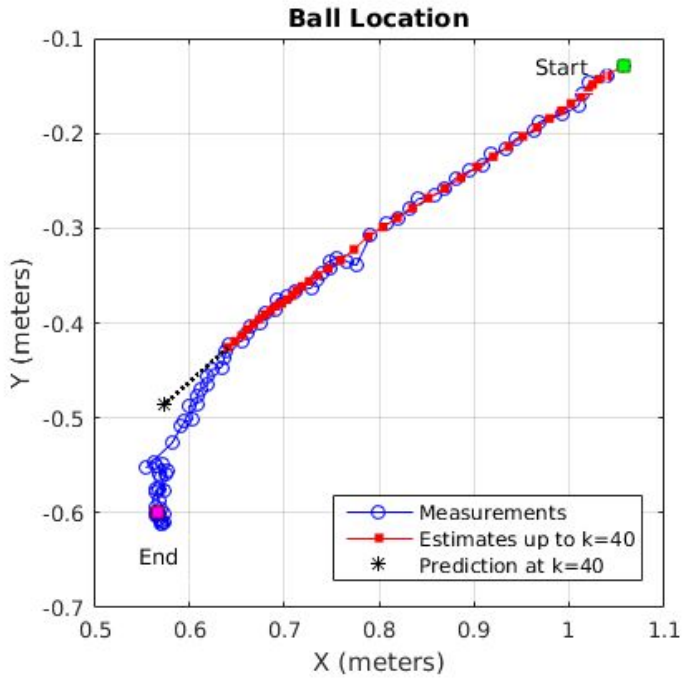
Programming Assignment Week 2

Kalman Filter–Target Tracking

1. Introduction

In this assignment you will be implementing a Kalman filter for ball tracking in 2D space. Imagine a soccer robot wants to predict the ball's location constantly for planning its next motion. The figure below depicts an example of ball location measurements, location estimates, and a predicted location at certain time.



2. Instructions

Phase 1. Design your KF Model

1. **Dynamics Model** (= System Model)
    a. In order to have an estimate of 2D location and predict a future location, the **state vector** (s) should include velocity (v) as well as position (p) of both dimensions, x and y.

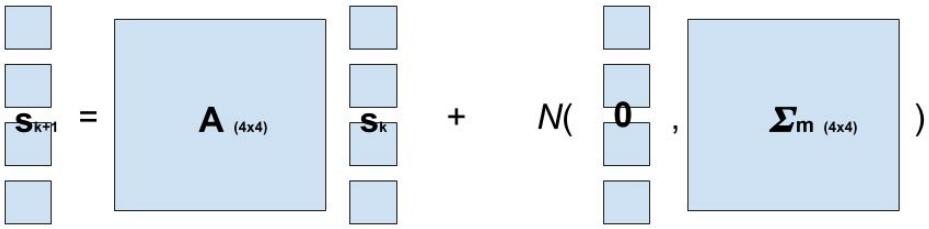
    Example: s = [px vx py vy]' or s = [px py vx vy]'

    * Note that the symbol *s* is used to indicate the state vector, which is denoted as x in the lecture material. This is to avoid confusion with the notation x of the Cartesian coordinate x, y.

b. Define your **transition matrix** A (See p.3 of slides [2-2-1]), where dt = 0.033 sec.

c. Now consider the type of the **system noise covariance** Σ_m . The most simple form is a diagonal matrix, with the four variances: Σ_m = *diag*([σ_{px}^2 , σ_{py}^2 , σ_{vx}^2 , σ_{vy}^2]) or Σ_m = *diag*([σ_{px}^2 , σ_{vx}^2 , σ_{py}^2 , σ_{vy}^2]).
In general, you don't have to restrict the form, but a diagonal matrix is a good starting point if you do not have a strong reason not to do so. How to set the values? It will be explained Phase 2.

System Model

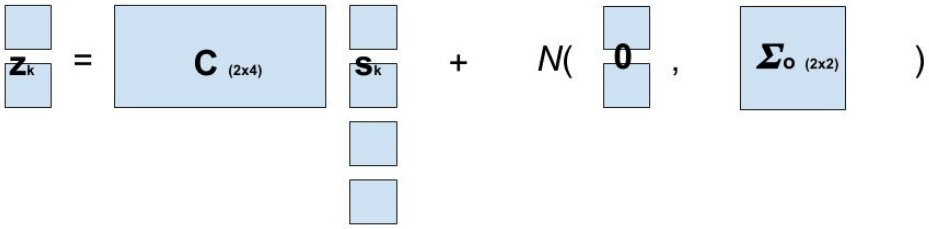


2. Measurement Model

a. The measurement data is readily in the 2D location unit.
z = [zx zy]' = [px py]'
Hence the **observation matrix** C will be a 2-by-4 matrix with two ones and six zeros. Please figure out what C is by yourself!

b. Consider the type of the **measurement noise covariance** Σ_o . The most simple form is a diagonal matrix, with the two variances: Σ_o = *diag*([σ_{zx}^2 , σ_{zy}^2]). We will see how to set the values in Phase 2.

