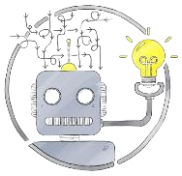


LAB5: HEIGHT ESTIMATION WITH A LINEAR KALMAN FILTER

OVERVIEW

In this lab, you will:

1. Acquire real-time distance data from **three** ST VL53L1 Time-of-Flight (ToF) sensors on the **X-NUCLEO-53L1A1** shield (attached to an STM32F411RE Nucleo board).
2. Implement a **discrete linear Kalman Filter** in several stages:
    - **Constant velocity** process model (using the *center* distance sensor) with the state in the order $\begin{bmatrix}\dot{x}\\ x\end{bmatrix}$].
    - **Constant acceleration** process model (still using the *center* sensor) with the state in the order $\begin{bmatrix}\ddot{x}\\ \dot{x}\\ x\end{bmatrix}$.
    - Fusing **all three** distance sensors for a more robust height estimate.
3. Explore the effects of filter tuning (process/measurement noise) and initialization on the estimated state.

Note: Each VL53L1 sensor outputs at **10 Hz** (once every 0.1 s). You will run the Kalman filter at **100 Hz** (every 0.01 s). Thus, you will do a *predict* step on **every** 100 Hz iteration and a *measurement update* only on every 10th iteration (matching the 10 Hz sensor rate). Please make sure you connect the board **X-NUCLEO-53L1A1 and distance sensors like the image below:**

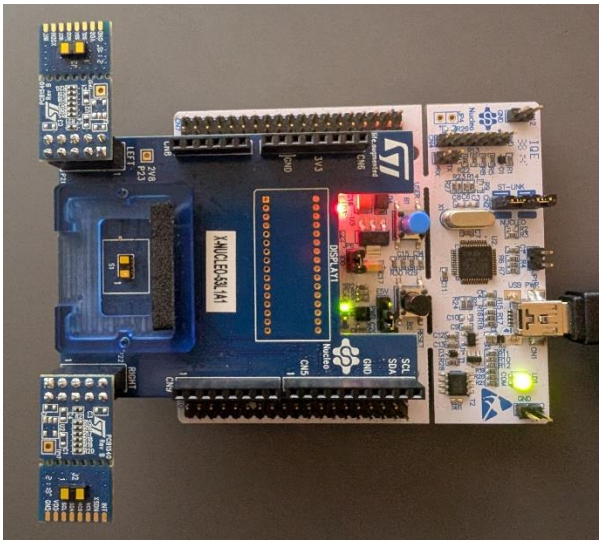
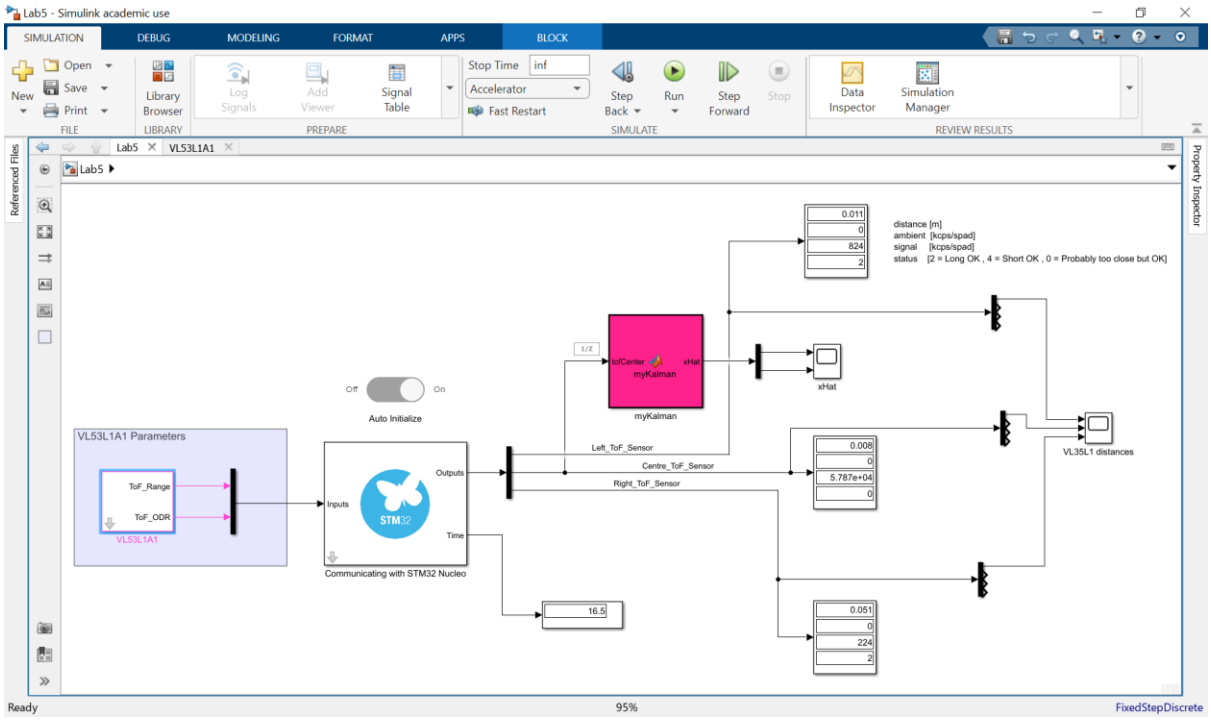


