ADA527 Lab-2: Qube and Kalman filter

So, you are familiar with the Qube system thanks to Lab-1. It is basically a 2 DoF inverted pendulum or also called "rotary pendulum". You can control the theta and alpha angles to, let's say, balance the pendulum.

What if you want to know about where the center point of the red rod is while you are doing some motions? How can you calculate it? Should you use a camera and basically implement a *red rod tracker* algorithm and find the middle point? Or should you use the encoder readings and do some kinematics to calculate the center of the rod?

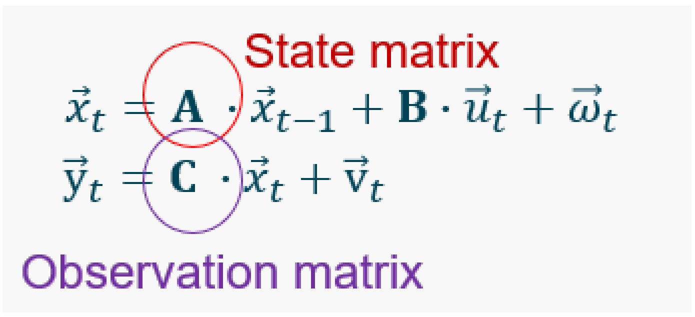
Well, we know that both methods have some limitations:

1. When you use a camera, you have to calibrate it so well that you would have a reliable *red rod tracker* algorithm. Moreover, you must keep in mind that any obstacles between the camera and the Qube is your enemy. Also, you better to spend a lot of money to have a camera with exceptionally good resolution and FPS.

2. When you use the system model, you are limited by the encoder resolution and how well the system model is defined. Luckily, the encoders are pretty good in this overly priced device. However, depending on your system model, the pose of center point of the rod would be less accurate. Also, any inaccuracies on the encoders would be multiplied by the link lenghts and would cause even bigger errors on the position of the center of the rod.

We know by now that it is better to *fuse* the things that to increase reliability of the estimation. Kalman filter is the best for this purpose. Therefore, this lab is about fusing the system model of the Qube and observations via camera to estimate the position of the center of the rode more reliably.

For that, we need to express things in this format.


State matrix
$\vec{x}_t = \mathbf{A} \cdot \vec{x}_{t-1} + \mathbf{B} \cdot \vec{u}_t + \vec{\omega}_t$
$\vec{y}_t = \mathbf{C} \cdot \vec{x}_t + \vec{v}_t$
Observation matrix

This lab consists of 2 parts. In the first part, you will be using a recorded data to complete the tasks. In the second part, you will connect your PC to Qube, record your own data and apply what you did in the first part on your data. Although, we will see how much time we will have for the next part :)

Important!

Please do not use HVL Onedrive as your project location in this lab. Choose a path (preferably short) and contains to space character in the path.

Part - 1: Use recorded data

Here you have the recorded data under **recorded_data_lab** folder. There is one mp4 and one CSV file. The csv file contains the pixel positions from both the system model and the image processing and the mp4 file is the raw video while the CSV data was recorded.

Deciding the state and observation matrices

As you know, **state matrix** is a matrix form of your **system model**. The system model is a mathematical expression that relates your system states to an output entity. The system states are alpha, theta, and their derivatives and the output is the pose of the center of rod in this setup. You can use various methods to *generate* such a mathematical expression.

It is the same for the **observation matrix**. There, you would write down a mathematical expression that links your states and whatever entity that you measure/observe/estimate.

a. A bit background/brainstorming

One way is just to use the state equations that you derived in Lab-1. Something like this:

$$\begin{bmatrix}\dot{\theta}\\ \dot{\alpha}\\ \ddot{\theta}\\ \ddot{\alpha}\end{bmatrix} = \underbrace{\frac{1}{J_t}\begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ 0 & m_p^2 l^2 rg & -J_p b_r & m_p l r b_p\\ 0 & -m_p g l J_r & m_p l r b_r & -J_p b_p\end{bmatrix}}_{A_c}\begin{bmatrix}\theta\\ \alpha\\ \dot{\theta}\\ \dot{\alpha}\end{bmatrix} + \underbrace{\frac{1}{J_t}\begin{bmatrix}0\\ 0\\ J_p\\ -m_p r l\end{bmatrix}}_{B_c}\tau.$$

