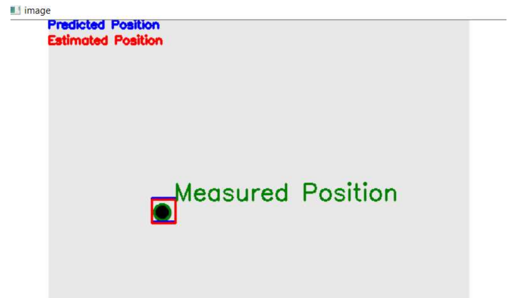
Exercise 1: Single Object Tracking with Kalman (Centroid-Tracker)

Objective: Implement object tracking in a 2D (two-dimensional) using pre-existing object detection algorithm and integrate the Kalman Filter for smooth and accurate tracking.

- Object representation: point (centroid)
- Single Object Tracking



Provided Materials:

- Video: A video containing sequence of frames with one object to be tracked
- Object Detection Code: Detector.py contains one function for detecting multi-objects in each frame using Canny edge detection. It returns the centers of detected objects.

To download the files, click on this link.

Python implementation:

The project will have two additional files: KalmanFilter.py and objTracking.py

1. Kalman Filter Implementation
   1.1 Create KalmanFilter.py. This file must contain one class called KalmanFilter consisting of three functions: __init__(), predict(), update()
   1.2 **Function Initialization**: __init()__ . The class will be initialized with six parameters:
      - *dt* : time for one cycle used to estimate state (sampling time)
      - *u_x, u_y* : accelerations in the x-, and y-directions respectively
      - *std_acc*: process noise magnitude
      - *x_sdt_meas, y_sdt_meas* :standard deviations of the measurement in the x- and y-directions respectively
   - Define:
      - Control input variables *u=[u_x, u_y]*
      - Initial state matrix: $(\widehat{x_k})=[x_0=0, y_0=0, v_x=0, v_y=0]$
      - Matrices describing the system model *A,B* with respect to the sampling time dt (Δt) :

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

$$\mathbf{B} = \begin{bmatrix} \frac{1}{2}(\Delta t)^2 & 0 \\ 0 & \frac{1}{2}(\Delta t)^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \quad (7)$$

      - Measurement mapping matrix *H*

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (10)$$

      - Initial process noise covariance matrix *Q* with respect to the standard deviation of acceleration (*std_acc*) σ_a :

$$\mathbf{Q} = \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & \Delta t^2 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & \Delta t^2 \end{bmatrix} \sigma_a^2 \quad (12)$$

- Initial measurement noise covariance *R.* Suppose that the measurements z (*x, y)* are both independent (so that covariance x and y is 0), and look only the variance in the x and y: *x_sdt_meas* $=\sigma_x^2$ *, y_sdt_meas*$=\sigma_y^2$

$$\mathbf{R} = \begin{array}{c c} & \begin{array}{cc} x & y \end{array} \\ \begin{array}{c} x \\ y \end{array} & \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \end{array} \tag{13}$$

- Initialize covariance matrix *P* for prediction error as an identity matrix whose shape is the same as the shape of the matrix *A*.

1.3 **Function predict()**

This function does the prediction of the state estimate $\hat{x}_k^-$ and the error prediction P_k^-. This task also call the time update process (*u*) because it projects forward the current state to the next time step.

- Update time state

$$\hat{\mathbf{x}}_k^- = A\hat{\mathbf{x}}_{k-1} + B\mathbf{u}_{k-1}$$

- Calculate error covariance

$$\mathbf{P}_k^- = A\mathbf{P}_{k-1}A^T + \mathbf{Q}$$

1.4 **Function: update ()**

This, function takes measurements z_k as input (centroid coordinates x,y of detected circles)

- Compute Kalman gain

$$S_k = HP_k^-H^T + R$$
$$K_k = P_k^-H^TS_k{}^{-1}$$

- Update the predicted state estimate $\widehat{x_k}$ and predicted error covariance P_k

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - H\hat{\mathbf{x}}_k^-)$$

$$\mathbf{P}_k = (I - \mathbf{K}_kH)\mathbf{P}_k^-$$

2. Create the main file of this project that will be execute to track an object (objTracking.py)

2.1 Import function detect() and KalmanFilter

2.2 Create the object of the class Kalman filter and set parameters values as:

- *dt=0.1, u_x=1, u_y=1, std_acc=1, x_dt_meas=0.1, y_dt_meas=0.1*

You can also try to set other values and observe the performance.

2.3 Create video capture object

2.4 Object Detection Integration. Use provided object detection code to detect black cercle in each frame.

2.5 If centroid is detected then track it. Call the Kalman prediction function and the Kalman filter updating function.

2.6 Visualize for tracking results:

- Draw detected circle (green color)
- Draw a blue rectangle as the predicted object position
- Draw a red rectangle as the estimated object position
- Draw the trajectory (tracking path) in an image