Figure 1: Nucleo board with ToF board mounted.

Assumptions

- The board is held *approximately parallel* to the desk, so each sensor measures roughly the same height above the surface.
- The measurement noise standard deviation (σ) of each ToF sensor is about **3 mm**.
- The distance sensors typically report a 4-element vector: [distance, ambient, signal, status]. We will use only the **distance**.
- You have been provided a Simulink model **lab5.slx** that already interfaces to all three sensors via an STM32 block. You only need to add Kalman code in the pink **myKalman** block.



TASK 1: READ TOF SENSOR DATA IN REAL TIME

Objective

Verify you can stream *live* distance data from the **VL53L1** sensors into Simulink.

Instructions

1. **Open** lab5.slx in Simulink.
2. **Connect** the STM32F411RE + X-NUCLEO-53L1A1 shield to your PC via USB.
3. In the model, look at the block labeled **"Communicating with STM32 Nucleo"**. It should output three signals (one for each ToF sensor): Left, Center, and Right.
4. Each sensor output is a 4×1 vector: **[distance in m, ambient, signal, status].**
5. **Open** the Scope (or Display blocks) labeled VL53L1 distances to observe the real-time signals.
6. **Click "Run"** in Simulink. Wave your sensor board and verify the distance readings respond in real-time.

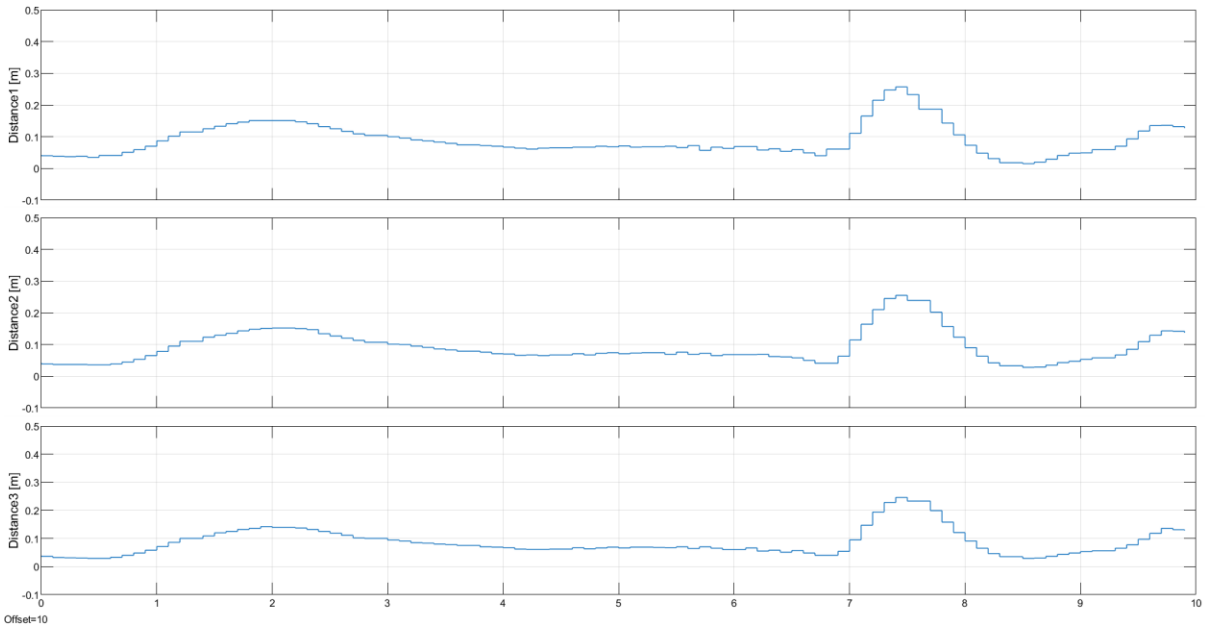


Figure 2: Scope output showing the three ToF sensors in real-time.

TASK 2: CONSTANT-VELOCITY KALMAN FILTER

Objective

Implement a **discrete Kalman Filter** with a *constant velocity* process model, but arrange your state vector as $\begin{bmatrix}\dot{x}\\ x\end{bmatrix}$]. You will use **only the centre** distance sensor for the measurement.

System Model

1. **State vector:**

$$\boldsymbol{X} = \begin{bmatrix}\dot{x}\\ x\end{bmatrix}$$

2. **Process model** (assuming constant velocity):

In matrix form (with Δt=0.0, because we run at 100 Hz) the state space model is:

$$\begin{bmatrix}\dot{x}_{k+1}\\ x_{k+1}\end{bmatrix} = \begin{bmatrix}1 & 0\\ \Delta t & 1\end{bmatrix}\begin{bmatrix}\dot{x}_k\\ x_k\end{bmatrix} + \boldsymbol{w_k}$$

