Table of Contents

ECE6555 HW4

Author: Teo Wilkening Due Date: 2022/11/10

Q-4 Trajectory + Measurements Plot

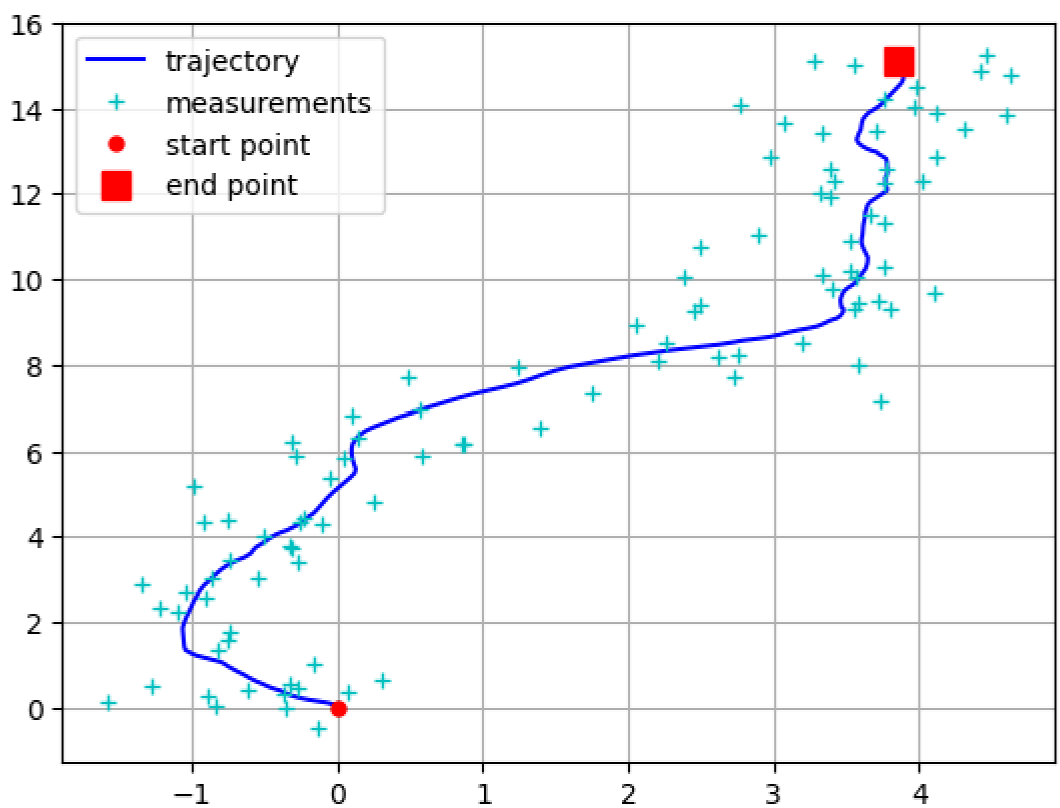
In [1]:

```
import numpy as np
from scipy import signal

np.random.seed(1992)
NumSteps = 201
TimeScale = np.linspace(0,10,NumSteps)
DeltaSim = np.diff(TimeScale)[0]
SigmaInput = 1
SigmaNoise = 0.5

F = np.array([[1,0,DeltaSim,0],[0,1,0,DeltaSim],[0,0,1,0],[0,0,0,1]])
Q = SigmaInput**2 * np.array([[DeltaSim**3/3,0,DeltaSim**2/2,0],
                              [0,DeltaSim**3/3,0,DeltaSim**2/2],
                              [DeltaSim**2/2,0,DeltaSim,0],
                              [0,DeltaSim**2/2,0,DeltaSim]])
H = np.array([[1,0,0,0],[0,1,0,0]])
R = SigmaNoise**2 * np.identity(2)

State = np.zeros((4,NumSteps))
NoisyMeasurements = np.zeros((2,NumSteps))

for t in np.arange(1,NumSteps):
    ProcessNoise = np.squeeze(np.matmul(np.linalg.cholesky(Q),np.random.randn(4,1)))
    State[:,t] = np.matmul(F,State[:,t-1]) + ProcessNoise
    MeasurementNoise = SigmaNoise * np.squeeze(np.random.randn(2))
    NoisyMeasurements[:,t] = np.matmul(H,State[:,t]) + MeasurementNoise
```

```
StateX1 = State[0,:]
StateX2 = State[1,:]

DownSampling=2
NoisyMeasurements = NoisyMeasurements[:,::DownSampling]
MeasurementY1 = NoisyMeasurements[0,:]
MeasurementY2 = NoisyMeasurements[1,:]
```