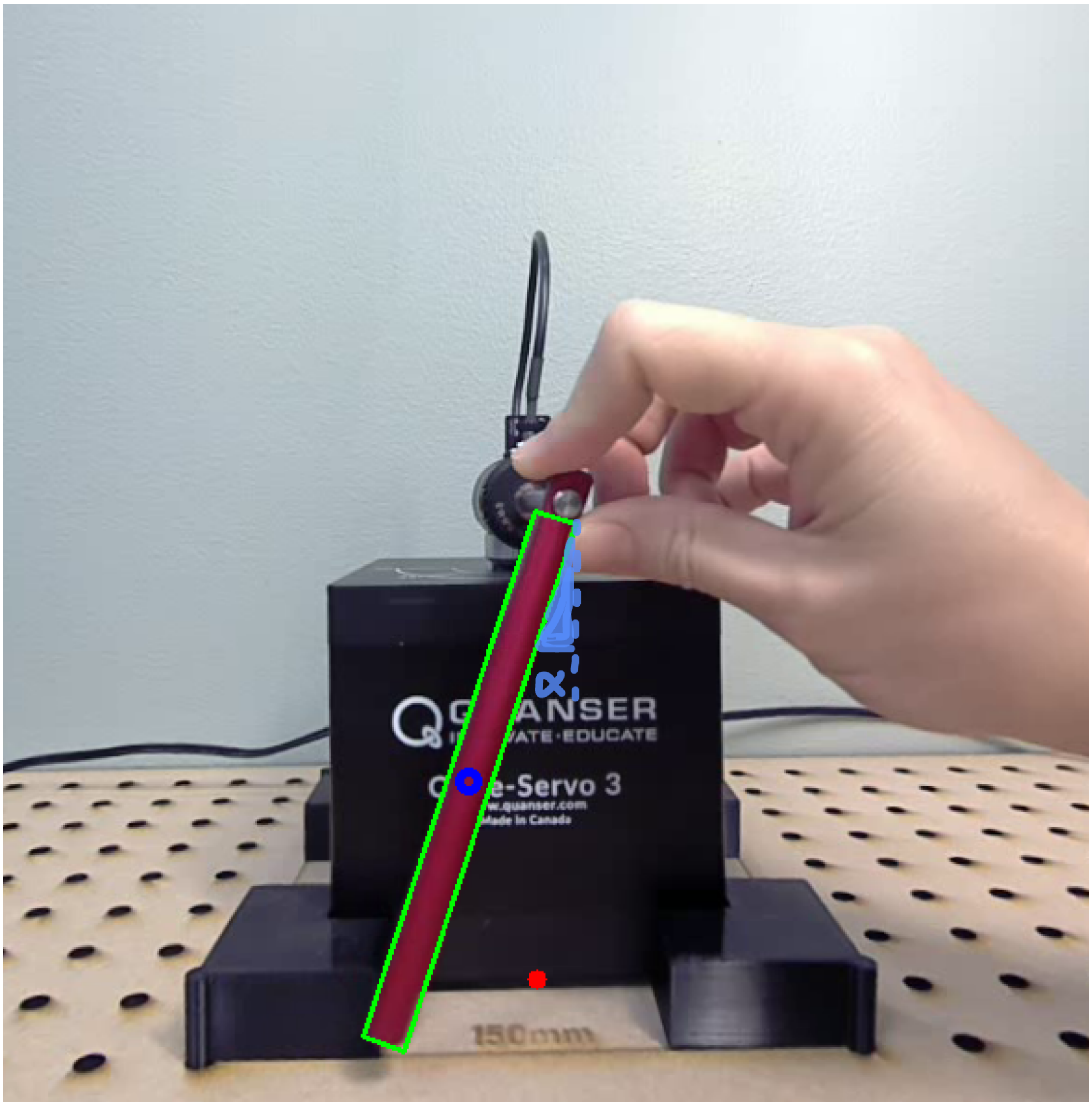
And if you used the encoder readings as your *observation*, then the observer equations would be like this:

$$\begin{bmatrix}\theta\\ \alpha\end{bmatrix} = \underbrace{\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\end{bmatrix}}_{C_c}\begin{bmatrix}\theta\\ \alpha\\ \dot{\theta}\\ \dot{\alpha}\end{bmatrix} + \underbrace{\begin{bmatrix}0\\ 0\end{bmatrix}}_{D_c}\tau.$$

However, we would like to elaborate with the camera readings in this lab.