Measurement Model



3. Review the models [Advanced option].

a. If you have chosen (block-)diagonal forms for the noise covariances, the x and y components are now ***uncorrelated***. Then you can simplify the model and have two independent identical filters. The simplified filter has s = [p v]', 2-by-2 A and Σ_m , z = p, the scalar variance Σ_o , and 1-by-2 C.

System Model

$$\begin{bmatrix} s_{k+1} \end{bmatrix} = \begin{bmatrix} A_{(2x2)} \end{bmatrix} \begin{bmatrix} s_k \end{bmatrix} + N\left(\begin{bmatrix} 0 \end{bmatrix}, \begin{bmatrix} \Sigma_{m\ (2x2)} \end{bmatrix} \right)$$

Measurement Model

$$\begin{bmatrix} z_k \end{bmatrix} = \begin{bmatrix} C_{(1x2)} \end{bmatrix} \begin{bmatrix} s_k \end{bmatrix} + N\left(\begin{bmatrix} 0 \end{bmatrix}, \begin{bmatrix} \Sigma_{o\ (scalar)} \end{bmatrix} \right)$$

You only need to keep two instances of this filter, one for x, and the other for y. This idea (separating uncorrelated systems) reduces the computational load, which can be effective for high-dimensional systems.

* Note that this reduction is optional in this assignment. If you got more confused because of Step 3.a, then just ignore it and go ahead to Phase 2.

b. If you have correlated terms, that is fine. You will keep the model as it is.

Phase 2. Set Parameter Values

1. In order to set the initial values of the state and state error covariance, you may use the first measurement as the position, and set zero for the velocity. Also, you may set the state covariance P to be a very large identity matrix, which implies a large uncertainty of the state.
2. You also need to set the model noise parameters Σ_m and Σ_o . Think about a rough yet reasonable range of the parameters by considering the physical units of the noise/uncertainty model of the problem.

   For example, If the measurement noise is in meters, then it could range from O(0.01) to O(0.1). Similarly, if the dynamics model includes position and velocity, then they have units such as meter and meter/sec. Because your system is a small ball rolling on the floor, the velocity will NOT be as large as 10 meter/sec.
3. **Coarse Tests**
   a. Still you will have many many choices. Now you can test the numbers of the reasonable range FIRST COARSELY.
   b. One good starting point is to set the system model noise covariance to have unreasonably large (for example, 1e^6), and the measurement noise covariance to be small in the reasonable range. Then your KF should just follow your measurements, unless you have a bug in your code.
   c. After that, you can test different values of the system model noise parameter.
4. **Fine Tuning**
   a. Once your KF seems to work, then you can fine-tune the parameters.
   b. You can even try more complex parameters, such as non-diagonal covariance matrix or adaptive (non-constant) covariances. We suggest that you do this ONLY AFTER your simplest KF works.
5. Be careful, the variance or covariance takes a squared form.

Phase 3. Implementation

1. You will complete a function that takes the state, parameters, and sensor data (x,y)as input, and returns a ball position prediction, possibly useful for goalie applications. The signature of the function is given as:

```
function [ predictx, predicty, state, param] = kalmanFilter( t, x, y, state, param, previous_t )
```

2. Please pass the state and state covariance as input/output arguments. Thus *param* should have the following fields:

   param.P

   You can name it differently, but make sure that you keep track of the state covariance P. (If you have two independent filters, then you may have *param*.Px and *param*.Py). You may use *param* to add other fields if you need.
3. Your function should return the predicted x and y positions of the ball at a time 330ms into the future. With a rate of 30Hz, this 330ms in time represents 10 frames in the future. In the kalmanFilter function, a naive implementation shows how to use instantaneous velocity from only two frames and no state to calculate the future position of the ball, returned as predictx and predicty. You will keep track of the four state components, comprised of x position, y position, x velocity, and y velocity. This state will be fed into each function call.
4. example_test.m is provided to help visualize your result.

Phase 4. Evaluation and submission

To submit your result to our server, you need to run the script runeval in your MATLAB command window. Please specify the path where the encrypted test data are located. A script will then evaluate your kalmanFilter function and generate an output file, **SubmissionFilter.mat**, to be uploaded to the Coursera web UI. You may submit your result multiple times, and we will count only the highest score towards your grade. The score reflects how close your predicted ball positions are to the observed ball positions.