EEL 4930/5934: Autonomous Robots
HW5: Robot Localization and Filtering (Spring 2025)

Tasks Overview:

A. 3D Robot localization from 3 landmarks (30%)
B. Kalman filtering for 2D object tracking in images (30%)
C. Bounding box tracker using KF (40%)

Reference:

- Lecture 8 materials
- Download the **HW5_Blank** folder from Canvas

Submission:

- Through Canvas only; Due: April 21, 2025 by 11.59 PM
- A single zip file with a folder (code) and PDF (report)
  - A PDF report for Part-A and Part-C
    - Handwritten solutions are fine as long as they are clear and readable
    - No need to write/draw anything for Part-B.
    - Write your formulation for Part-C (similar to Slide-35 of Lecture-8):
      - Report the $\mathbf{x}_t$, $\mathbf{F}_t$, $\mathbf{B}_t$, $\mathbf{H}_t$, $\mathbf{Q}_t$, and $\mathbf{R}_t$ matrices.
  - Your completed code
    - For Part-B and Part-C. No need to add any files, just add your code to complete the class or functions left for you.
    - You CAN NOT use any Kalman Filtering libraries or packages
    - See the individual assignment instructions below.
- Assignments of more than 25 MB in file size will get a negative penalty (-10% to -25%)

Part A: 3D Robot localization from 3 landmarks

Refer to the concepts of robot localization from landmarks in Lecture-8.

Given: coordinates of 3 non-planar points

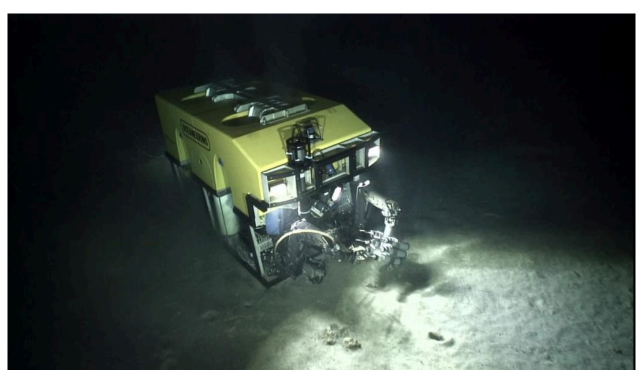
$$^GP_1 = \begin{bmatrix} ^Gx_1 \\ ^Gy_1 \\ ^Gz_1 \end{bmatrix},\ ^GP_2 = \begin{bmatrix} ^Gx_2 \\ ^Gy_2 \\ ^Gz_2 \end{bmatrix},\text{ and } ^GP_3 = \begin{bmatrix} ^Gx_3 \\ ^Gy_3 \\ ^Gz_3 \end{bmatrix}$$

$$^AP_1 = \begin{bmatrix} ^Ax_1 \\ ^Ay_1 \\ ^Az_1 \end{bmatrix},\ ^AP_2 = \begin{bmatrix} ^Ax_2 \\ ^Ay_2 \\ ^Az_2 \end{bmatrix},\text{ and } ^AP_3 = \begin{bmatrix} ^Ax_3 \\ ^Ay_3 \\ ^Az_3 \end{bmatrix}$$

Find the position $^G\boldsymbol{P}_A$ and rotation matrix $^G\boldsymbol{R}_A$

Where

$$^G_A\mathbf{T} = \left[\begin{array}{ccc:c} & ^G_AR & & ^GP_A \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right]$$



Part B: Kalman filtering for 2D object tracking in images

You will implement the basic Kalman Filtering algorithm for 2D object tracking in images.

- Download the **HW5_Blank** folder
- For Part-B, you will implement a simple **circle tracker** based on KF
- Template code is given for you; the two files relevant for Part-B are:
  - `circleTracker.py`
  - `KF_2D.py`
- We need to **track** two variables: the center of the circle (`u_x, u_y`)
- So this is a 2D tracker, what are we tracking? we are tracking the detected center of a ball moving around in the image frame. **Check the data/rBall.avi**
- Check the **`circleDetector`** function that detects the ball and returns a center
- We will then use the **`KF_2D`** filter to track them by **Predict** and **Update** rules of KF
- The following skeleton code should then be enough to test your code

