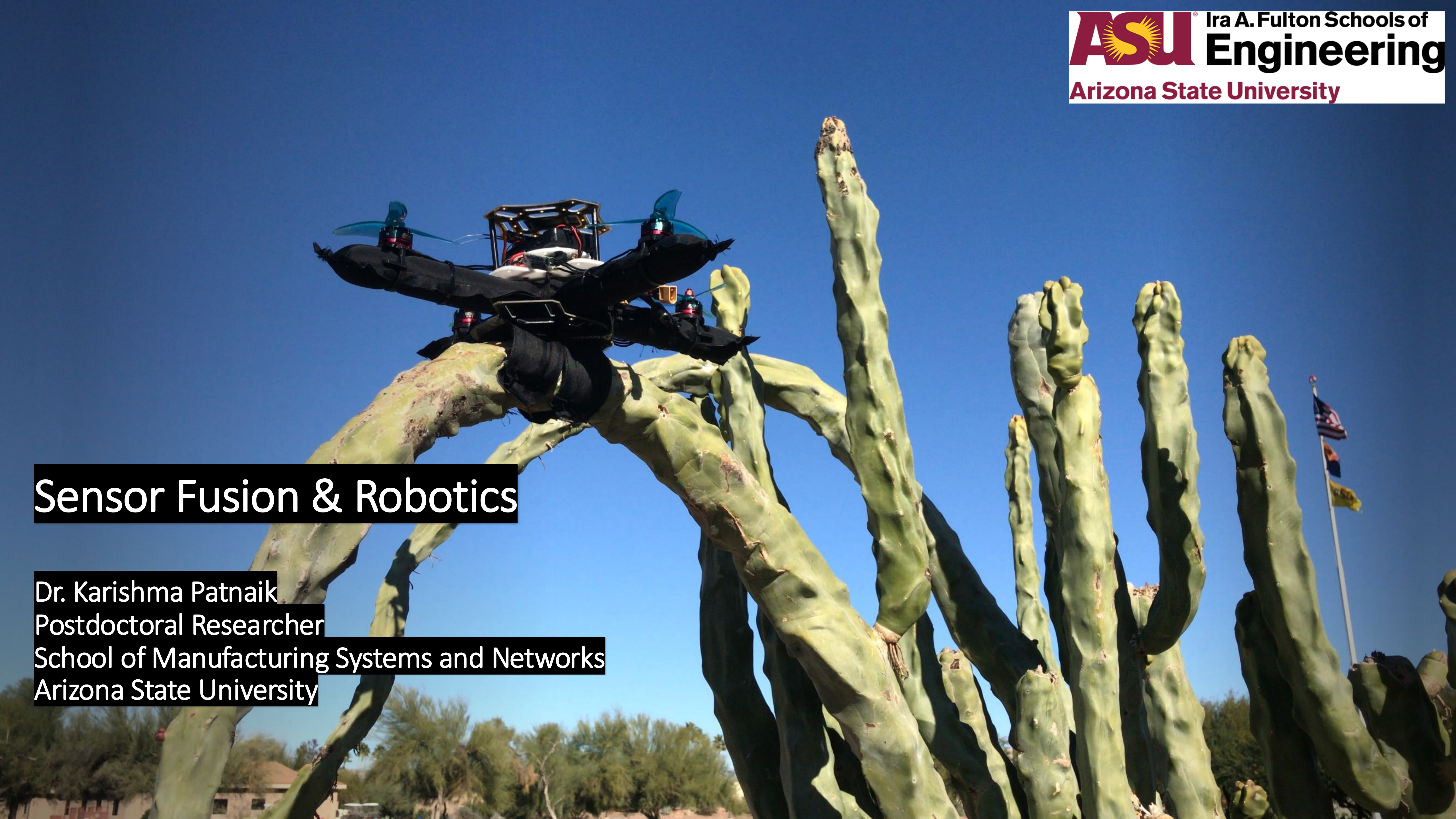


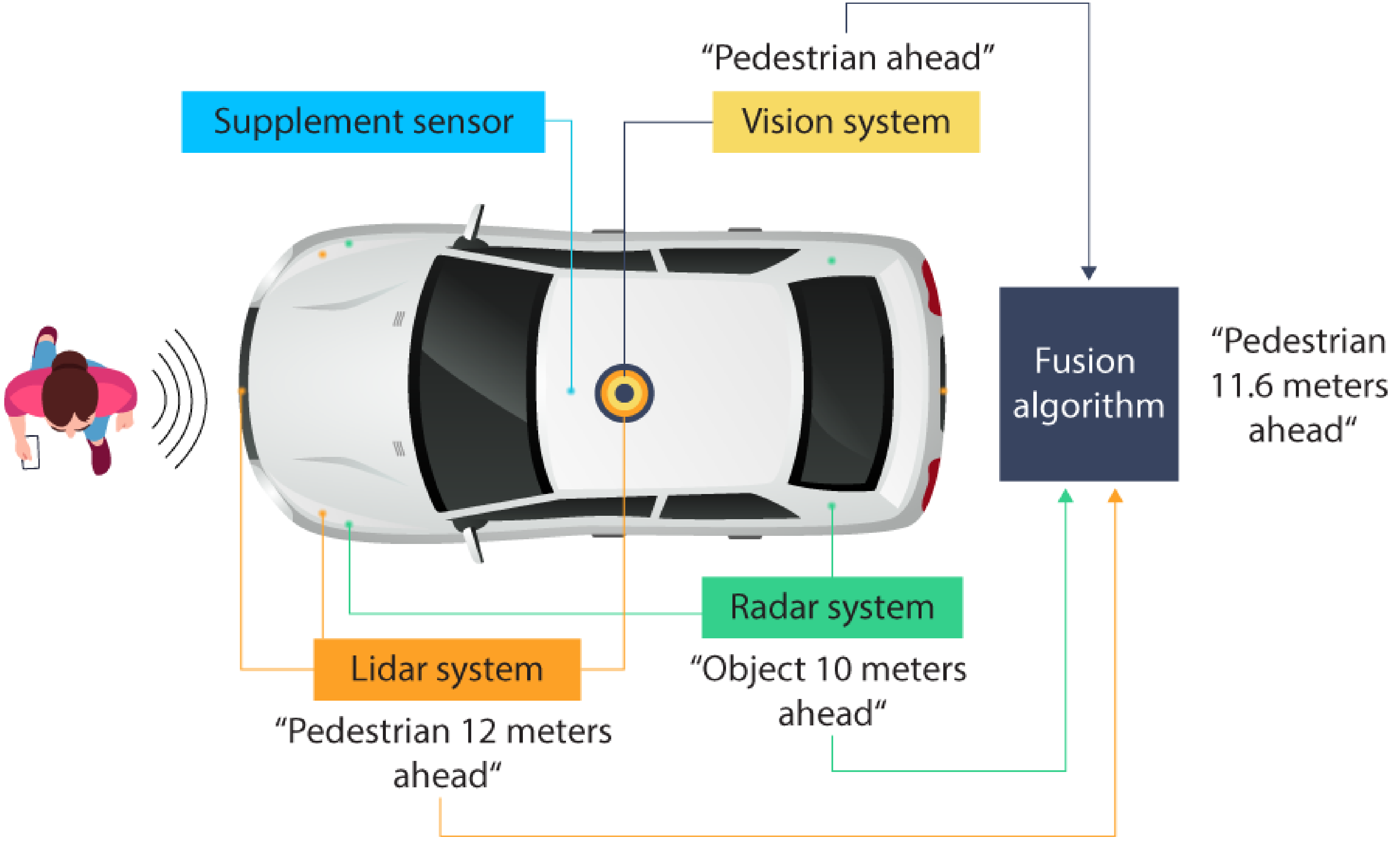
Sensor Fusion & Robotics

Dr. Karishma Patnaik
Postdoctoral Researcher
School of Manufacturing Systems and Networks
Arizona State University

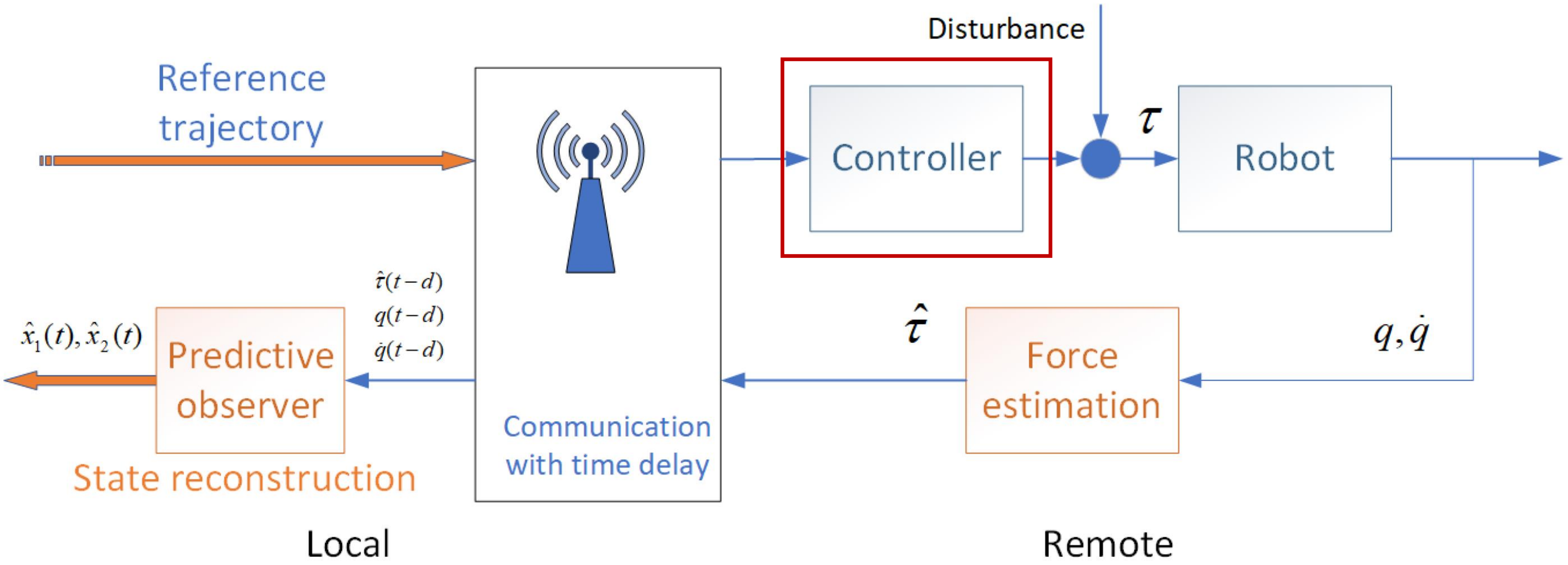
Ira A. Fulton Schools of Engineering
Arizona State University

What is Sensor Fusion?