At this point, you should take a decision. Are you planning to **observe the theta and alpha angles** or the **pixel positions** via your camera.

1. In the first option, you can extract the feature, let's say a red rectangle, in your frame, then calculate the angle. For alpha angle, it would look like this:



2. Or we can extract pixel positions, and they can be our observation. Then, our observer equations would rather be like in this format:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \alpha \\ \dot{\theta} \\ \dot{\alpha} \end{bmatrix}$$

where x and y are the pixel positions on our recorded image/video and shown as the blue circle in the picture above.

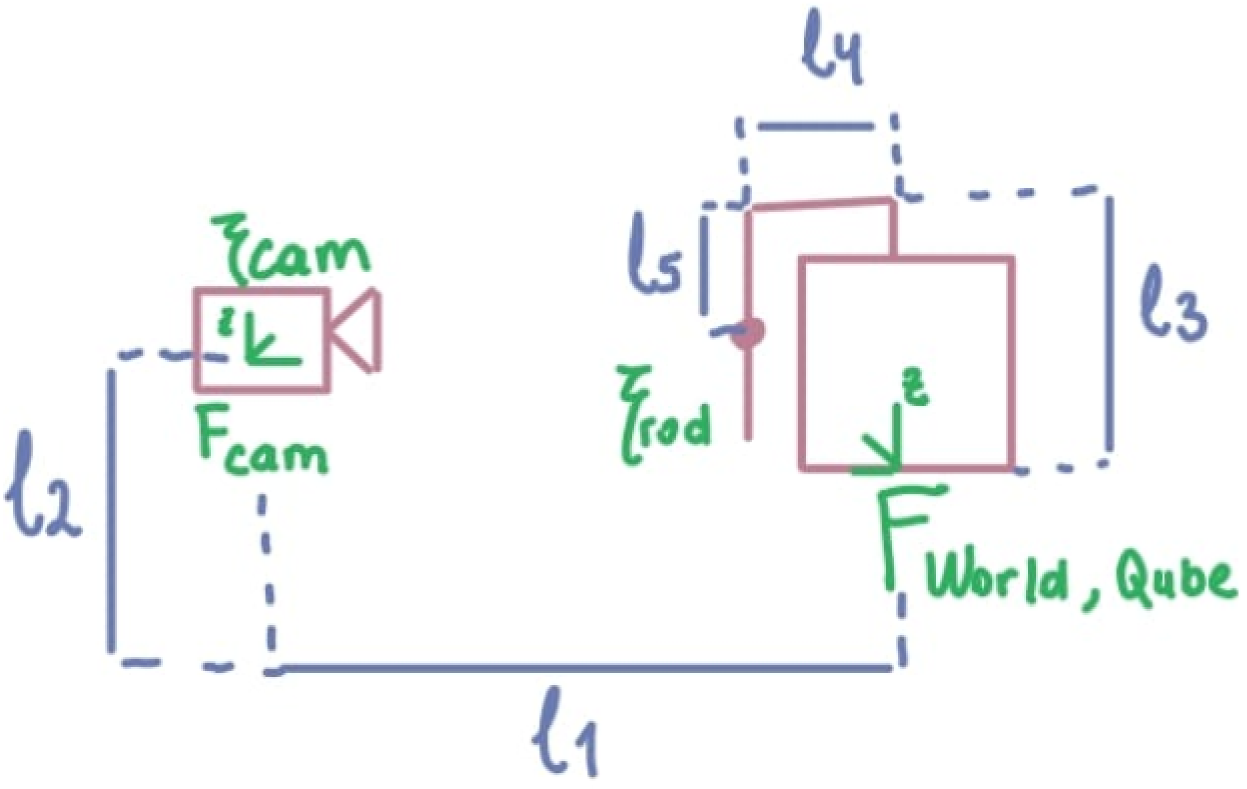
Disclaimer: The first method might given you better estimation results. You could/should EKF or UKF instead but due to "various limitations", we will go with the second method and use linear Kalman filter in this lab. However, you are given the equations and a lot of source code material. You can try the first method later at home and I'd be happy to support you whenever you need help.