In [2]:

```
import matplotlib.pyplot as plt

# plt.figure(figsize=(3,3), dpi = 180)
plt.figure(1)
plt.plot(StateX1,StateX2,'b')
plt.plot(MeasurementY1,MeasurementY2,'c+')
plt.grid(True)
plt.plot(StateX1[0],StateX2[0],'r.',markersize=10)
plt.plot(StateX1[-1],StateX2[-1],'rs',markersize=10)
plt.legend(['trajectory','measurements','start point','end point'])

plt.rcParams['figure.figsize'] = [6, 6]
plt.rcParams['figure.dpi'] = 120 # 200 e.g. is really fine, but slower
plt.show()
```



Q-5 Kalman Filter Equations

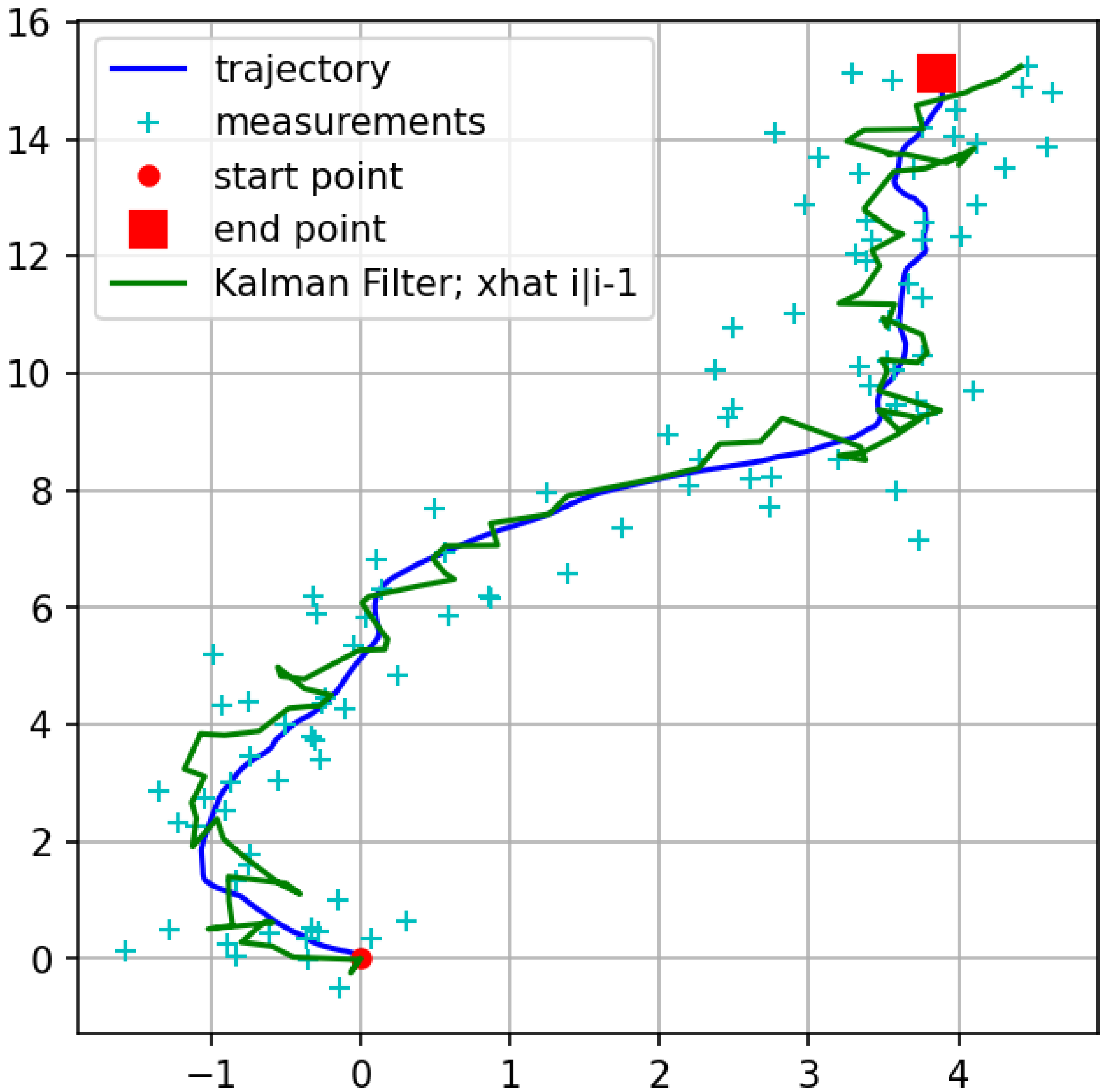
(see written work)