Where $w_k \sim N(0, \sigma_v)$

3. **Measurement model (**only measuring x, i.e. height**)**:

$$y_k = [0 \quad 1] \begin{bmatrix} \dot{x}_k \\ x_k \end{bmatrix} + v_k$$

4. **Noise**

- with $R = (0.003)^2$ (since the ToF stdev $\sigma \approx 0.003$ m).
- Choose a modest $\sigma = 0.1$ to allow some variability in the constant-velocity assumption.

Instructions

1. **Open** the pink **myKalman** block and replace its contents with something like the code below.
2. In the block's dialog, ensure there is **1 input** (the center sensor's 4×1 vector) and **1 output** (the estimated state, xHat).
3. The code below uses a *counter* to perform the measurement update every 10th iteration (i.e., 10 Hz sensor with a 100 Hz filter).
4. Copy the code and run the simulation. Make sure the ToF sensor is pointing downwards:

```
function xHat = myKalman_CV(tofData)
%#codegen
% Piecewise Constant Velocity model => x=[v; x], random acceleration.
% "tofData" is 4x1: [distance(m), ambient, signal, status].
% Output xHat = estimated position (meters).

persistent x P dt counter sigma_a
if isempty(x)
    % Initialize state & cov
    x = [0; 0.05];          % e.g., v=0, x=0.05
    P = 1e-3 * eye(2);

    dt = 0.01;              % filter runs at 100 Hz
    counter = 0;

    % Tune the std dev of acceleration
    sigma_a = 0.1;          % (example) m/s^2
end

% F matrix
F = [1, 0;
     dt, 1];

% Piecewise white-noise acceleration => Q = Gamma * sigma_a^2 * Gamma'
Gamma = [dt; 0.5*dt^2];
Q = sigma_a^2 * (Gamma * Gamma');

% Measurement matrix, sensor measures x only
H = [0 1];
R = (0.005)^2;   % e.g. 5 mm stdev

% 1) Predict
xPred = F*x;
PPred = F*P*F' + Q;

% 2) Measurement update ~10 Hz => only every 10 steps
counter = counter + 1;
if counter >= 10
```

```
    z = tofData(1);
    S = H*PPred*H' + R;     % scalar
    K = PPred*H'/S;         % 2x1
    x = xPred + K*(z - H*xPred);
    P = (eye(2) - K*H)*PPred;
    counter = 0;
else
    x = xPred;
    P = PPred;
end

% Output the bottom element (position)
xHat = x;
```