b. Decision

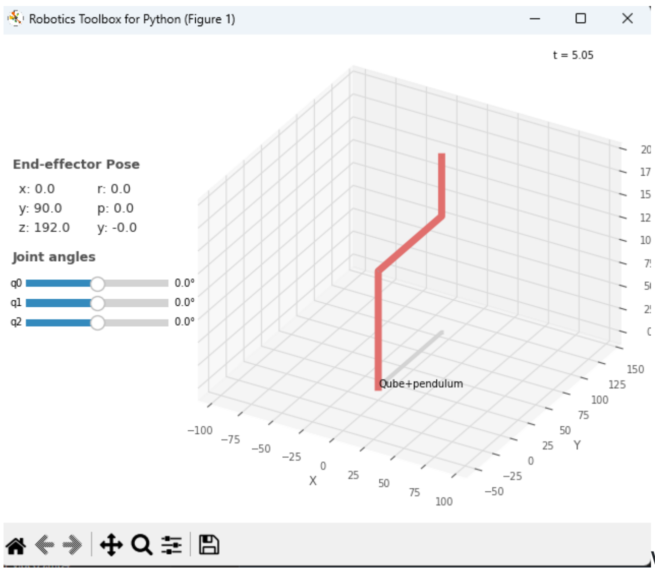
Since we decided to observe pixel positions, the procedure will be like this:

Obtain pixel positions using kinematics

1. Choose the relevant frame of references for the camera, Qube and the world. For simplicity, we can choose the world frame and the Qube frame coincided like this:



Obtain the position of the center of the red rod using robot kinematics in the Qube frame ${}^{Q}\xi_{rod}$. To do that, you should go **Classes>qube_class.py>kinematics (func)**. Create a 3 DoF robot that is representing the Qube. It should look like this:


Robotics Toolbox for Python (Figure 1)

t = 5.05

End-effector Pose

x: 0.0 r: 0.0
y: 90.0 p: 0.0
z: 192.0 y: -0.0

Joint angles

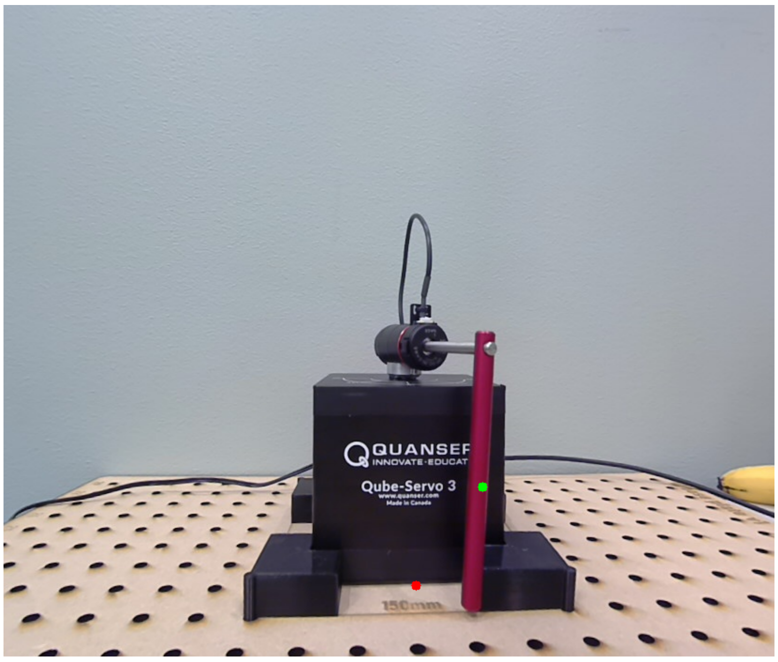
q0 0.0°
q1 0.0°
q2 0.0°

where q0=theta, q1=alpha, q2=0 (for end effector). For sanity check ${}^{Q}\xi_{rod} = {}^{W}\ \xi_{rod}$ = [-9.0000000e+01 2.4492936e-14 1.9200000e+02] when q0=0, q1=0, q2=0. You can simply run the script for test purposes since it has a main function implemented for you.

2. Transform end effector pose into the camera frame:

   ${}^{C}\xi_{rod} = {}^{W}\ R_C \cdot {}^{W}\ \xi_{rod}$. This one was tricky since the lens rotation in the camera was a bit weird. That's why it is given for you in **Classes>qube_class.py>qube_to_camera (func)**.