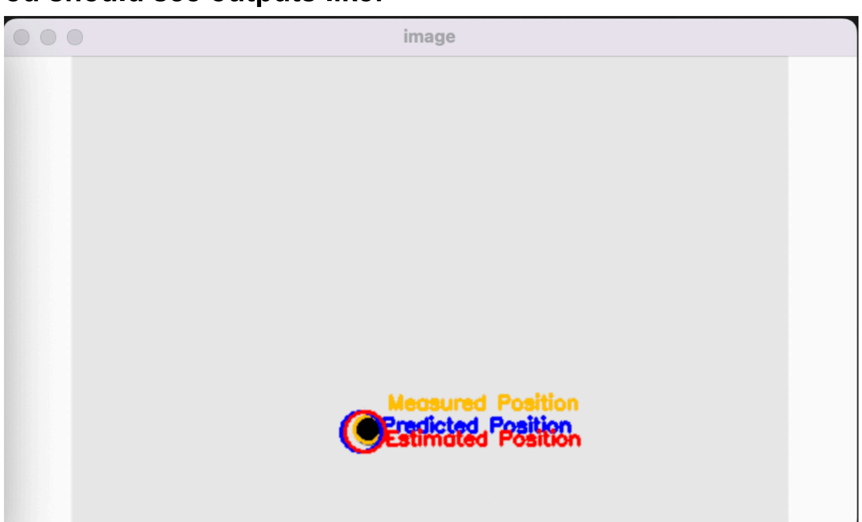
```
filter = KF_2D(dt=0.1, u_x=1, u_y=1) # define the filter
centers = circleDetector(frame) # Detect object

if (len(centers) > 0):
    cv2.circle    # draw circle[0]
    cv2.putText    # write "Measured Position" on the frame

    # Predict
    (x, y) = filter.predict()
    cv2.circle(frame, (x,  y), 15, (255, 0, 0), 2)  # draw circle
    cv2.putText    # write "Predicted Position" on the frame

    # Update
    (x1, y1) = filter.update(centers[0])
    cv2.circle(frame, (x1, y1), 15, (0, 0, 255), 2)  # draw circle
    cv2.putText # write "Estimated Position" on the frame
```

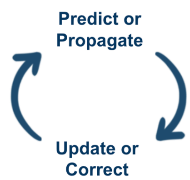
You should see outputs like:



The **`circleDetector`** function is completed for you. It used some basic OpenCV tricks to extract the ball foreground, then enclose a contour around it to detect the blob's center (contour). Check the sequence of steps for the detection windows below.

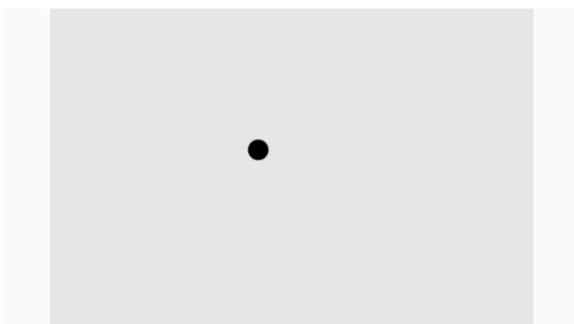
That center is used as (`u_x, u_y`) for the circle, on which KF is applied for tracking. The skeleton of the class **`KF_2D`** is also given to you for your convenience. Please complete the rest of the functions as discussed in the KF formulation (Lecture-8).

$$\begin{aligned} \text{State transition:} \quad x_t &= \mathbf{F}_t\, x_{t-1} + \mathbf{B}_t\, u_t + \epsilon_t \\ \text{Observation:} \quad z_t &= \mathbf{H}_t\, x_t + \delta_t \end{aligned}$$

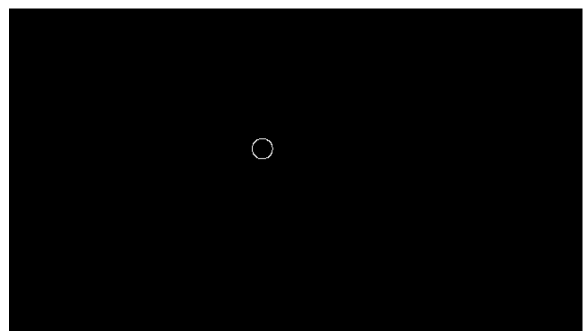


$$\begin{aligned} \text{Predict State:} \quad x_t &= \mathbf{F}_t\, x_{t-1} + \mathbf{B}_t\, u_t \\ \text{Predict covariance:} \quad \mathbf{P}_t &= \mathbf{F}_t\, \mathbf{P}_{t-1}\, \mathbf{F}_\mathbf{t}^\mathbf{T} + \mathbf{Q}_t \end{aligned}$$