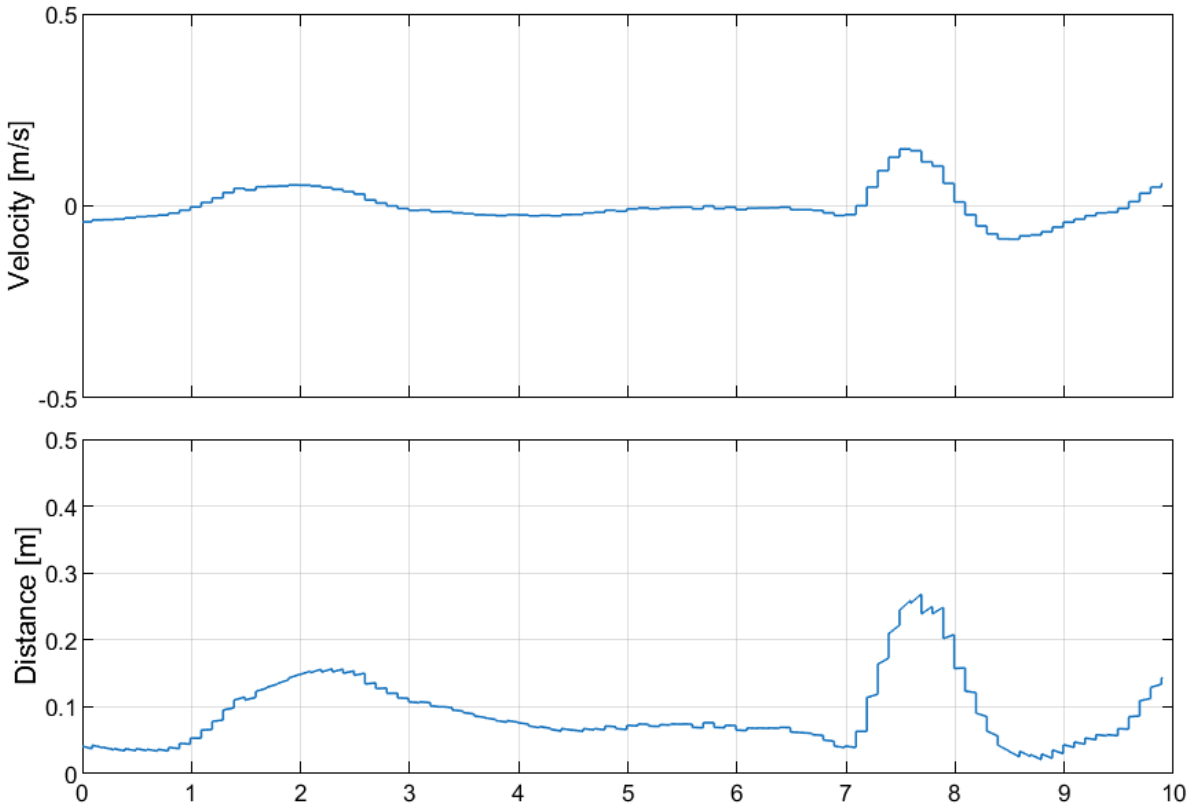


Figure 3: Example scope output of the constant acceleration Kalman Filter.

TASK 3: CONSTANT-ACCELERATION KALMAN FILTER

Objective

Switch to a constant acceleration model to handle faster motions.

