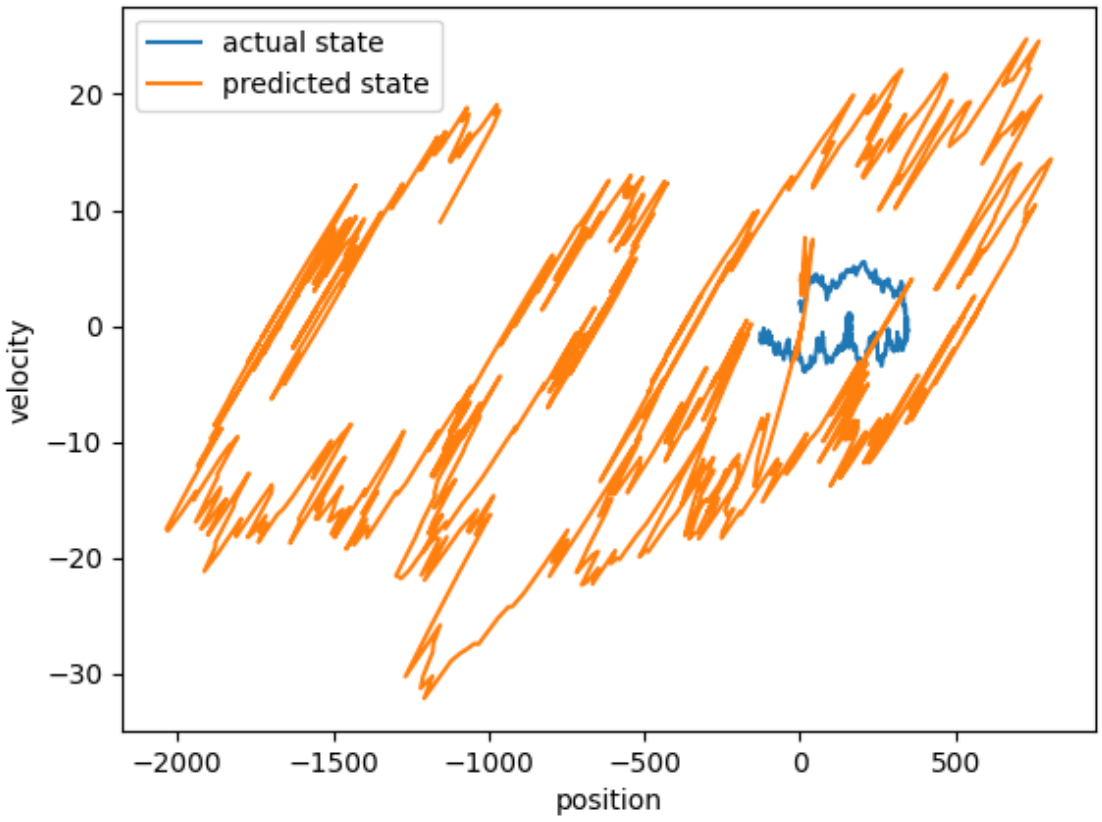

Figure 7: Problem 3d, Output using large r

Problem 4: Non-Euclidean RRT

In the code, we use a small trick to reach the goal configuration faster. For each qnew we get, we will compute the distance between qnew and GOAL. If their distance is pretty close, we connect qnew directly to GOAL, and use $clear_path$ function. If the path is clear, we reach the GOAL.

(a) The final image of RRT and workspace of qnew path in environment 1 is:

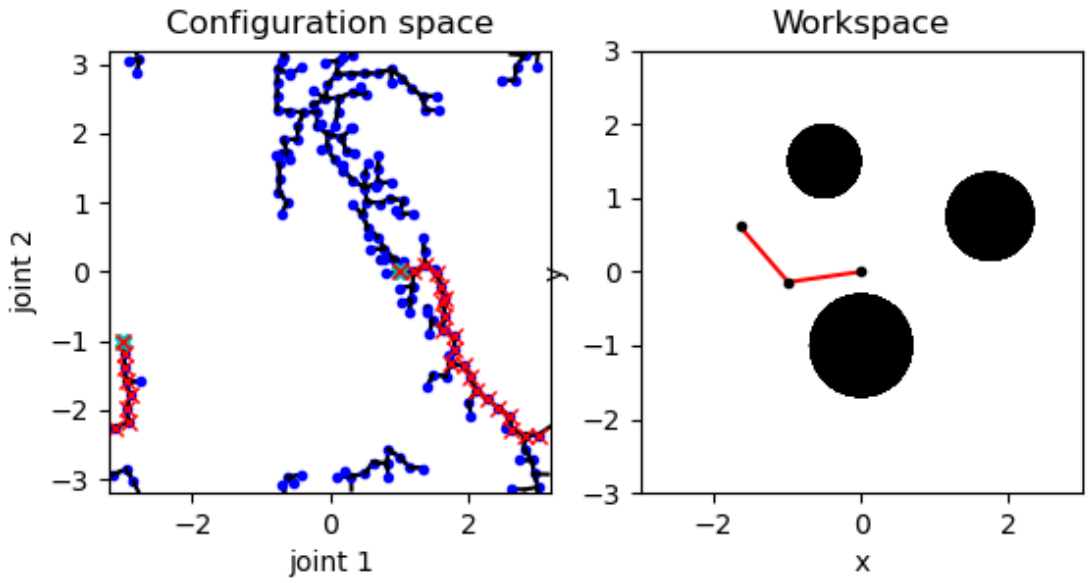

Figure 8: Problem 4, Env1-Qnew-Path

The video of qnew path in environment 1 is: https://youtu.be/7o3__3bvVXE

(b) The final image of RRT and workspace of qnew path in environment 2 is:

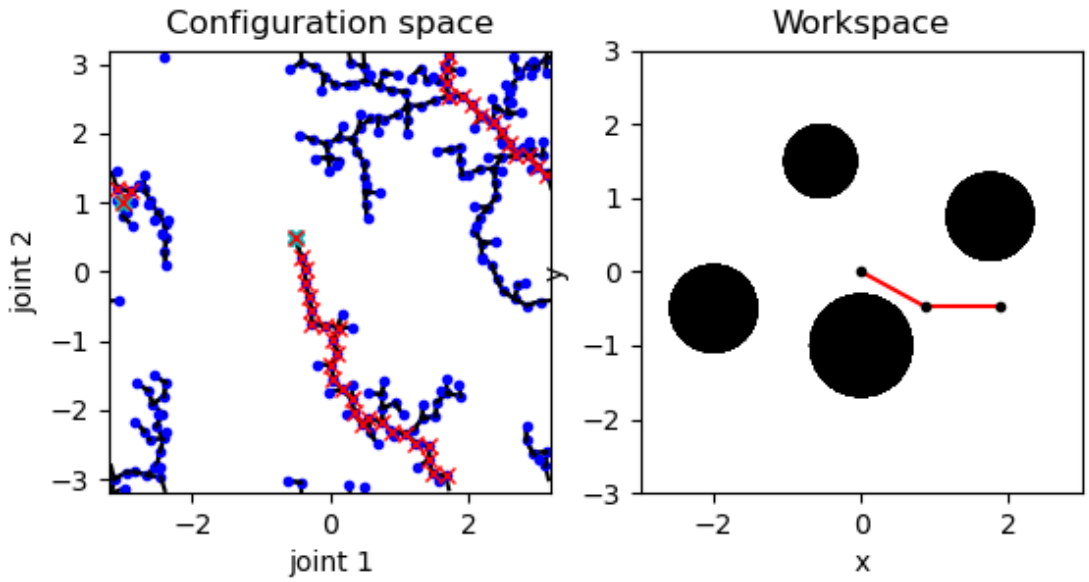

Figure 9: Problem 4, Env2-Qnew-Path

The video of qnew path in environment 2 is: https://youtu.be/KJ2bwjTy11s

(c) The final image of RRT and workspace of qnew-greedy path in environment 1 is:

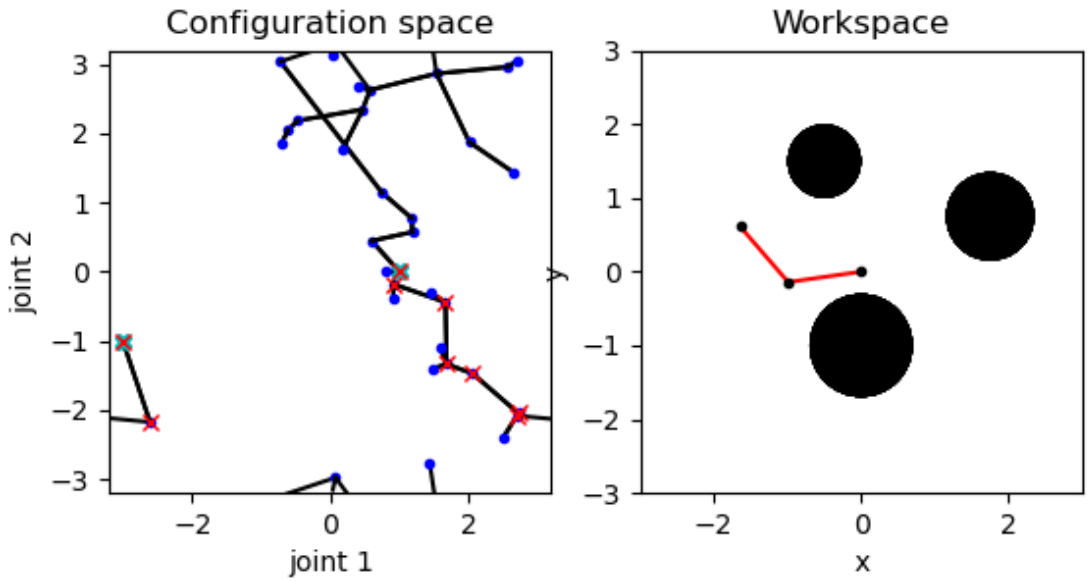

Figure 10: Problem 4, Env1-Qnew-Greedy-Path

The video of qnew greedy path in environment 1 is: https://youtu.be/B6FAW3zlqYQ

(d) The final image of RRT and workspace of qnew-greedy path in environment 2 is:

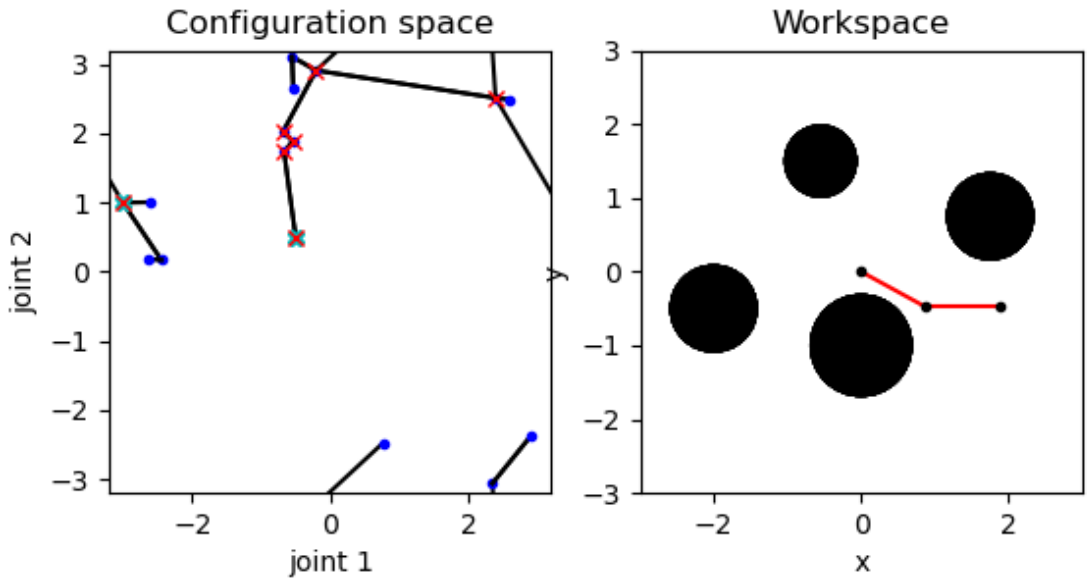

Figure 11: Problem 4, Env2-Qnew-Greedy-Path

The video of qnew greedy path in environment 2 is: https://youtu.be/dECE5FXSGyU

Appendix:

1. Question4 Code

```
def closest_euclidean(q, qp):
    """
    :param q, qp: Two 2D vectors in S1 x S1
    :return: qpp, dist. qpp is transformed version of qp so that L1 Euclidean
        distance between q and qpp
    is equal to toroidal distance between q and qp. dist is the corresponding
        distance.
    """
    q = np.array(q)
    qp = np.array(qp)

    A = np.meshgrid([-1,0,1], [-1,0,1])
    qpp_set = qp + 2*np.pi*np.array(A).T.reshape(-1,2)
    distances = np.linalg.norm(qpp_set-q, 1, axis=1)
    ind = np.argmin(distances)
    dist = np.min(distances)

    return qpp_set[ind], dist

def clear_path(arm, q1, q2):
    """
    :param arm: NLinkArm object
    :param q1, q2: Two configurations in S1 x S1
    :return: True if edge between q1 and q2 sampled at EDGE_INC increments
        collides with obstacles, False otherwise
    """
    q2p, dist = closest_euclidean(q1,q2) #from q1 to q1p(q2prime) is the shorter
        segment
    q_angle = np.arctan2(q2p[1]-q1[1], q2p[0]-q1[0])
    #print(f"q2p is {q2p}, dist is {dist}, q_angle is {q_angle}")

    #continue add EDGE_INC
    config = np.array(q1)
    while (EDGE_INC < dist) :
        config[0] += np.cos(q_angle) * EDGE_INC
        config[1] += np.sin(q_angle) * EDGE_INC
        if detect_collision(arm, config) == True:
            return True
        else:
            dist -= EDGE_INC
    if detect_collision(arm, q2p) == True:
```

```
        return True
    else:
        return False

def find_qnew(arm, tree, qrand):
    """
    :param tree: RRT dictionary {(node_x, node_y): (parent_x, parent_y)}
    :param qrand: Randomly sampled configuration
    :return: qnear in tree, qnew between qnear and qrand with distance DELTA from
        qnear
    """
    #dists includes (dist, key)
    dists = [(closest_euclidean(qrand, key)[1], key) for key in tree.keys()]
    qnear = sorted(dists, key = lambda x: x[0])[0][1]
    qrandp, dist = closest_euclidean(qnear, qrand)
    qnew = [0,0]
    if dist < DELTA:
        qnew = qrandp
    else:
        q_angle = np.arctan2(qrandp[1]-qnear[1], qrandp[0]-qnear[0])
        qnew[0] = qnear[0] + np.cos(q_angle) * DELTA
        qnew[1] = qnear[1] + np.sin(q_angle) * DELTA

    if clear_path(arm, qnear, qnew):
        qnew = None
    else:
        if qnew[0] > np.pi:
            qnew[0] -= 2*np.pi
        elif qnew[0] < -np.pi:
            qnew[0] += 2*np.pi
        if qnew[1] > np.pi:
            qnew[1] -= 2*np.pi
        elif qnew[1] < -np.pi:
            qnew[1] += 2*np.pi
        qnew = tuple(qnew)

    return qnear, qnew

def find_qnew_greedy(arm, tree, qrand):
    """
    :param arm: NLinkArm object
    :param tree: RRT dictionary {(node_x, node_y): (parent_x, parent_y)}
    :param qrand: Randomly sampled configuration
    :return: qnear in tree, qnew between qnear and qrand as close as possible to
```

```
    qrand in increments of DELTA
"""
#dists includes (dist, key)
qnear, qnew = find_qnew(arm, tree, qrand)
final_qnew = [0,0] #final_qnew is used to store the latest qnew which will be
    returned
if qnew == None:
    final_qnew = None
    return qnear, final_qnew
else:
    qrandp, dist = closest_euclidean(qnew, qrand)
    q_angle = np.arctan2(qrandp[1]-qnew[1], qrandp[0]-qnew[0])
    while True:
        new_qnew = [0,0] #new_qnew is the next qnew after the current qnew,
            clear_path will be used afterwards
        if dist < DELTA:
            new_qnew = qrandp
            if clear_path(arm, qnew, new_qnew):
                final_qnew = qnew
                break
            else:
                final_qnew = qrandp
                break
        else:
            new_qnew[0] = qnew[0] + np.cos(q_angle) * DELTA
            new_qnew[1] = qnew[1] + np.sin(q_angle) * DELTA
        if clear_path(arm, qnew, new_qnew):
            final_qnew = qnew
            break
        else:
            qnew = new_qnew
            dist -= DELTA
if final_qnew[0] > np.pi:
    final_qnew[0] -= 2*np.pi
elif final_qnew[0] < -np.pi:
    final_qnew[0] += 2*np.pi
if final_qnew[1] > np.pi:
    final_qnew[1] -= 2*np.pi
elif final_qnew[1] < -np.pi:
    final_qnew[1] += 2*np.pi
final_qnew = tuple(final_qnew)


return qnear, final_qnew
```

```
def construct_tree(arm):
    """
    :param arm: NLinkArm object
    :return: roadmap: Dictionary of nodes in the constructed tree {(node_x,
       node_y): (parent_x, parent_y)}
    :return: path: List of configurations traversed from start to goal
    """
    reach_goal = False
    tree = {START:None}
    for i in range(1,MAX_NODES):
        np.random.seed()
        if np.random.rand() > BIAS:
            qrand = (np.random.uniform(-np.pi, np.pi), np.random.uniform(-np.pi,
               np.pi))
        else:
            qrand = GOAL
        #change the function between(find_qnew, find_qnew_greedy) to see the
           different strategies
        #qnear, qnew = find_qnew_greedy(arm, tree, qrand)
        qnear, qnew = find_qnew_greedy(arm, tree, qrand)
        if qnew == None:
            continue
        _,dist2goal = closest_euclidean(qnew, GOAL)
        if dist2goal <= DELTA:
            #set qnew = GOAL
            if clear_path(arm, qnear, GOAL):
                continue
            else:
                tree[GOAL] = qnear
                reach_goal = True
                break
        else:
            tree[qnew] = qnear

    if reach_goal == True:
        route = [GOAL]
        tree_temp = tree.copy()
        while True:
            parent = tree_temp.pop(route[-1])
            route.append(parent)
            if parent == START:
                break
        route = route[::-1]
```

```
        return tree, route
    else:
        return tree,[]
```