3. Transform ${}^{C}\xi_{rod}$ into pixel frame. To transform pixels into world coordinates, or vice versa, you would need a camera calibration matrix. We already provide the calibration matrix for you, that's why we don't get into details. Let's call these pixel points as $\begin{bmatrix} x_{enc} \\ y_{enc} \end{bmatrix}$ indicated that they were obtained from encoder readings (and thus via kinematics). Make sure that everything is fine by running **qube_camera_kinematics.py** and see a green dot in the center of the rod:



Note: You may check the relevant code for calibration under calibration folder.

Obtain pixel positions using kinematics

4. Now you need to obtain the pixel positions via camera. Let's call these pixel points as $\begin{bmatrix} x_{cv2} \\ y_{cv2} \end{bmatrix}$ indicated that they were obtained via image processing *using CV2 library*. This is pure image processing and outside the scope of this course. Therefore, we provide you the code that automatically detects a red rectangle (this is how the Qube's rod looks like in 2D). However, for the completion, we would like you to calculate the middle point of the rectangle using some simple math :)

   Write down the proper in **Classes>red_rectangle_class.py** in these lines:

   > middle_x = 150
   >
   > middle_y = 150

   where 150 is just a place holder. You should calculate the middle point of the rectangle using **x,y,w** and **h**.

   You can test if your calculations are right by running **pendulum_position_via_camera.py** script.

Implement the Kalman filter

Since you have both calculated and observed pixel positions, you can put everything into the matrix form. You are supposed to do some modifications in the script called **KF-qube-using-**

pixels.py between lines 12 and 21. Currently, the script is not working but as you write down correct dimensions and values into matrices, it will work. In the end, it will create a file called **estimated_data.csv** under **recorded_data_lab** folder.

5. Realize that we simplified the system model from pendulum angles to pixel tracking:

$$\begin{bmatrix} x_{enc} \\ y_{enc} \end{bmatrix} = \mathrm{f}(\theta, \alpha, \dot{\theta}, \dot{\alpha}) + \mathbf{B} \,.\, \tau \rightarrow \begin{bmatrix} x_{enc,k+1} \\ y_{enc,k+1} \end{bmatrix} = \boldsymbol{A} \,.\, \begin{bmatrix} x_{enc,k} \\ y_{enc,k} \\ \dot{x}_{enc,k} \\ \dot{y}_{enc,k} \end{bmatrix}$$

   Find the elements of matrix A:

   (Hint: remember the formula $x_{k+1} = x_k + \Delta t \cdot \dot{x_k}$ *)*

6. Therefore, our observation can be represented like this:

$$\begin{bmatrix} x_{cv2} \\ y_{cv2} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \end{bmatrix} \cdot \begin{bmatrix} \theta \\ \alpha \\ \dot{\theta} \\ \dot{\alpha} \end{bmatrix}$$

   Find the elements of matrix C:

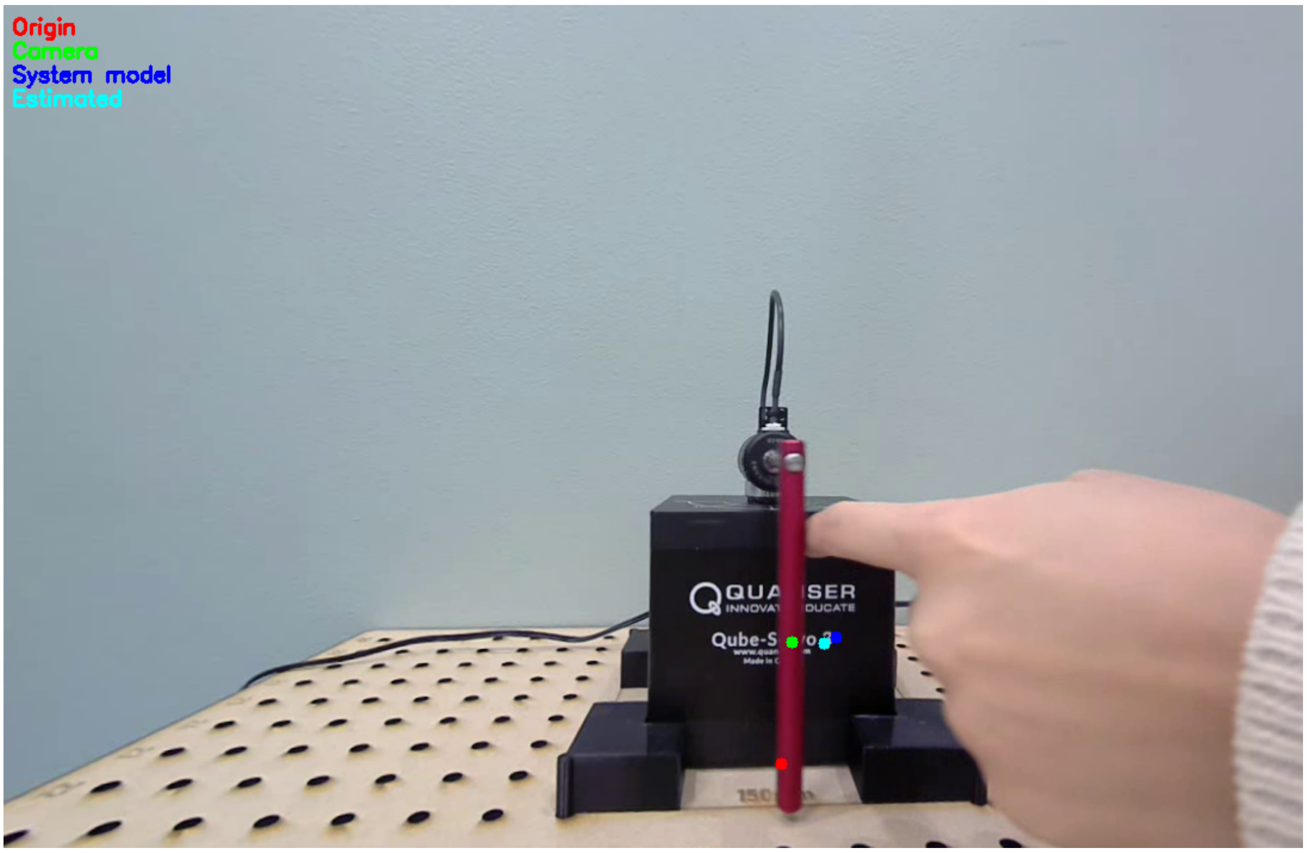
   (Hint: not all states are observable.)

7. What were Q, R and P matrices? Think about what they represent and decide their dimensions. np.eye(10) is wrong but you should use another value than 10.

   (Hint: you may want to check other Kalman filter exercises we did in the class.)

8. You are done! If you don't have any more errors in the script, you can visualize the estimated pixel points by running **visualize_estimated_positions.py**.

   Here you will see that my system model was very poor, like really really poor... You may want to improve it later, maybe :)



Part - 2: Record your own data and compare

The purpose of this step is to setup your Kalman filter and ensure that it is working on a system that you know what the output should be (somehow). In part-2, you will record your own data and use the KF that you configured in part-1.

Calibration

1. Make sure that the placements of the Qube and the camera is correct. For that, run **calibration>manual_calibration.py** and make sure the qube is properly in the red rectangle. Move the camera properly if needed.

2. You can skip this step if you are using the Huddly camera in the lab but if you are using another camera, you have to calibrate the camera, as well. For that,

    1. Have a checkerboard with 20x20 mm squares and total 7x9 squares.

    2. Run **calibration>checkerboard_take_picture.py** and take at least 10 pictures.

    3. Run **calibration>calibration_yaml_generator.py** and you will have a calibration_matrix.yaml file. Now you are done with camera calibration.

3. Time to record data. If you don't want to lose what is already recorded, you must either change the name of the already recorded data, or the names of the recordings in the code. Currently, the recorded video will be named as "output_video_with_encoders.mp4" and the CSV file will be "recorded_data.csv".

   For recording, you will use **qube_camera_kinematics_video.py**. (Sorry, this code is a mass, forgot to clean.)

   As you run this script, the recording will automatically start and will record for 30 seconds. Do some motions, NOW!

4. You are done with the recording. You can now test your KF implementation on your data:

   1. If you changed the recoding names, modify the following scripts accordingly.

   2. Run **KF-qube-using-pixels.py** to generate estimated pixel positions.

   3. Run **visualize_estimated_positions.py** to visualize them.