System Model

1. **State vector:**

$$\boldsymbol{X} = \begin{bmatrix} \dddot{x} \\ \ddot{x} \\ x \end{bmatrix}$$

2. **Process model** (assuming constant acceleration):

In matrix form (with Δt=0.0, because we run at 100 Hz) the state space model is:

$$\begin{bmatrix} \ddot{x}_{k+1} \\ \dot{x}_{k+1} \\ x_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \Delta t & 1 & 0 \\ \frac{1}{2}\Delta t^2 & \Delta t & 1 \end{bmatrix} \begin{bmatrix} \ddot{x}_k \\ \dot{x}_k \\ x_k \end{bmatrix} + \boldsymbol{w_k}$$

Where $w_k \sim N(0, \sigma_a)$

3. **Measurement model (**only measuring x, i.e. height**)**:

$$y_k = [0 \quad 0 \quad 1] \begin{bmatrix} \ddot{x}_k \\ \dot{x}_k \\ x_k \end{bmatrix} + v_k$$

4. **Noise**

- with $R = (0.003)^2$ (since the ToF stdev $\sigma \approx 0.003$ m) – assume all sensors are the same.
- Choose a modest $Q = 1$ to allow large variability in the constant-acceleration assumption.

Instructions

1. **In the myKalman block, switch your state initialization to `[ddot{x}; dot{x}; x]`. For instance,** `[0; 0; 0.05]` (acceleration=0, velocity=0, height=0.05 m).
2. **Adjust $\boldsymbol{F}$, $\boldsymbol{H}$, $\boldsymbol{Q}$, etc. for the 3x3**
3. Retain the logic of only updating the measurement every 10 steps.
4. Copy the following code and study its operation:

```
function xHat = myKalman_CA(tofData)
%#codegen
% Piecewise Constant Acceleration => x=[a; v; x], random JERK.

persistent x P dt counter sigma_j
if isempty(x)
    % e.g. initial guess: a=0, v=0, x=0.05
    x = [0; 0; 0.05];
    P = 1e-3*eye(3);

    dt = 0.01;        % 100 Hz
    counter = 0;

    % Tune jerk std dev
    sigma_j = 0.1;    % (example) m/s^3
end
```

```
% F
F = [1,   0,     0;
     dt,  1,     0;
     0.5*dt^2, dt, 1];

% Gamma and Q
Gamma = [1; dt; 0.5*dt^2];
Q = sigma_j^2 * (Gamma * Gamma');

% Measurement: we measure x => H=[0 0 1]
H = [0 0 1];
R = (0.005)^2;

% Predict
xPred = F*x;
PPred = F*P*F' + Q;

% Update every 10 steps to emulate 10 Hz
counter = counter + 1;
if counter >= 10
    z = tofData(1);    % distance
    S = H*PPred*H' + R;  % scalar
    K = PPred*H'/S;      % 3x1
    x = xPred + K*(z - H*xPred);
    P = (eye(3)-K*H)*PPred;
    counter = 0;
else
    x = xPred;
    P = PPred;
end

xHat = x;  % the bottom element is position
```