Q-6 & Q-7 Implementing + Plot the Kalman Filter

In [3]:
```
# initialize all of the variables that we're going to need; F, H, G, Q, R
Delta = 0.1
Fk = np.array([[1,0,Delta,0],[0,1,0,Delta],[0,0,1,0],[0,0,0,1]])
Gk = np.identity(4)
Qk = np.array([[Delta**3/3,0,Delta**2/2,0],
              [0,Delta**3/3,0,Delta**2/2],
              [Delta**2/2,0,Delta,0],
              [0,Delta**2/2,0,Delta]])
Hk = np.array([[1,0,0,0],[0,1,0,0]])
Sigma = 0.5
Rk = Sigma**2 * np.identity(2)
```

In [10]:
```
# the initial guesses of x and P
xhat_init = np.zeros((4,1))
P_init = np.identity(4)
NumSteps = len(MeasurementY1)

# initializing the matrices for computing Kalman filter state evolution over time
xhat_i_pred = np.zeros((4,NumSteps))
xhat_i_curr = np.zeros((4,NumSteps))
P_i_pred = np.zeros((NumSteps,4,4))
P_i_curr = np.zeros((NumSteps,4,4))
Kfi_curr = np.zeros((NumSteps,4,2))

# start running the Kalman Filter, using the NoisyMeasurements
for t in np.arange(0,NumSteps):
    # make the prediction update (time update)
    if t == 0:
        # use the initial guesses
        xhat_i_pred[:,t] = (Fk @ xhat_init).reshape(1,4)
        P_i_pred[t,:,:] = Fk @ P_init @ Fk.T + Gk @ Qk @ Gk.T
    elif t > 0:
        # then use the time-update (prediction) calculation
        # x i|i-1
        xhat_i_pred[:,t] = (Fk @ xhat_i_curr[:,t-1]).reshape(1,4)
        # P i|i-1
        P_i_pred[t,:,:] = Fk @ P_i_curr[t-1,:,:] @ Fk.T + Gk @ Qk @ Gk.T

    # make the measurement update
    # K f,i
    Kfi_curr[t,:,:] = P_i_pred[t,:,:] @ Hk.T @ np.linalg.inv(Hk @ P_i_pred[t,:,:] @
                                                          Hk.T + Rk)
    # P i|i
    P_i_curr[t,:,:] = (np.identity(4) - Kfi_curr[t,:,:] @ Hk) @ P_i_pred[t,:,:]
    # x i|i
    xhat_i_curr[:,t] = xhat_i_pred[:,t] + Kfi_curr[t,:,:] @ (NoisyMeasurements[:,t] -
                                                          Hk @ xhat_i_pred[:,t])


# plot the results
plt.figure()
plt.plot(StateX1,StateX2,'b')
plt.plot(MeasurementY1,MeasurementY2,'c+')
plt.grid(True)
plt.plot(StateX1[0],StateX2[0],'r.',markersize=10)
```

```
plt.plot(StateX1[-1],StateX2[-1],'rs',markersize=10)
plt.plot(xhat_i_pred[0,:],xhat_i_pred[1,:],'g')
plt.legend(['trajectory','measurements','start point',
            'end point','Kalman Filter; xhat i|i-1'])

plt.rcParams['figure.figsize'] = [4, 4]
plt.rcParams['figure.dpi'] = 120 # 200 e.g. is really fine, but slower
plt.show()
```



Q-8 RMS Error

In [5]:
```
StateX = State[:,::DownSampling][0:2,:]
error_measure = NoisyMeasurements - StateX
rms_measure = np.linalg.norm( np.linalg.norm( error_measure, ord=2, axis=0 ) ,ord=2)
error_kalman_f = xhat_i_pred[0:2,:] - StateX
rms_kalman_f = np.linalg.norm( np.linalg.norm( error_kalman_f, ord=2, axis=0), ord=2)
```

```
print(f'''RMS of the noisy measurements: {rms_measure}''')
print(f'''RMS of the Kalman filter estimates: {rms_kalman_f}''')
```

```
RMS of the noisy measurements: 7.085174167758084
RMS of the Kalman filter estimates: 3.889252921005451
```

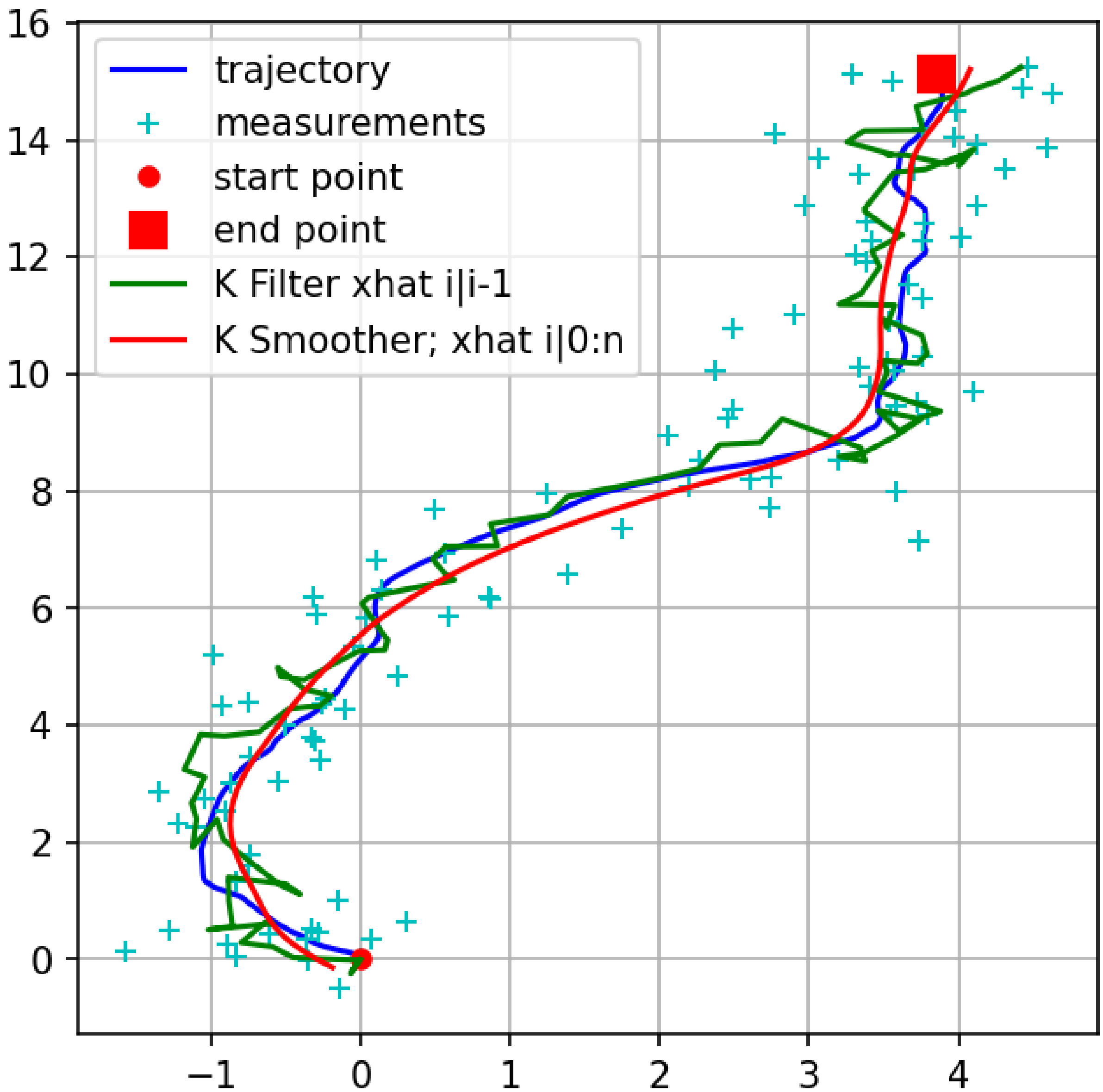
RMS Conclusion

Based on the above RMS calculation, the RMS of the Kalman filter is less than the RMS of the Noisy Measurements. This is indeed expected since we are using the Kalman filter to "filter" out some of the effects of the noise on our measurements.

Q-14 & Q-15 Implementation + Plot of Kalman Smoother

In [12]:
```
# Recall the initial setup for computing Kalman filter
# xhat_i_pred = np.zeros((4,NumSteps))          # xhat i|i-1
# xhat_i_curr = np.zeros((4,NumSteps))          # xhat i|i
# P_i_pred = np.zeros((NumSteps,4,4))           # P i|i-1
# P_i_curr = np.zeros((NumSteps,4,4))           # P i|i
# Kfi_curr = np.zeros((NumSteps,4,2))           # K f,i

# initialize new variables to hold the Kalman Smoother data
xhat_i_min_1_given_n = np.zeros((4,NumSteps))   # xhat i-1|0:n
Ki = np.zeros((NumSteps,4,4))                   # K_i

## Implement the Kalman Smoothing algorithm, given the Kalman Filtered data
# for xhat n|0:n, use the last step of the Kalman Filter
xhat_i_min_1_given_n[:,NumSteps-1] = xhat_i_curr[:,NumSteps-1]

# now for each time step compute the Kalman Smoothed state evolution
for t in np.arange(NumSteps-1,0,-1):
    # K_i
    Ki[t,:,:] = P_i_curr[t-1,:,:] @ Fk.T @ np.linalg.inv(Fk @ P_i_curr[t-1,:,:] @
                                                         Fk.T + Qk)
    # xhat i-1|0:n
    xhat_i_min_1_given_n[:,t-1] = (xhat_i_curr[:,t-1] +
                                   Ki[t,:,:] @ (xhat_i_min_1_given_n[:,t] -
                                                Fk @ xhat_i_curr[:,t-1]))


# plot the results
plt.figure()
plt.plot(StateX1,StateX2,'b')
plt.plot(MeasurementY1,MeasurementY2,'c+')
plt.grid(True)
plt.plot(StateX1[0],StateX2[0],'r.',markersize=10)
plt.plot(StateX1[-1],StateX2[-1],'rs',markersize=10)
plt.plot(xhat_i_pred[0,:],xhat_i_pred[1,:],'g')
plt.plot(xhat_i_min_1_given_n[0,:],xhat_i_min_1_given_n[1,:],'r')
plt.legend(['trajectory','measurements','start point','end point',
            'K Filter xhat i|i-1','K Smoother; xhat i|0:n'])
```

```
plt.rcParams['figure.figsize'] = [6,6]
plt.rcParams['figure.dpi'] = 150 # 200 e.g. is really fine, but slower
plt.show()
```



Q-16 RMS Error of Kalman Smoother

In [14]:
```
StateX = State[:,::DownSampling][0:2,:]
error_measure = NoisyMeasurements - StateX
rms_measure = np.linalg.norm( np.linalg.norm( error_measure, ord=2, axis=0 ) ,ord=2)
error_kalman_f = xhat_i_pred[0:2,:] - StateX
rms_kalman_f = np.linalg.norm( np.linalg.norm( error_kalman_f, ord=2, axis=0), ord=2)
error_kalman_f_curr = xhat_i_curr[0:2,:] - StateX
rms_kalman_f_curr = np.linalg.norm( np.linalg.norm( error_kalman_f_curr,
                                                    ord=2, axis=0), ord=2)
error_kalman_s = xhat_i_min_1_given_n[0:2,:] - StateX
rms_kalman_s = np.linalg.norm( np.linalg.norm( error_kalman_s, ord=2, axis=0), ord=2)
```

```
print(f'''RMS of the noisy measurements: {rms_measure}''')
print(f'''RMS of the Kalman Filter xhat i|i-1 estimates: {rms_kalman_f}''')
print(f'''RMS of the Kalman Filter xhat i|i estimates: {rms_kalman_f_curr}''')
print(f'''RMS of the Kalman Smoother xhat i|0:n estimates: {rms_kalman_s}''')
print(f'''RMS Kalman Smoother/RMS Kalman Filter: {rms_kalman_s/rms_kalman_f}''')
```

```
RMS of the noisy measurements: 7.085174167758084
RMS of the Kalman Filter xhat i|i-1 estimates: 3.889252921005451
RMS of the Kalman Filter xhat i|i estimates: 3.3022335707201216
RMS of the Kalman Smoother xhat i|0:n estimates: 2.2990220257280076
RMS Kalman Smoother/RMS Kalman Filter: 0.5911217584516627
```

RMS Conclusion

Based on the above RMS calculation, the RMS of the Kalman Smoother is ~41% less than the RMS of the Kalman Filter. This is good news since we expect our RMS to improve when using all of the measurments for each step.