- Process of combining sensor data or data derived from disparate sources
- For e.g. cameras and lidars have both pros and cons
- With sensor fusion, resulting information has less uncertainty



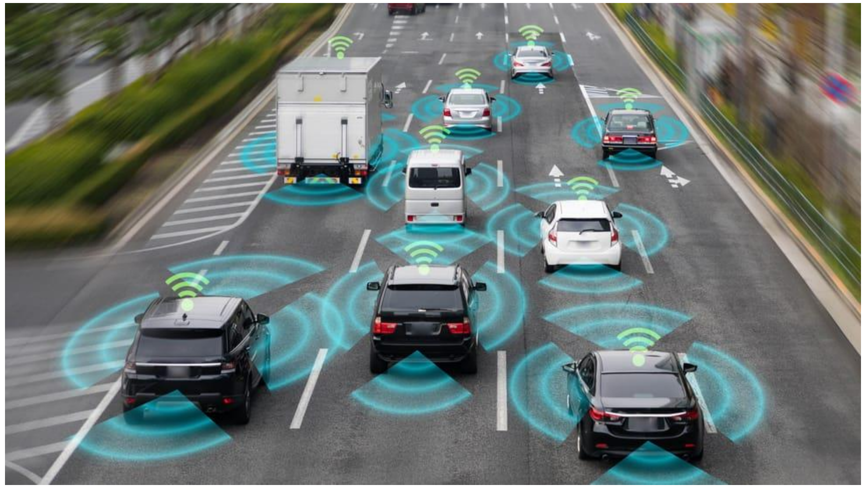
Sensor Fusion in Autonomy – Crucial for Control



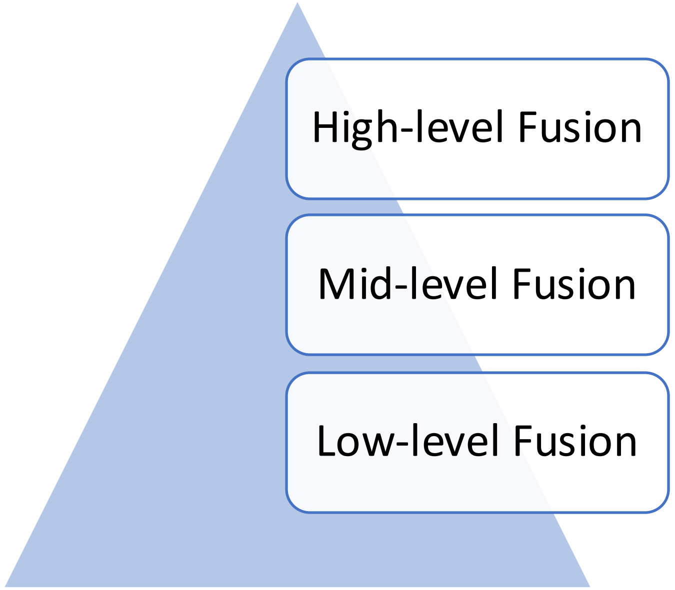
Applications

- **Autonomous Vehicles**: Combining data from cameras, LiDAR, and radar to accurately perceive obstacles and navigate safely.
- **Robotics**: Using a combination of cameras, ultrasonic sensors, and inertial measurement units (IMUs) to navigate in complex environments and avoid collisions.
- **Wearable Devices**: Combining accelerometer, gyroscope, and GPS data to accurately track movement and location.
- **Industrial Monitoring**: Using multiple sensors to monitor equipment health and predict potential failures.

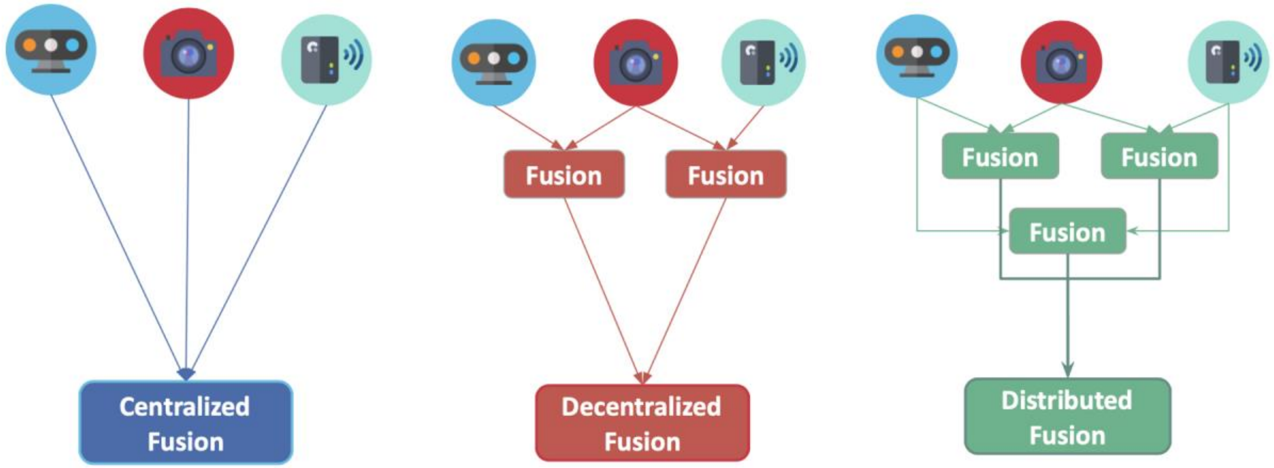





Types of Sensor Fusion



Based on the level of autonomy –
When?

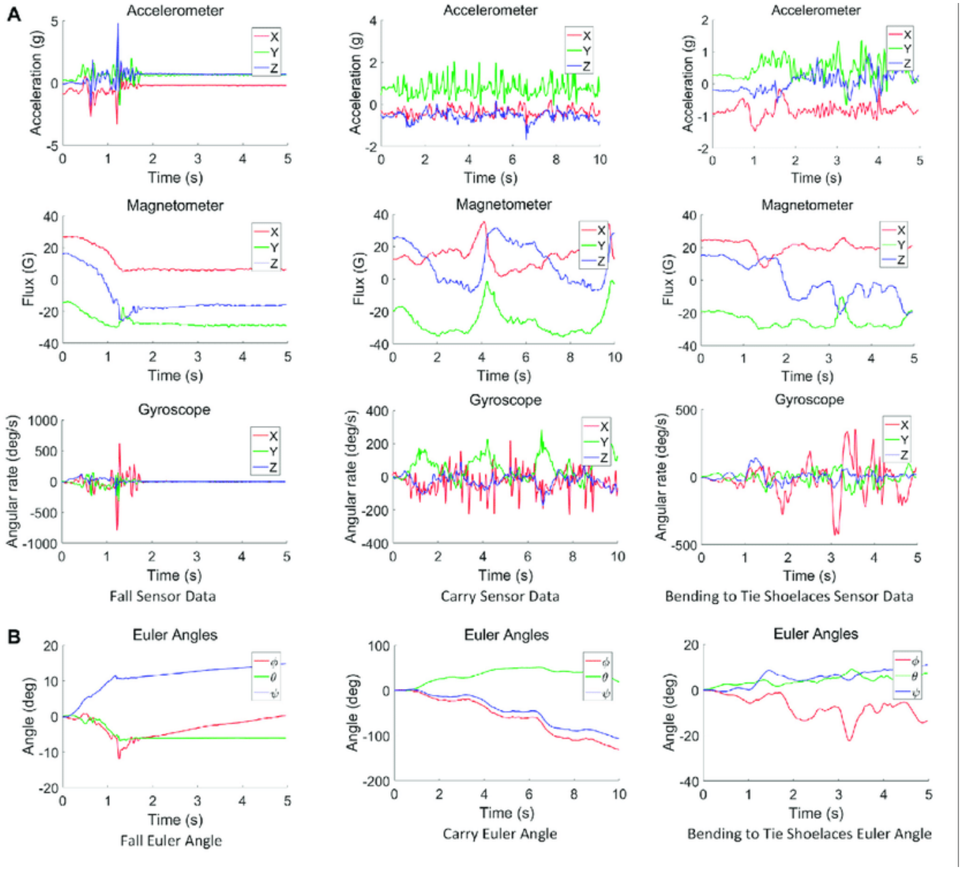


Other types based on Algorithm –
Where?

Types of Sensor Fusion – Low-level Example 1

Low-Level Fusion (Data-Level Fusion)

- Raw data from multiple sensors are combined before any processing.
- Happens early in the pipeline
- Ex: Merging accelerometer, magnetometers & gyroscope in (IMUs).
    - Accelerometer:
        - Linear accelerations
        - Effected by gravity
    - Magnetometer
        - Accurate yaw
        - Takes measurements over long time to converge
    - Gyro
        - Gives angular velocities
        - Drifts over time due to integration under noise, bias



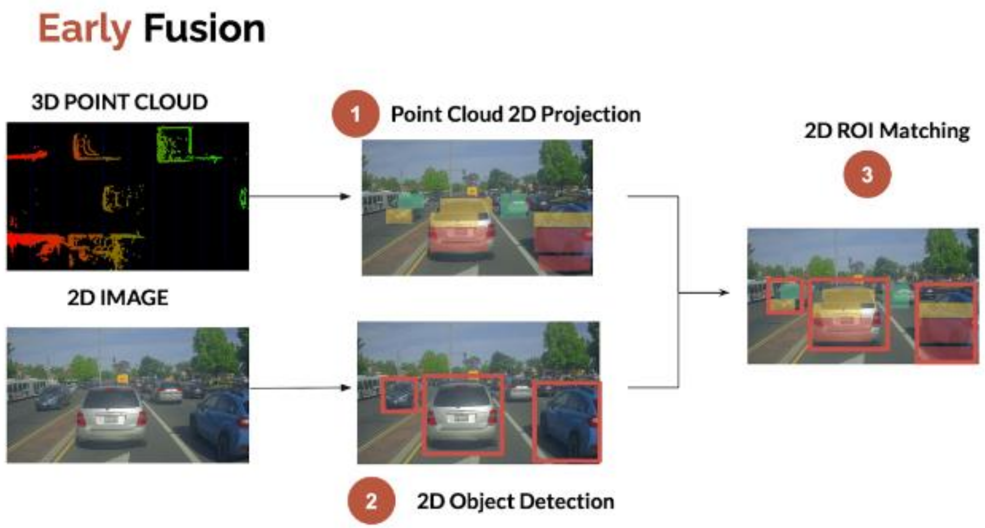
IMU-based Sensing [1]

[1] Yu, Zheqi, et al. "IMU sensing–based Hopfield neuromorphic computing for human activity recognition." *Frontiers in Communications and Networks* 2 (2022): 820248.

Types of Sensor Fusion – Low-level Example 2

Combining LiDAR point clouds and camera pixel data for enhanced perception

- Project the 3D PCL into 2D projection – using geometric principles
- Use YOLO to detect objects from the camera image
- Region Of Interest **(ROI) Identification**
    - For each bounding box, the camera gives us the classification
    - For each LiDAR projected point, we have a very accurate distance.
    - Can use clustering / thresholding
- Feature Extractions – Extract edge, color information,
- Data Fusion – Combine using feature concatenation

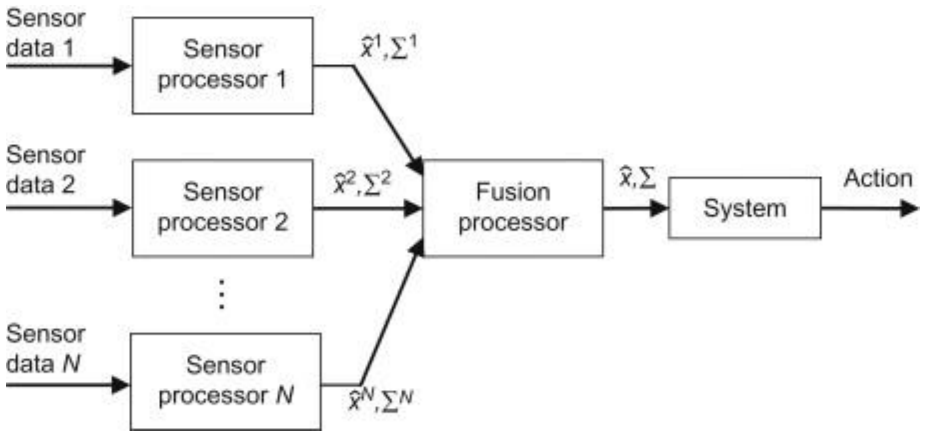


Fusing PCL data with camera image [2]

[2] LiDAR and Camera Sensor Fusion in Self-Driving Cars https://www.thinkautonomous.ai/blog/lidar-and-camera-sensor-fusion-in-self-driving-cars/

Other Low-Level Fusion Techniques

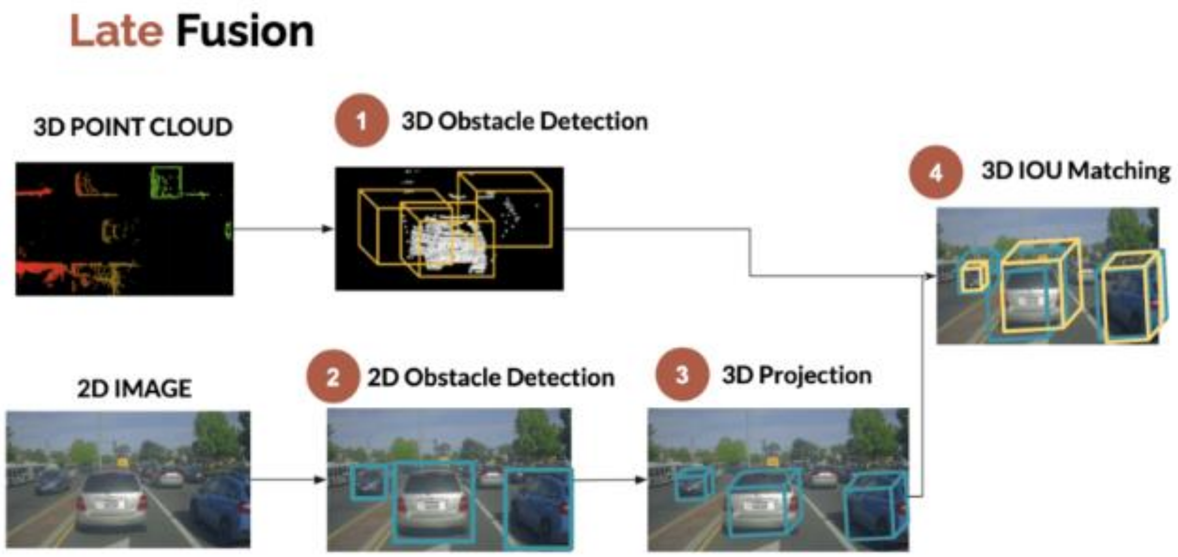
- **Weighted Averaging:**
  - Combines raw data by assigning weights based on sensor confidence.
  - Example: Fusing temperature readings from multiple sensors.
- **Bayesian Networks:**
  - Probabilistic models that combine raw data with prior knowledge.
  - Example: Fusing radar and LiDAR point clouds for obstacle detection.
- **Direct Concatenation**:
  - Combines raw data vectors directly (e.g., concatenating LiDAR point clouds with camera RGB data).
- **Wavelet Transform**:
  - Used for combining sensor signals by decomposing them into frequency bands.
  - Example: Fusing audio signals with vibration sensor data.



Types of Sensor Fusion – Mid-level

Mid-Level Fusion (Feature-Level Fusion)

- Extracted features from each sensor (e.g., **edges in an image or clusters in LiDAR data**) are combined.
- Fusion occurs at the level of processed data representations.
- Examples:
  - Combining object detections from cameras and depth information from LiDAR to **classify and localize objects**.
  - Using radar speed readings and image-based tracking for better object trajectory prediction.
- Pros:
  - Reduces data size compared to low-level fusion.
  - Easier to handle than raw data.
- Cons:
  - Loss of detailed information.
  - Requires effective feature extraction techniques.



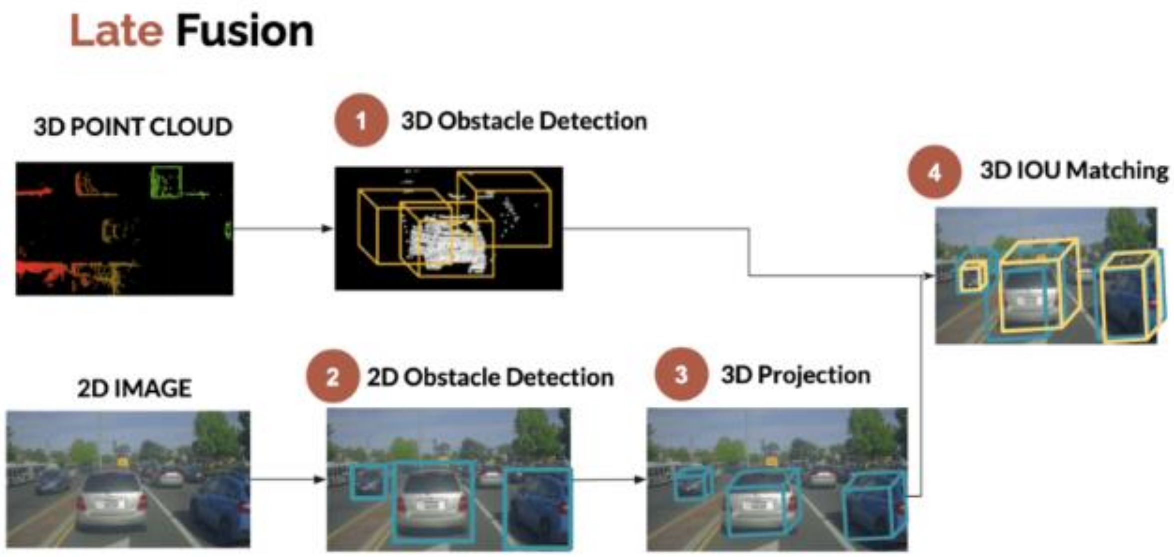
Fusing PCL data with camera image [2]

[2] LiDAR and Camera Sensor Fusion in Self-Driving Cars https://www.thinkautonomous.ai/blog/lidar-and-camera-sensor-fusion-in-self-driving-cars/

Types of Sensor Fusion – Example Mid-level

Combining 3D Bounding boxes from LiDAR point clouds and camera pixel data

- 3D bounding boxes in LIDAR
  - Unsupervised 3D machine learning
  - Deep learning
- 3D obstacle detection via camera
  - Camera calibration
  - Depth map
- Intersection over Union (IoU) Matching
  - If the bounding boxes from camera and LiDAR overlap, in 2D or 3D, we consider that obstacle to be the same

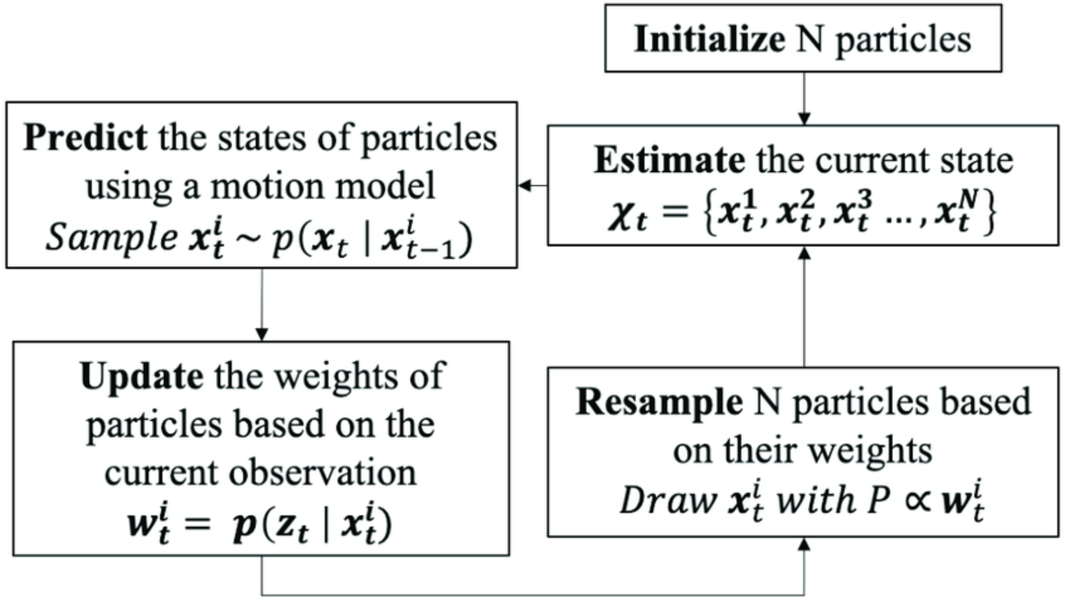


Fusing PCL data with camera image [2]

[2] LiDAR and Camera Sensor Fusion in Self-Driving Cars https://www.thinkautonomous.ai/blog/lidar-and-camera-sensor-fusion-in-self-driving-cars/

Other Mid-Level Fusion Techniques

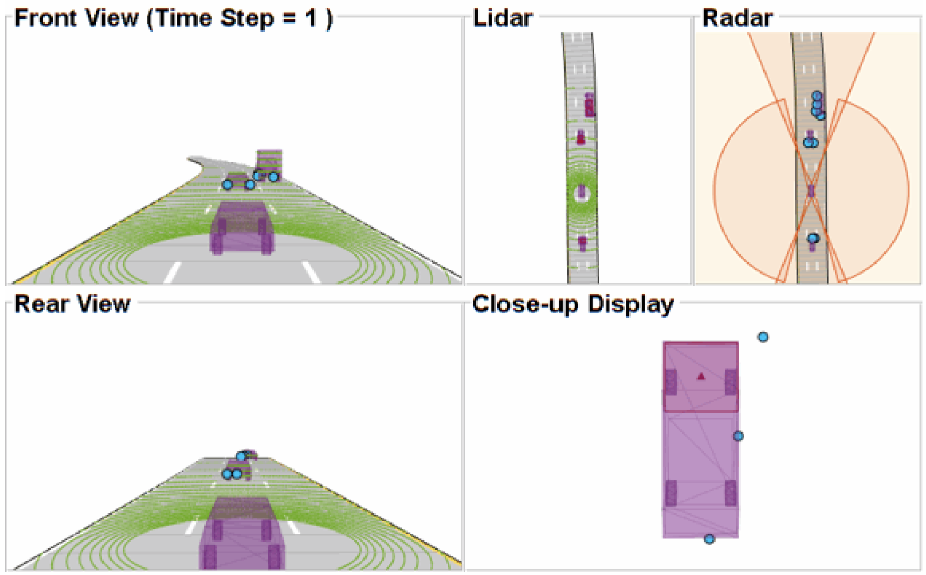
- Kalman Filters:
  - Recursive estimation for fusing features like position & velocity
- Particle filters:
  - Non-linear non-Gaussian approach for tracking and estimation
  - Example: fusing radar and visual features for object tracking
- Principal Component Analysis:
  - Reduces dimensionality of fused features while preserving variance
  - Example: Fusing multiple-camera views for 3D reconstruction
- Neural Networks:
  - CNNs for images, LSTMs for temporal features
  - Example: fusing image and LIDAR features for object detection

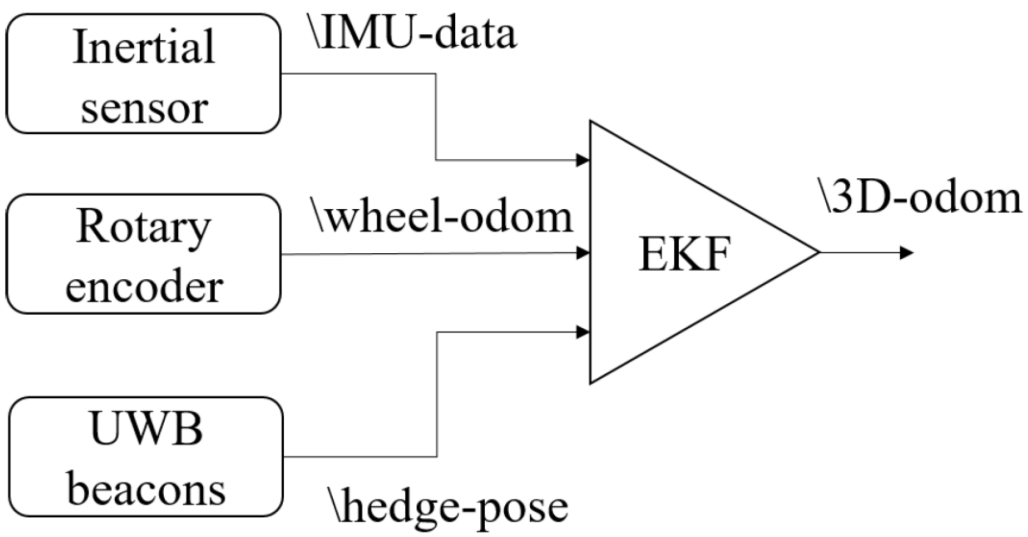


Sensors: Measurement, Process and Noise

Goal: You are a robot

1. Need to know your own position, velocity etc. – state (6D pose)
2. Need to know your surroundings, e.g other cars state





Position, velocity and uncertainty!

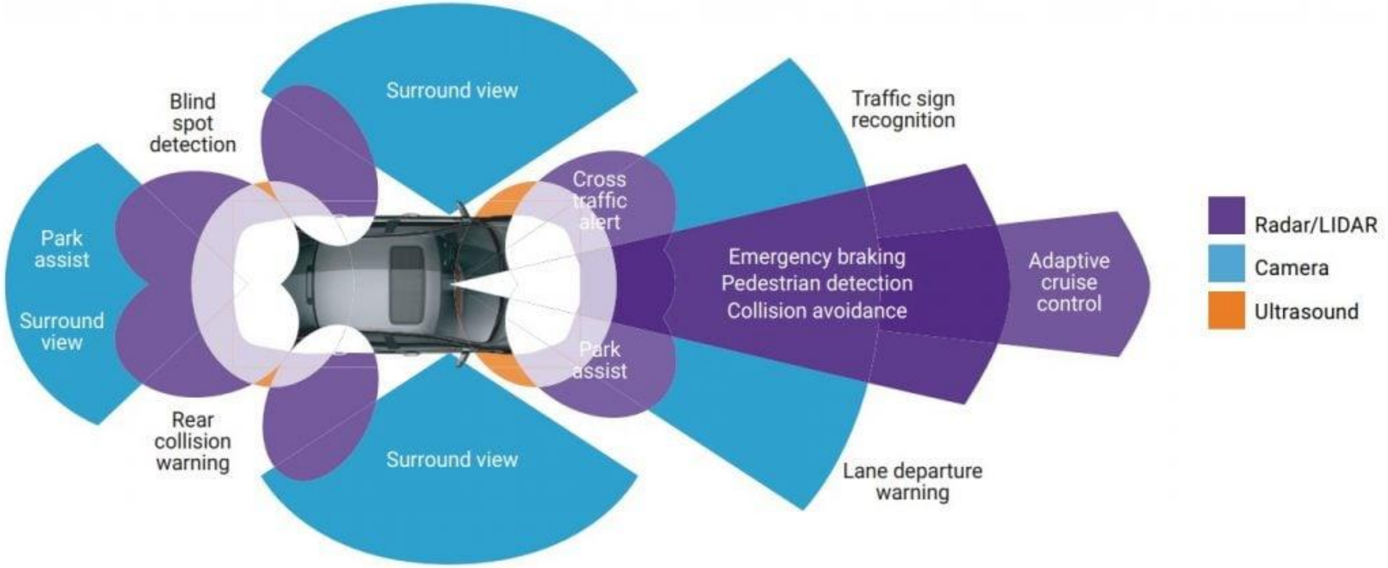
Sensors: Measurement, Process and Noise

Have a sensor :

1. Raw data is available, or some kind of filter has been applied
2. You want accuracy

Have multiple sensors:

1. One sensor gives one kind of information only for example, radar give speed
2. Want additional information fused to get a full estimate

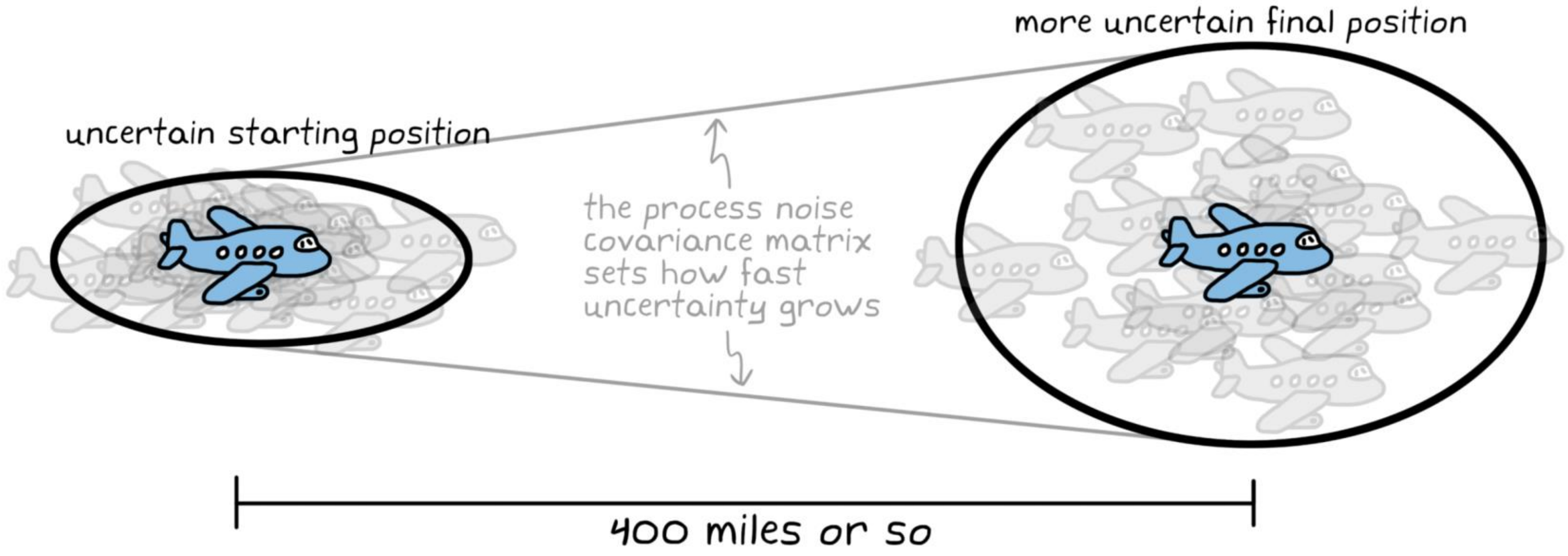


Sensors: Measurement, Process and Noise

Have problems:

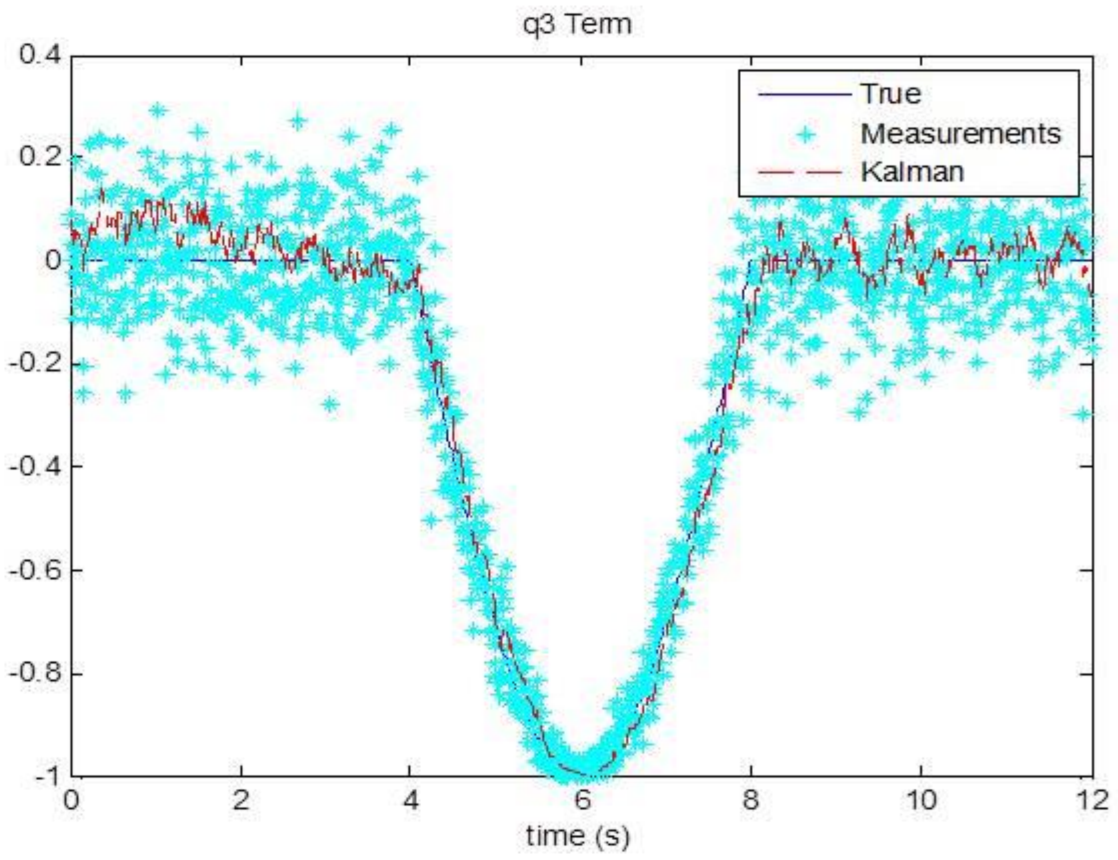
1. Less information
2. Sensor inefficient

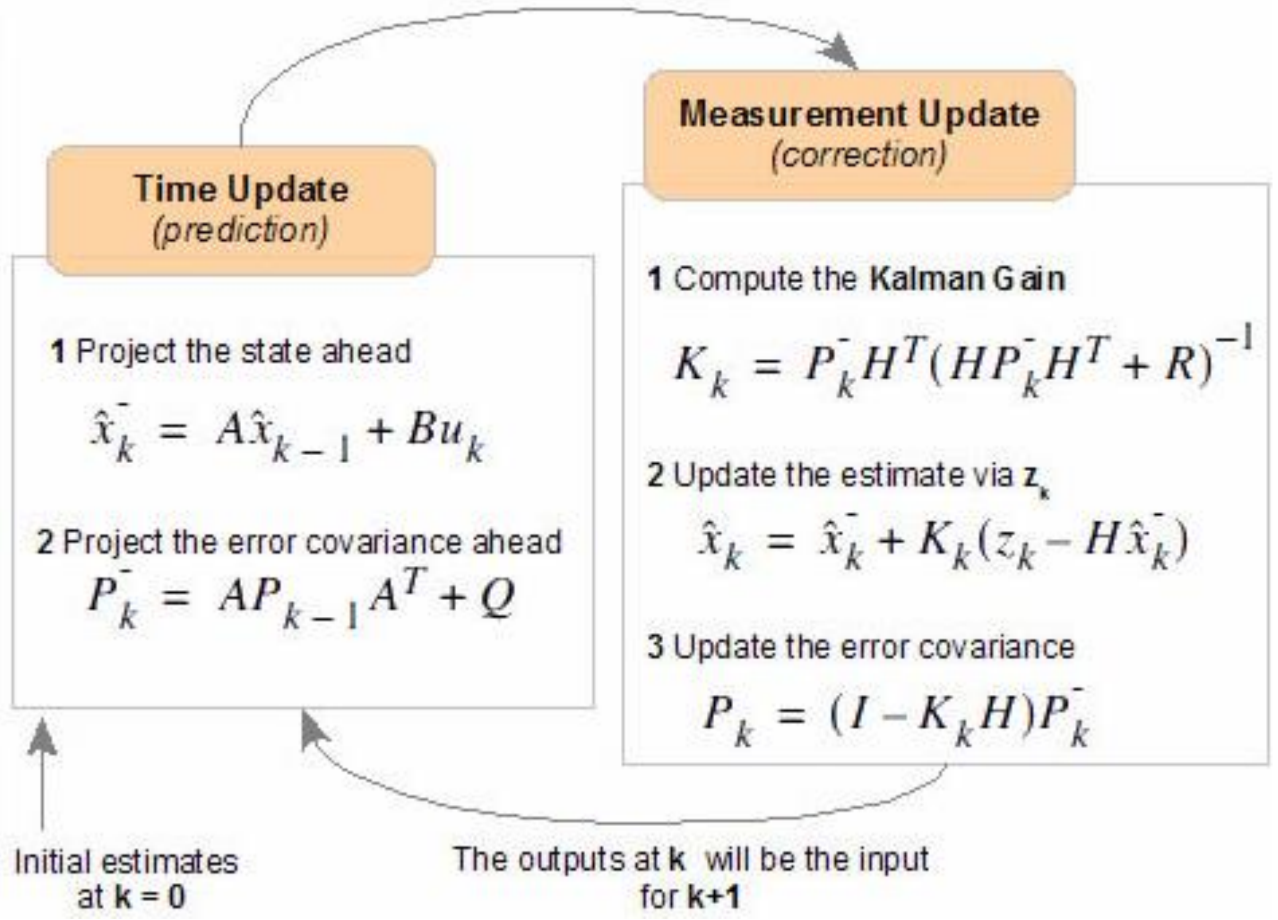
That's why Kalman Filters!



Position, velocity and uncertainty!

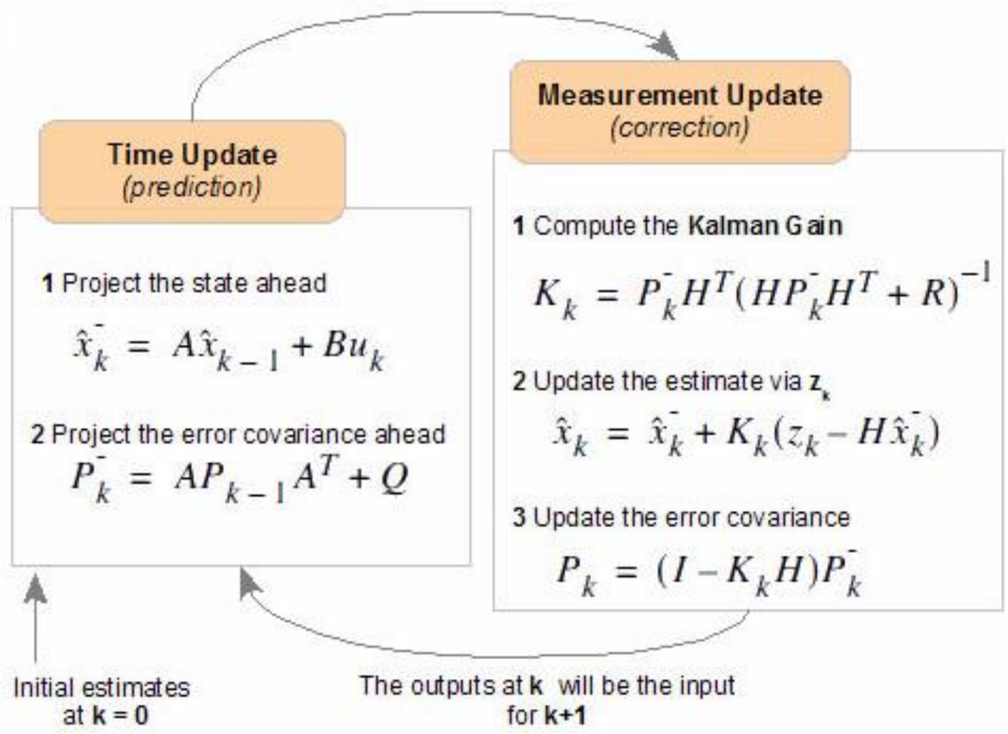
Discussion: Kalman Filters

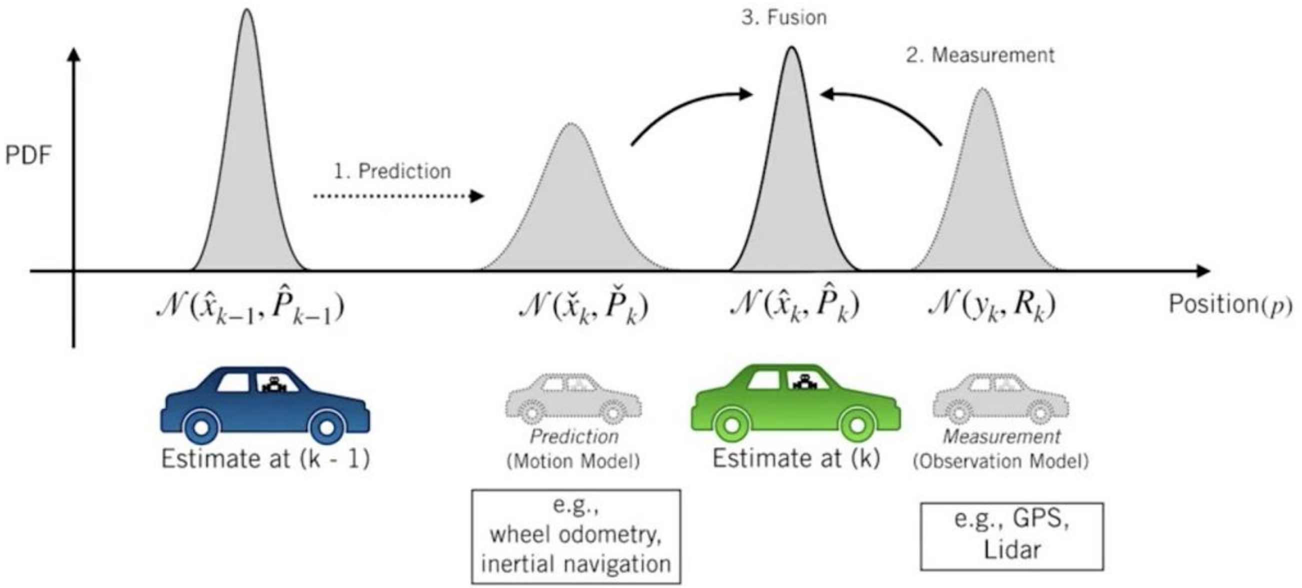




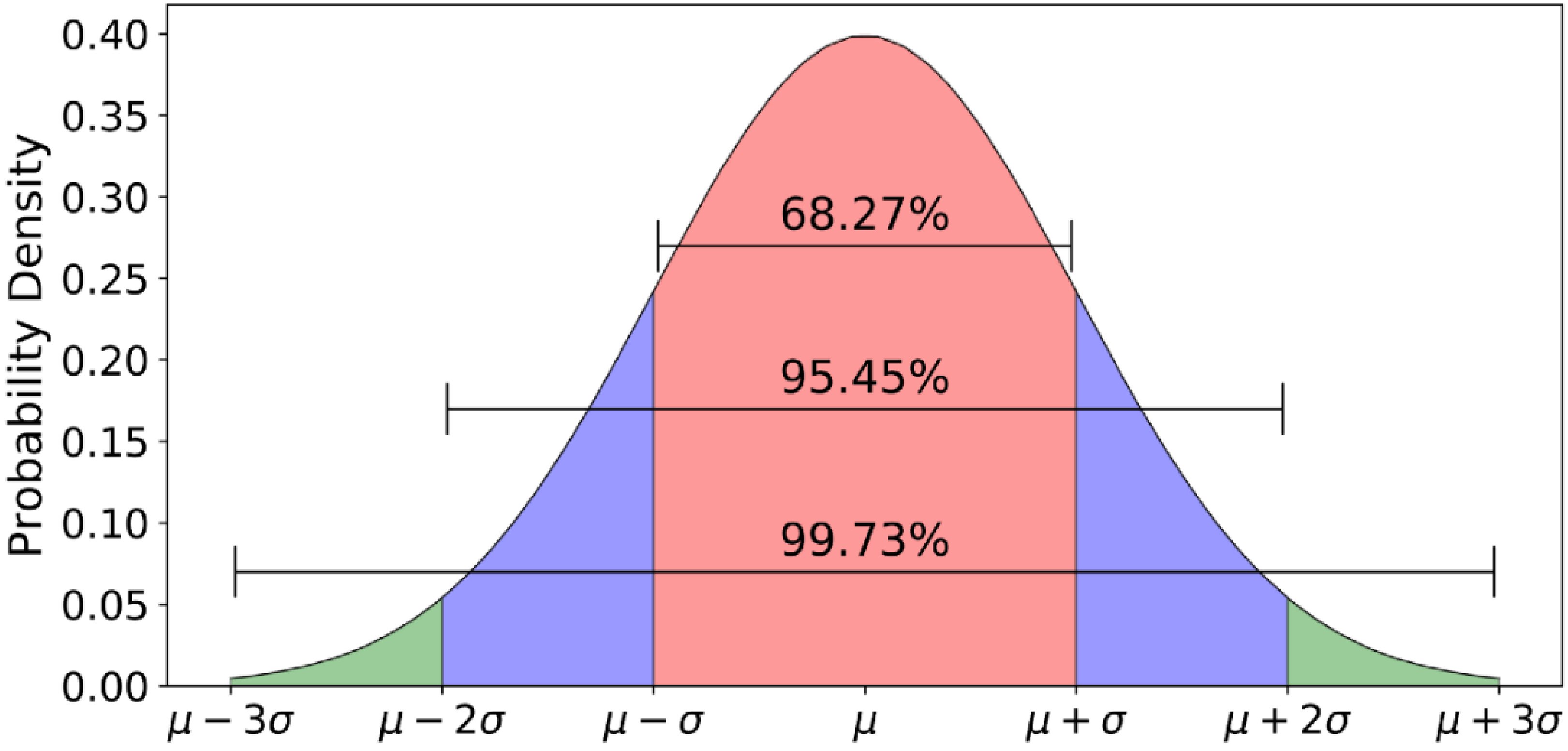
Discussion: Kalman Filters

Ira A. Fulton Schools of Engineering
Arizona State University

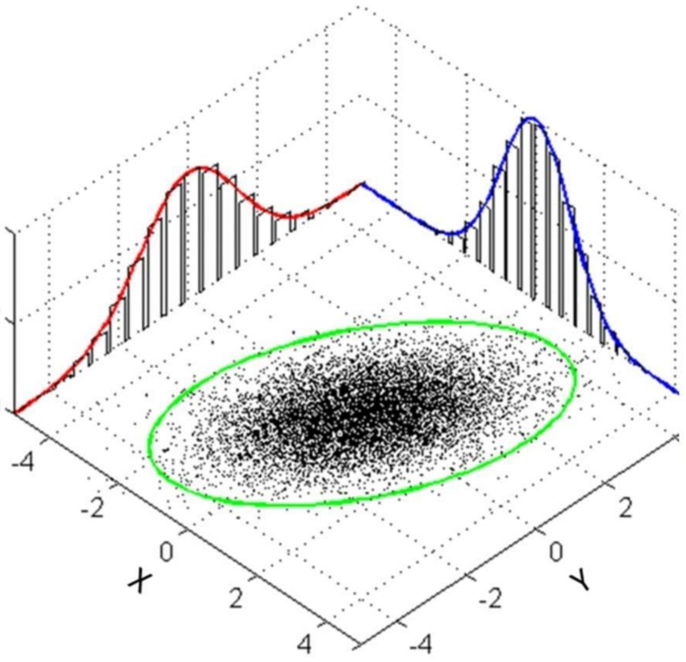


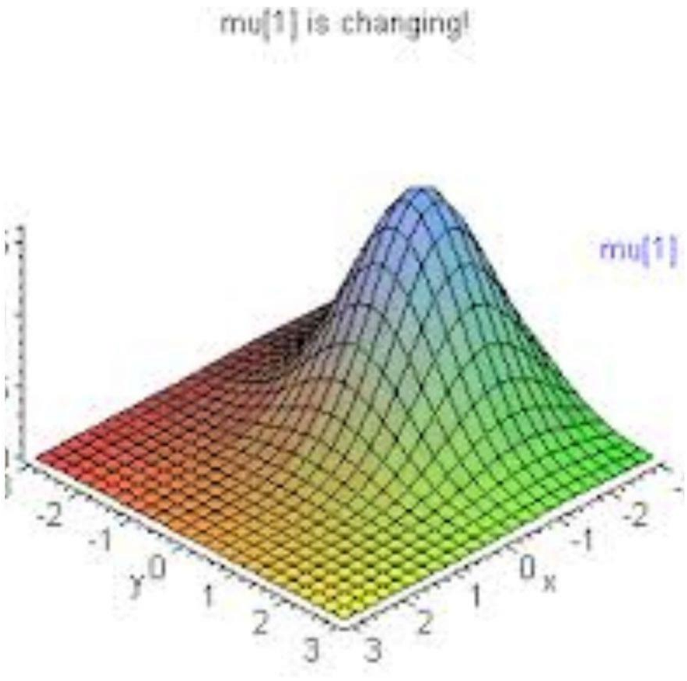


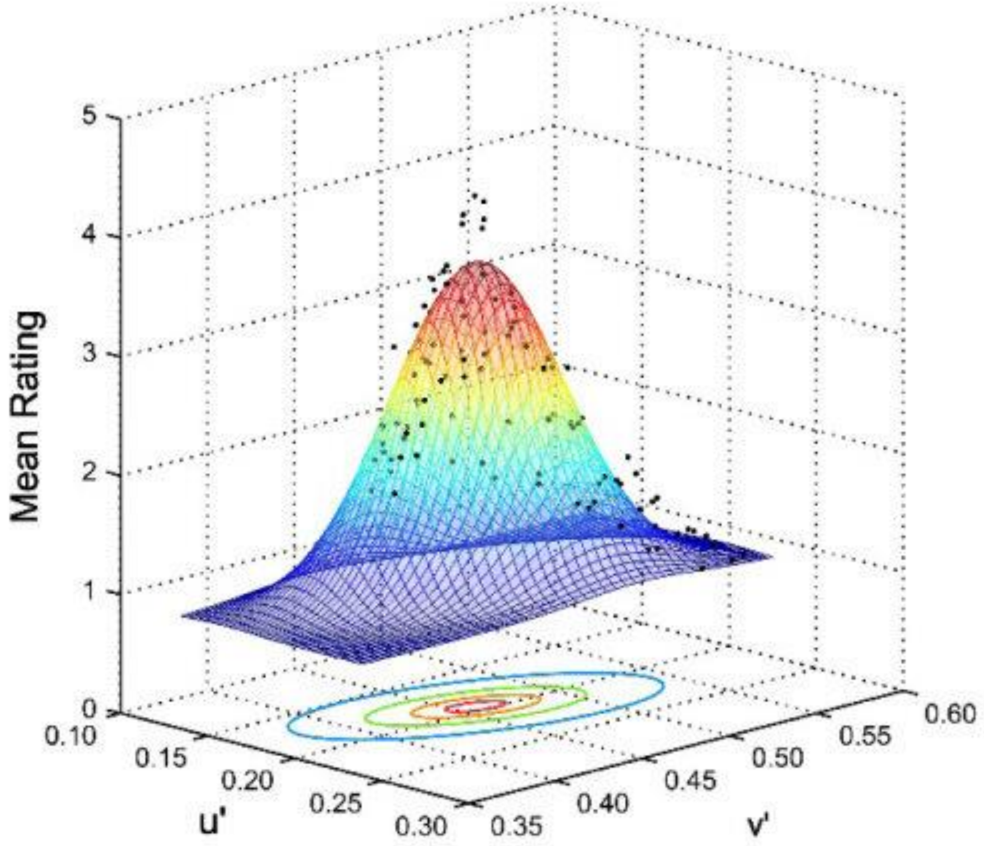
Discussion: Kalman Filters (Gaussian Noise)



Discussion: Kalman Filters







KFs: Measurement, Process and Noise

- **Linear system model**: The system dynamics (how the state evolves over time) and the measurement model (how the state is observed) must be represented by linear equations with matrices F (state transition), H (observation), and B (control input).

- **Gaussian noise assumptions**: Both the process noise (system uncertainty) and measurement noise (sensor uncertainty) must be assumed to be white Gaussian noise with known covariance matrices (Q and R respectively).

- **Known initial conditions**: The initial state estimate ($\hat{x}$) and its corresponding error covariance matrix (P) must be provided to start the filtering process.

KFs: Measurement, Process and Noise

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}u_k + \mathbf{w}_k$$

$\mathbf{x}_k = [x_k; \dot{x}_k]$: State vector at time k (position and velocity).

$\mathbf{A}$: State transition matrix.

$\mathbf{B}$: Control input matrix.

u_k: Control input (constant here).

$\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$: Process noise with covariance $\mathbf{Q}$.

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$$

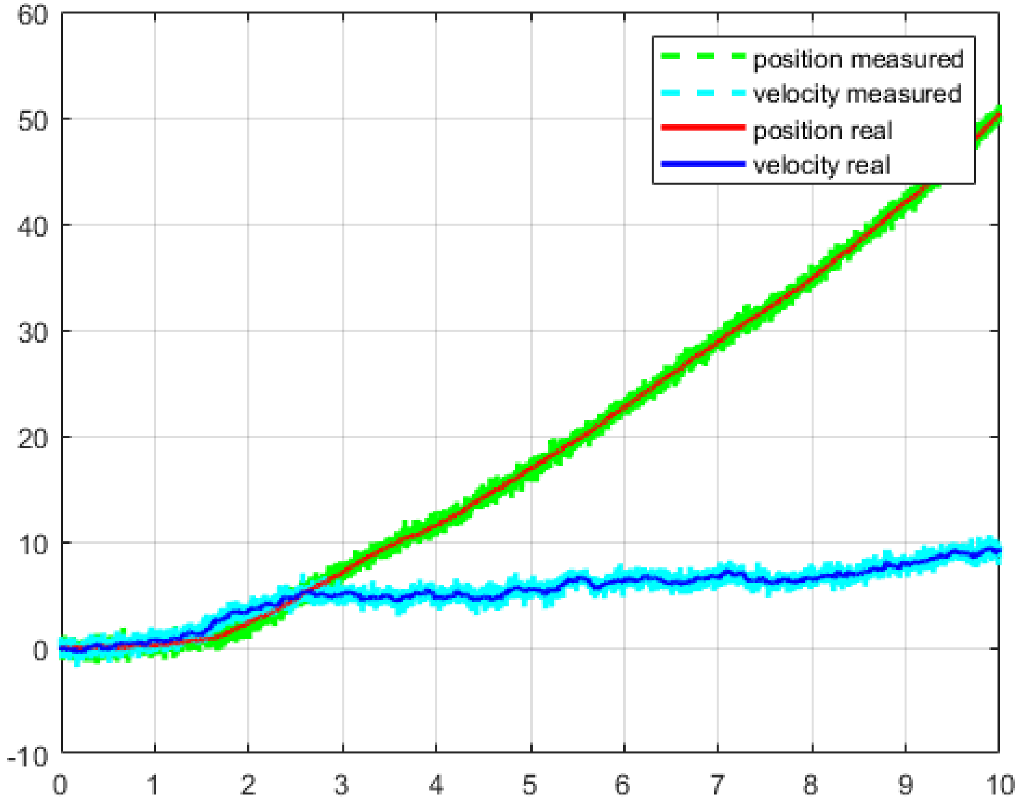
$\mathbf{z}_k = [z_{k,1}; z_{k,2}]$: Measurement vector (noisy position and velocity).

$\mathbf{H}$: Measurement matrix.

$\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$: Measurement noise with covariance $\mathbf{R}$.

```
% state equations, discrete system
muQ = 0;
Q = [0.01 0; 0 0.03]; % we need to know this: noise covariance of state
A = [1.0000 0.0010; 0 1.0000];
B = 1.0e-03 * [0.0005; 1.0000];
wk = normrnd(muQ,Q);

% output equations
H = [1 0; 0 1];
muR = 0;
R = [0.5 0;0 0.5]; % we need to know this: noise covariance of output
vk = normrnd(muR, R);

% simulation parameters:
t0 = 0;
dt = 0.001;
tend = 10;
x = [0 0]'; % initial state
z = x; % initial measurement;
len = length(t0:dt:tend);
xfs = zeros(len, length(x));
xfs(1,:) = x';
zfs(1,:) = z';
u = 1;

% apriori x
xhat_minus = [0 0]';
xhat = [0 0]';
xhatfs = xfs;
P = [ 0.1 0.2; 0.3 0.1];
```

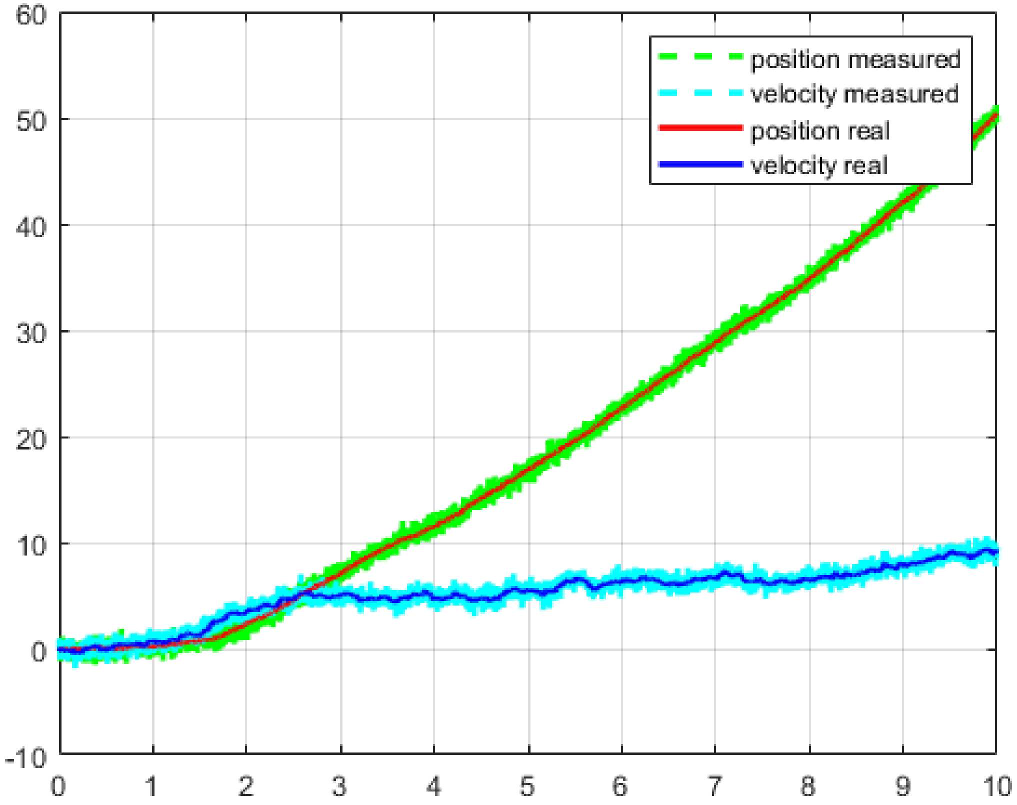
KFs: Measurement, Process and Noise

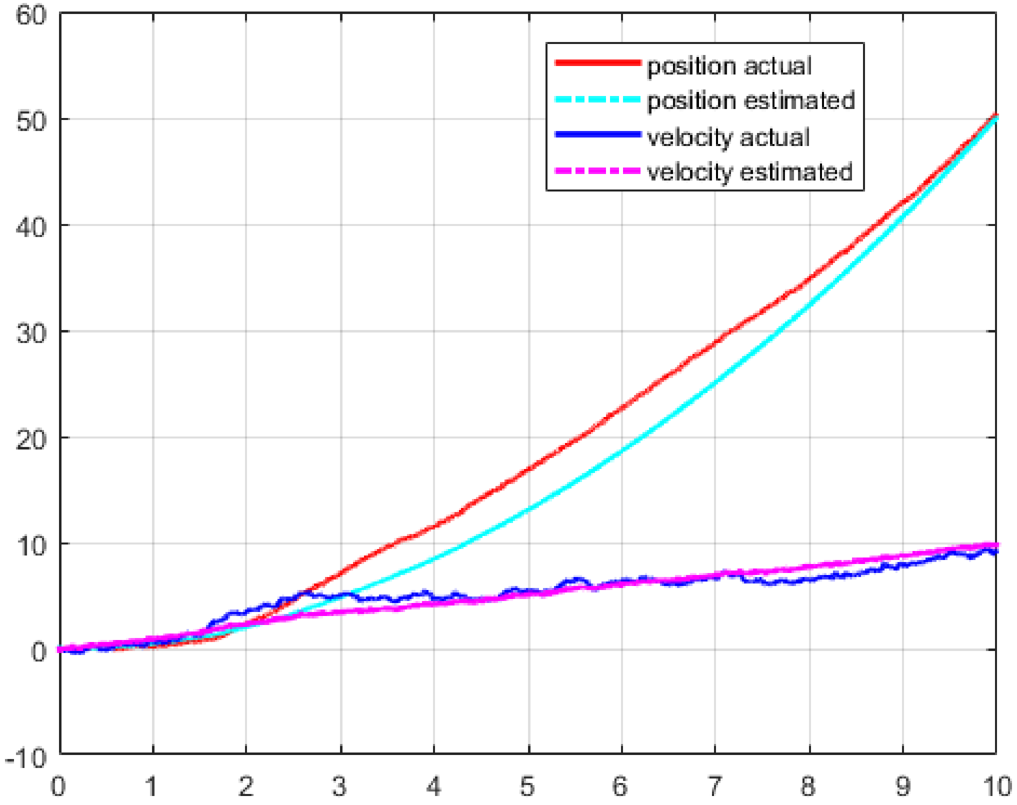



```
% state equations, discrete system
muQ = 0;
Q = [0.01 0; 0 0.03]; % we need to know this: noise covariance of state
A = [1.0000 0.0010; 0 1.0000];
B = 1.0e-03 * [0.0005; 1.0000];
wk = normrnd(muQ,Q);

% output equations
H = [1 0; 0 1];
muR = 0;
R = [0.5 0;0 0.5]; % we need to know this: noise covariance of output
vk = normrnd(muR, R);

% simulation parameters:
t0 = 0;
dt = 0.001;
tend = 10;
x = [0 0]'; % initial state
z = x; % initial measurement;
len = length(t0:dt:tend);
xfs = zeros(len, length(x));
xfs(1,:) = x';
zfs(1,:) = z';
u = 1;

% apriori x
xhat_minus = [0 0]';
xhat = [0 0]';
xhatfs = xfs;
P = [ 0.1 0.2; 0.3 0.1];
```

[3] Codes available at https://github.com/karishmapatnaik/sensor_fusion/tree/main

KFs: Measurement, Process and Noise





[3] Codes available at https://github.com/karishmapatnaik/sensor_fusion/tree/main

Sliding Window State Estimation using Optimization

Ira A. Fulton Schools of Engineering
Arizona State University

State estimation can also be formulated as a simple linear/quadratic optimization problem! Here, we know the system dynamics, so we try to find the best state estimate that the minimizes some error metric over a window.

$$y = \underbrace{x_1\, sin(t) + x_2\, sin(2t) + x_3\, sin(3t) + x_4 + sin(4t) + x_5\, sin(5t) + x_6\, sin(6t)} + noise$$

$$i.e, y = Ax + noise, with\;\; x = [x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6] \longleftarrow$$ unknown

known

$$Let\ us\ predict\ x, so\ denote\ prediction\ by\ \hat{x}$$

$$e = y - A\hat{x}$$

$$\boxed{\| e \|_1 := \sum_{i=1}^{n} |e_i|}$$

derived from L_1

$$\boxed{\| e \|_2 := \sum_{i=1}^{n} \| e_i^2 \|}$$

derived from L_2

$$\boxed{\| e \|_\infty := \max_i |e_i|}$$

derived from L_∞

Sliding Window State Estimation using Optimization

Depending on the nature of the objective function i.e, whether it is L1, L2 or L∞ norm on x, we can use LP, QP or LP respectively.

$$\boxed{\| e \|_1 := \sum_{i=1}^{n} |e_i|}$$

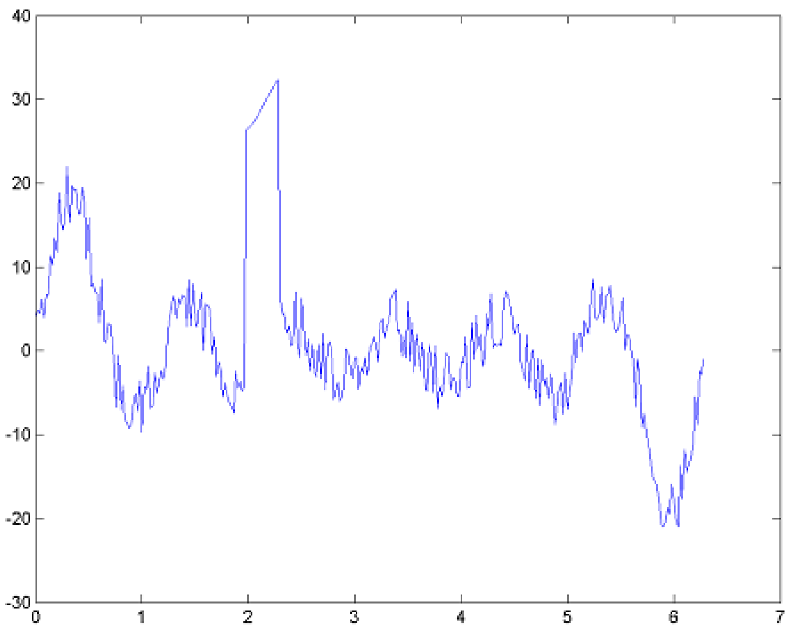
derived from L_1

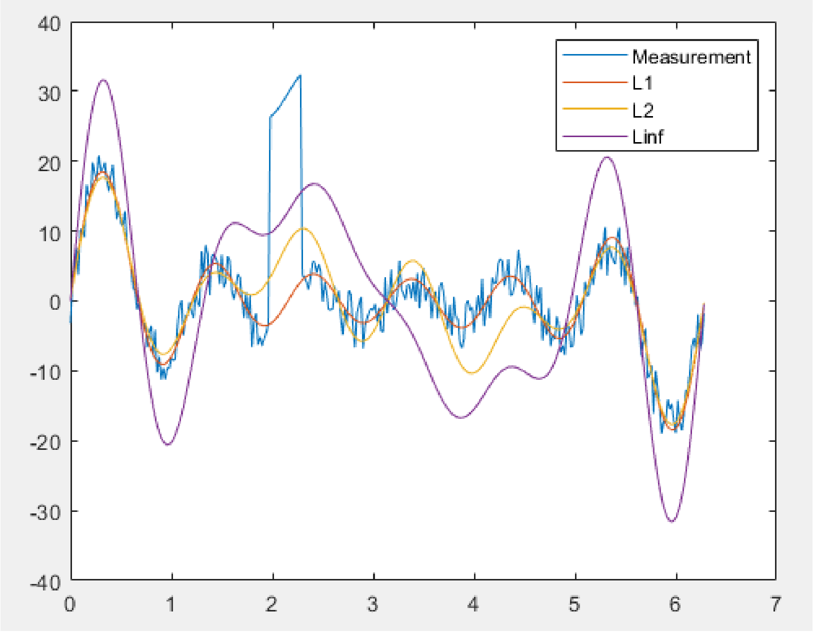
$$\boxed{\| e \|_2 := \sum_{i=1}^{n} \| e_i^2 \|}$$

derived from L_2

$$\boxed{\| e \|_\infty := \max_i |e_i|}$$

derived from L_∞





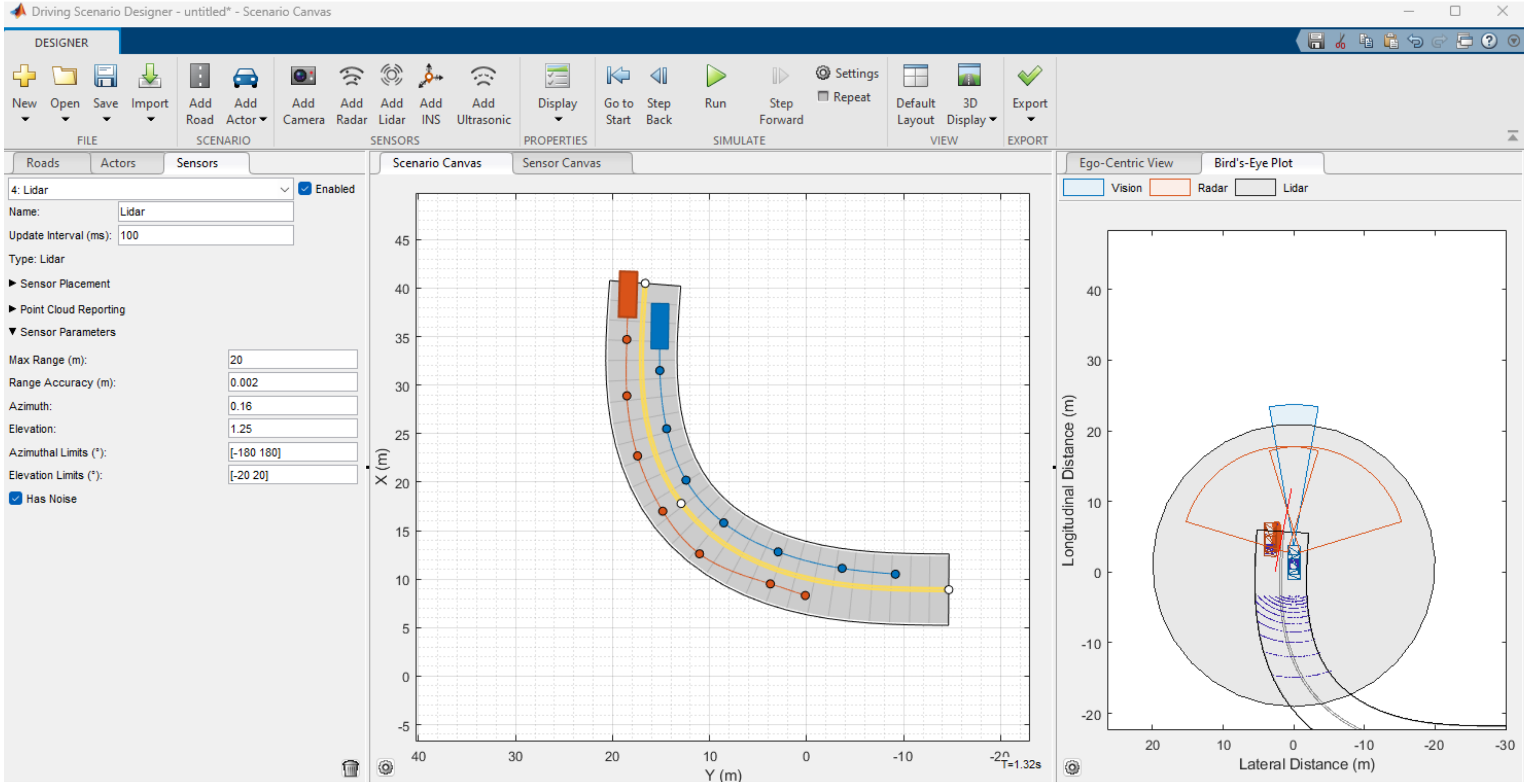
Implementing KFs in Python and ROS2

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32, Float32MultiArray
import numpy as np
from filterpy.kalman import KalmanFilter
```

```python
class KalmanFilterNode(Node):
    def __init__(self):
        super().__init__('kalman_filter_node')

        # Subscriber for measurements
        self.subscription = self.create_subscription(
            Float32, 'measurement', self.measurement_callback, 10)

        # Publisher for estimated state
        self.state_publisher = self.create_publisher(Float32MultiArray, 'estimated_state', 10)

        # Kalman filter initialization
        dt = 0.1  # Time step
        self.kf = KalmanFilter(dim_x=2, dim_z=1)
        self.kf.x = np.array([0, 0])  # Initial state: [position, velocity]
        self.kf.F = np.array([[1, dt], [0, 1]])  # State transition matrix
        self.kf.H = np.array([[1, 0]])  # Measurement matrix
        self.kf.P = np.eye(2) * 500  # Covariance matrix (large initial uncertainty)
        self.kf.R = 1  # Measurement noise covariance
        self.kf.Q = np.array([[0.1, 0], [0, 0.1]])  # Process noise covariance
```

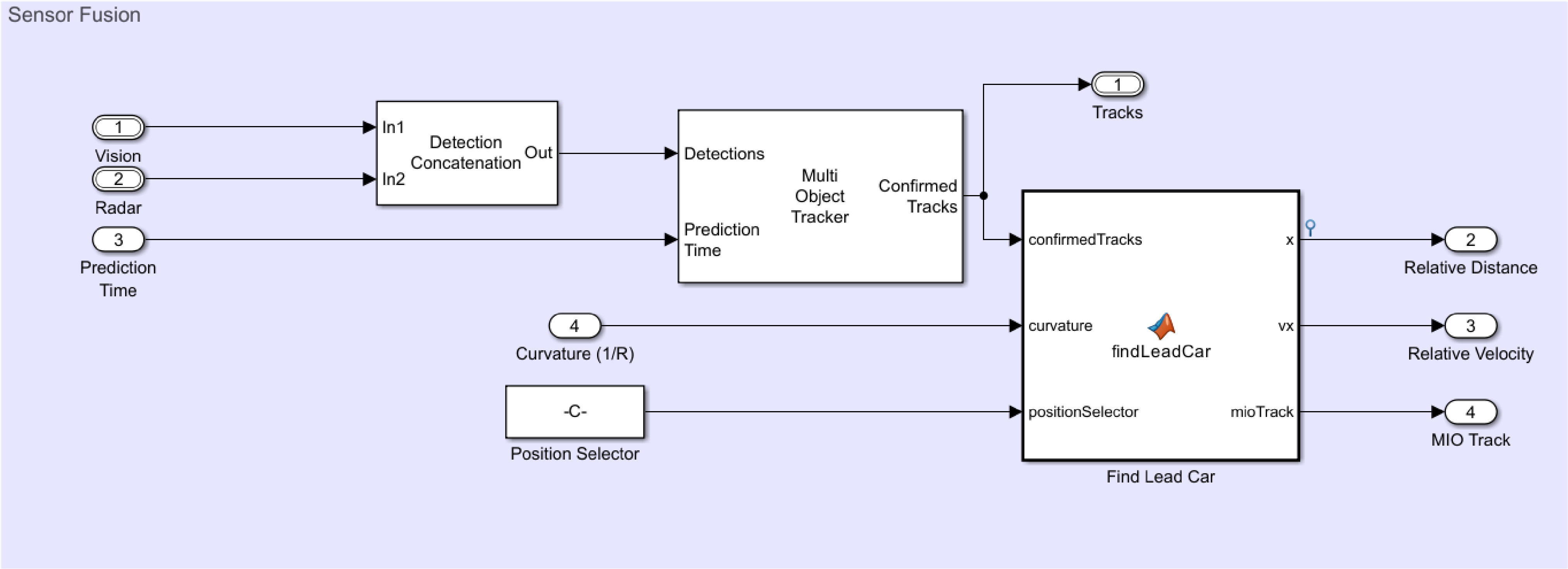
[3] Codes available at https://github.com/karishmapatnaik/sensor_fusion/tree/main

Radars, Cameras and Lidars
Creating Synthetic Data in MATLAB



[3] Codes available at https://github.com/karishmapatnaik/sensor_fusion/tree/main
[4] https://www.mathworks.com/help/driving/ug/create-driving-scenario-interactively-and-generate-synthetic-detections.html

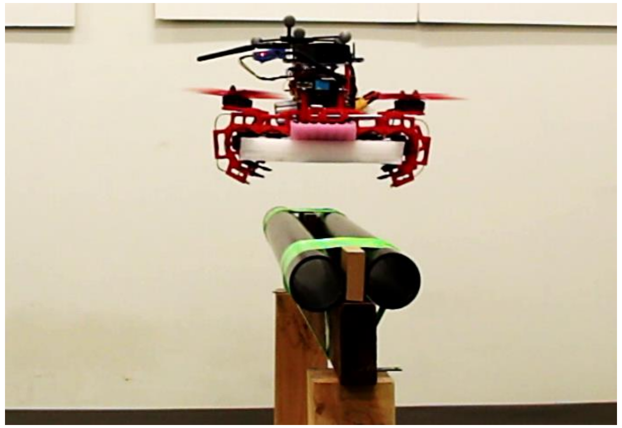
Creating Synthetic Data in MATLAB
Visualization and Fusing



[3] Codes available at https://github.com/karishmapatnaik/sensor_fusion/tree/main
[4] https://www.mathworks.com/help/driving/ug/create-driving-scenario-interactively-and-generate-synthetic-detections.html

Ira A. Fulton Schools of Engineering
Arizona State University



Questions