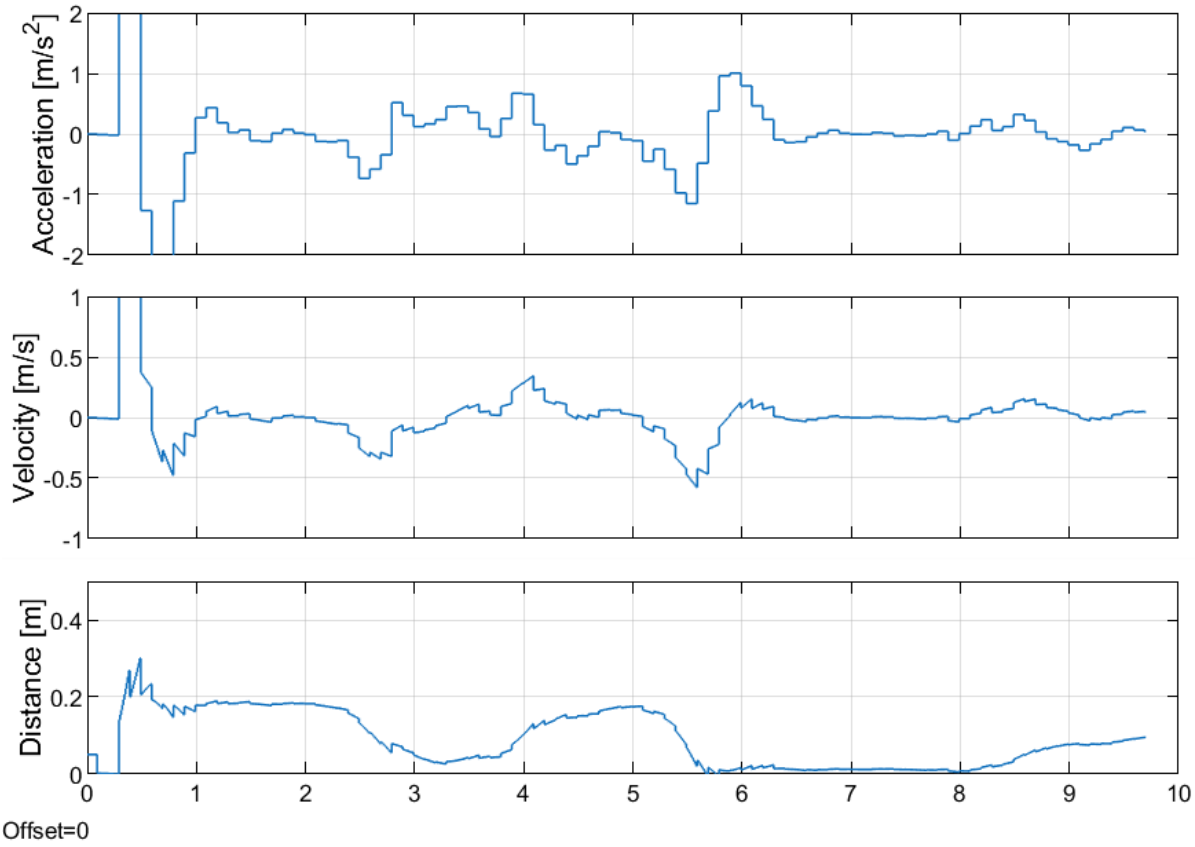


Figure 4: Example output using the Constant Acceleration Model.

TASK 4: FUSE ALL THREE SENSORS

Finally, if you want to use **all three** ToF measurements:

- Let $z = [z_{left} \quad z_{centre} \quad z_{right}]^T$
- Since each sensor has the same measurement state:

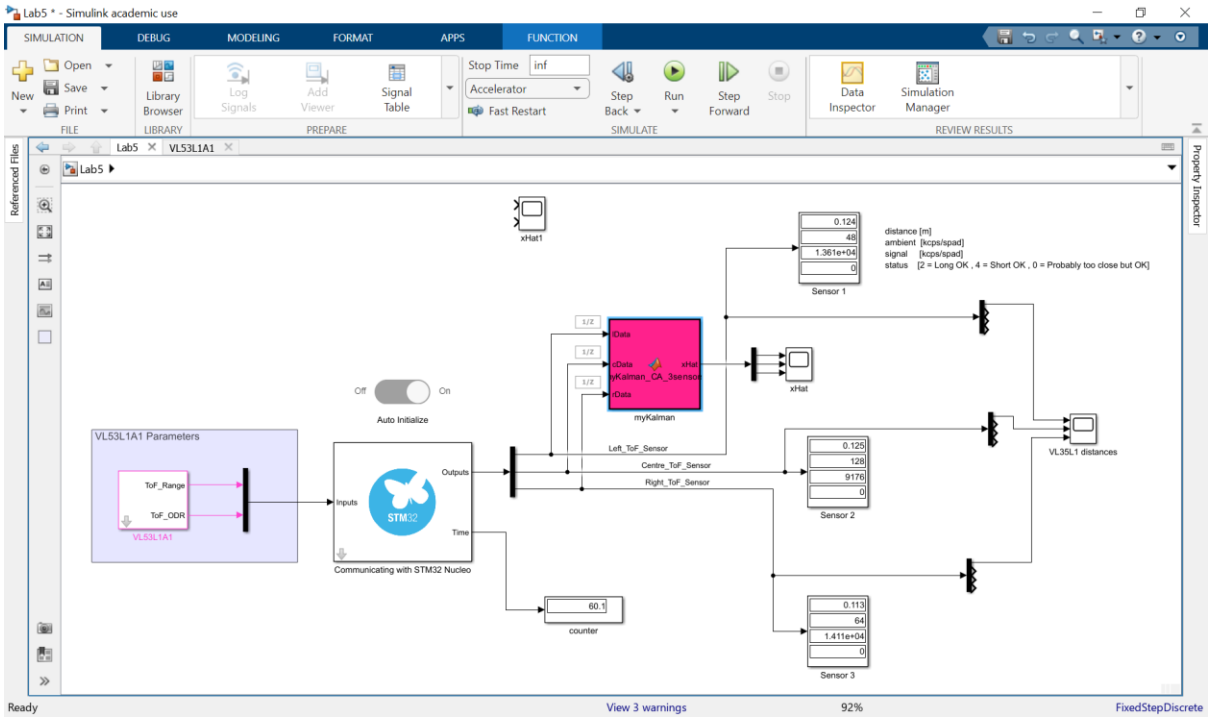
$$H = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

- $\boldsymbol{R}$ is now a 3x3 diagonal matrix if the noise covariances are the same ie. σ^2 on the diagonals.
-
- Copy the following code:

```
function xHat = myKalman_CA_3sensors(lData, cData, rData)
%#codegen
% 3-sensor fusion, x=[a; v; x], piecewise-constant jerk.

persistent x P dt counter sigma_j
if isempty(x)
    x = [0; 0; 0.05];
    P = 1e-3*eye(3);
```

```
    dt = 0.01;
    counter = 0;
    sigma_j = 1.0;    % tune jerk stdev
end

% State transition
F = [1,   0,       0;
     dt,  1,       0;
     0.5*dt^2, dt, 1];

% Q for piecewise-constant jerk
Gamma = [dt; 0.5*dt^2; (1/6)*dt^3];
Q = sigma_j^2*(Gamma*Gamma');

% Build the measurement z (3x1)
z = [lData(1); cData(1); rData(1)];

% H is 3x3, each row picks out x from [a; v; x]
H = [0 0 1;
     0 0 1;
     0 0 1];
R = (0.005)^2 * eye(3);

% Predict
xPred = F*x;
PPred = F*P*F' + Q;

% Update every 10 steps
counter = counter + 1;
if counter >= 10
    S = H*PPred*H' + R;      % 3x3
    K = PPred*H'/S;          % 3x3
    x = xPred + K*(z - H*xPred);
    P = (eye(3)-K*H)*PPred;
    counter = 0;
else
    x = xPred;
    P = PPred;
end

xHat = x;   % state vector estimate
```

- Connect the other sensors accordingly, it should look like this:



CONCLUSION

1. **Piecewise White Noise**: We treat the unmodeled derivative (acceleration in CV, jerk in CA) as a *constant* random variable per time step.
2. **Tune** σ_a (for CV) or σ_j (for CA) carefully. Larger values let the filter "trust" the model less and adapt faster to big changes. Too large => overfitting to sensor data.
3. **Measurement Covariance**: $\sigma_{sensor} = 0.005$ for the ToF sensors. This may be affected by signal to ambient light ratio and different surfaces. Remember this for the project 😊
4. **Initialization**: Choose reasonable starting states (e.g. near 0 for velocity/acceleration, a plausible starting height for position).
5. **Every 10 Steps**: Because your sensors sample at 10 Hz and the filter runs at 100 Hz, do the measurement update only at steps counter >= 10.

That's it! You now have **Tasks (2), (3), and (4)** in your lab.

The end! 😊