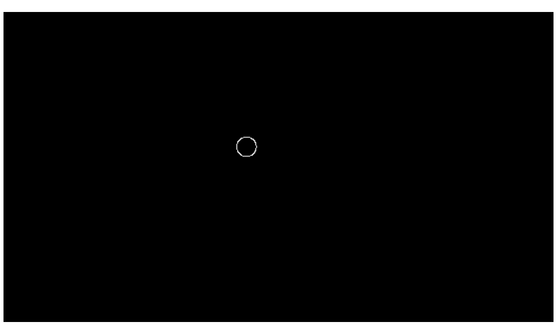
$$\begin{aligned} \text{Observation residual:} \quad y_t &= z_t - \mathbf{H}_t\, x_{t-1} \\ \text{Observation covariance:} \quad \mathbf{S}_t &= \mathbf{H}_t\, \mathbf{P}_{t-1}\, \mathbf{H}_\mathbf{t}^\mathbf{T} + \mathbf{R}_t \\ \text{Kalman gain:} \quad \mathbf{K}_t &= \mathbf{P}_t\, \mathbf{H}_\mathbf{t-1}^\mathbf{T}\, \mathbf{S}_\mathbf{t}^{-1} \\ \text{Update state:} \quad x_t &= x_t + \mathbf{K}_t\, y_t \\ \text{Update covariance:} \quad \mathbf{P}_t &= (\mathbf{I}_t - \mathbf{K}_t\, \mathbf{H}_\mathbf{t})\, \mathbf{P}_t \end{aligned}$$



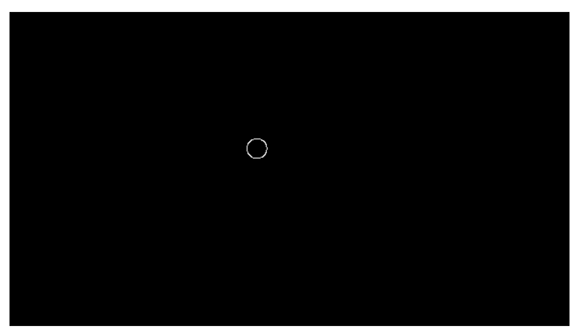
Sample input images from the input video



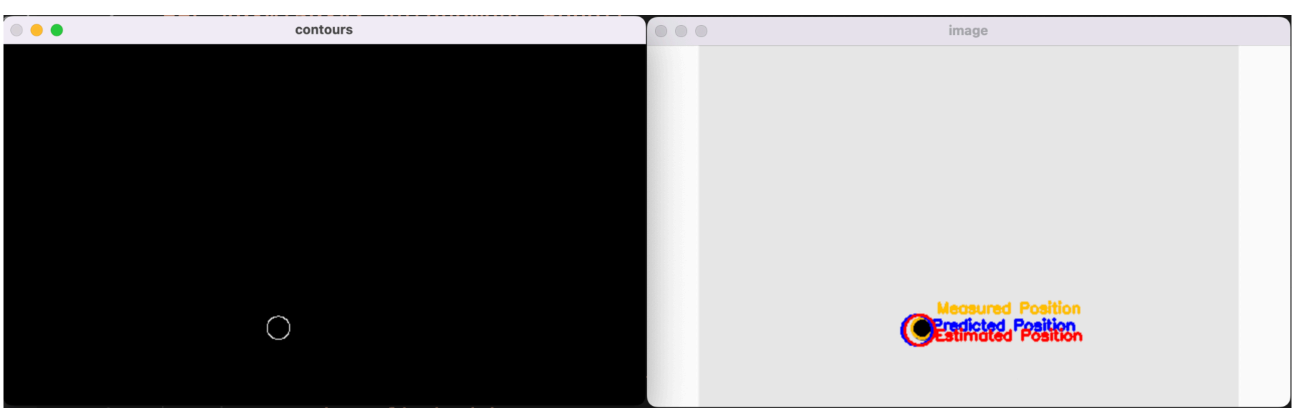
circleDetector: Detect edges using OpenCV



circleDetector: Convert to binary by thresholding



circleDetector: Find the object contour (circle)



Finally, apply the 2D KF to continuously **Predict** and **Update** the estimated positions

Part C: Kalman filtering bounding box tracking in images

You will now extend the 2D filter to track a bounding box on the same setup.

- That is, you will implement a simple **box tracker** based on KF
- Template code is given for you; the two files relevant for Part-B are:
  - `boxTracker.py`
  - `KF_4D.py`
- We need to **track** four variables for the bounding box: (`u_x, u_y, u_w, u_h`)
- So this is a 4D tracker, you need to first implement
  - The **`boxDetector`** function to return a rectangle
  - Hint: previously, **`circleDetector`** function returned a circle, you can extend this a little bit to return an enclosing rectangle instead!
- We will then use the **`KF_4D`** filter to track them by **Predict** and **Update** rules of KF
- A similar bounding box tracker is discussed in Lecture-8
  - Think about how to formulate it for this particular problem
  - The output will be much smoother than the circle tracker, if your formulation is good
    - This is slightly open-ended, so try a few things!
    - Study how to design and tune the control variables for KF problems like this one. Bounding box tracking is a popular applications, so you should be able to find some resources online as well!
  - In addition to completing your code, add the following information **in report**
    - Present your formulation (similar to Slide-35 of Lecture-8)
    - Write the $\mathbf{x}_t$, $\mathbf{F}_t$, $\mathbf{B}_t$, $\mathbf{H}_t$, $\mathbf{Q}_t$, and $\mathbf{R}_t$ matrices.

Outputs should like the following: