Gesture Recognition & Air Drumming

Ferdi Lesporis, Eric Li, Ben Gur

Overview

- Introduction
- Background and Foreground Detection for Gestures
- Thresholding
- Movement Tracking
- Kalman Filter and related
- Future Considerations

Gesture Recognition

1. Seperate the hand from the rest of the background

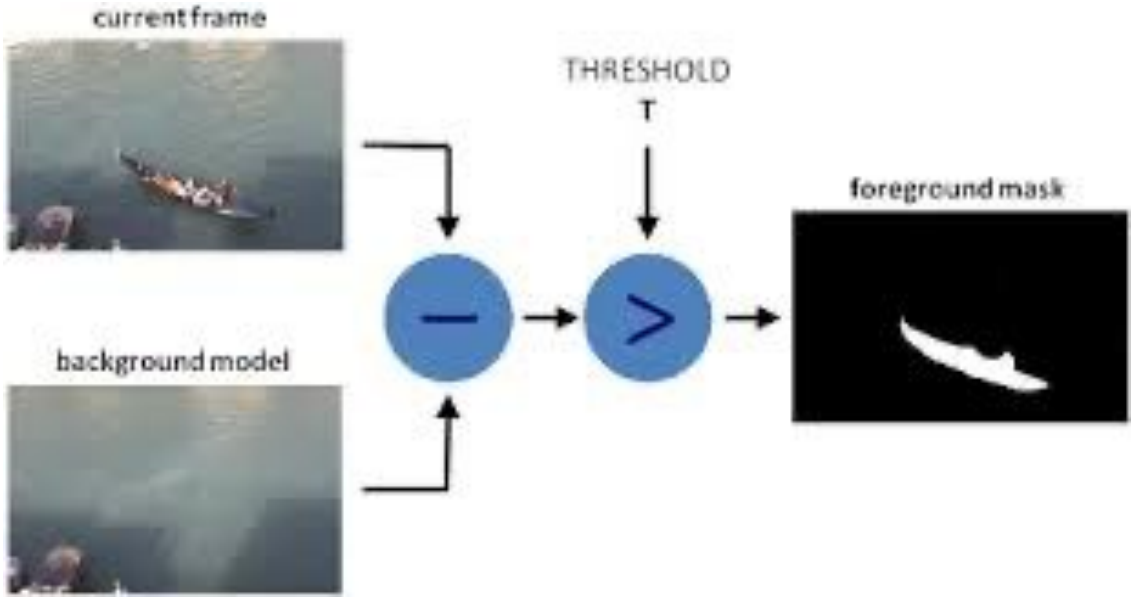
2. Use Convex Hull to detect gestures (Hand signs)

Methods for Seperate the hand from the rest of the background

1. Background Subtraction

2. Projecting into different color spaces

Background Subtraction

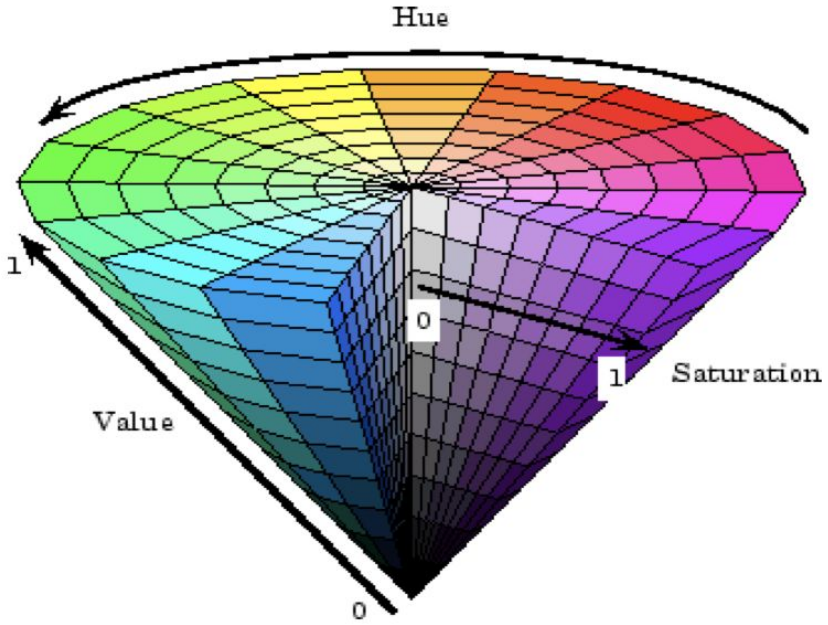


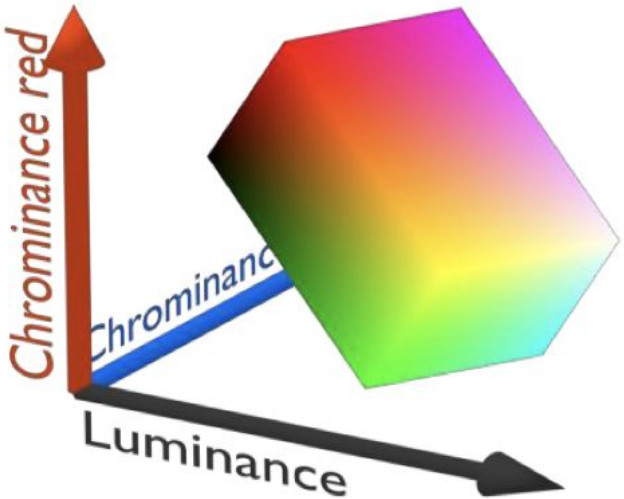
Three Phases in Background Subtraction

1. Create a Background Model
2. Foreground Detection
3. Background Maintenance

Skin Detection

HSV vs YCbCr





Thresholding

Position tracking uses thresholding.

GaussianBlur is applied first to remove noise and allow for more accurate thresholding/ better contour finding.

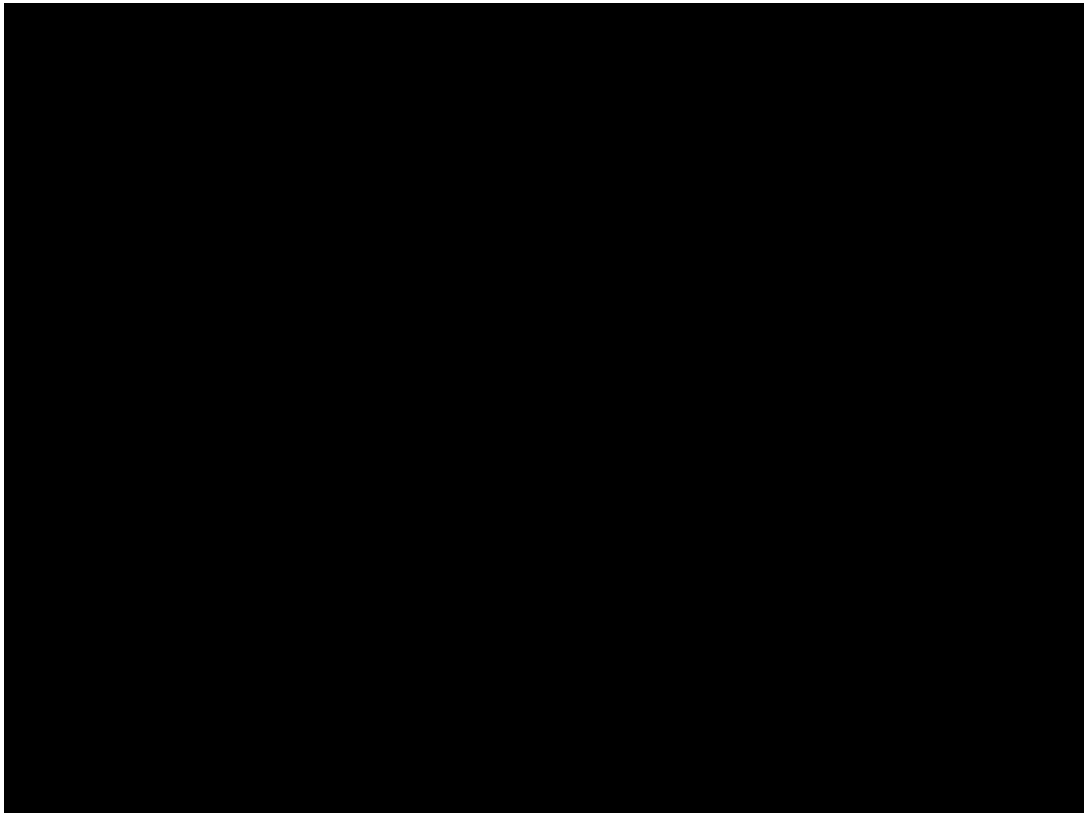
Thresholding via the HSV color space. Allows for accurate differentiation even in uneven backgrounds.

OpenCV find contourArea function then finds the object. Finds the largest object in the scene.

Center then found using the objects moments.

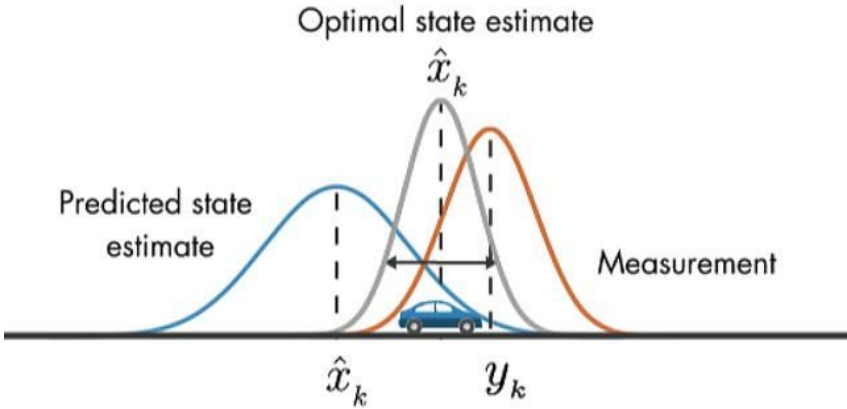
Minimum size found via an enclosing circle.

Movement Tracking



Kalman Filter

- Optimal estimation algo; uses past measurements to predict future
- When it’s hard to measure directly, noise and errors accumulate
- Used in trajectory prediction and navigation in computer vision
- AKA linear quadratic estimation



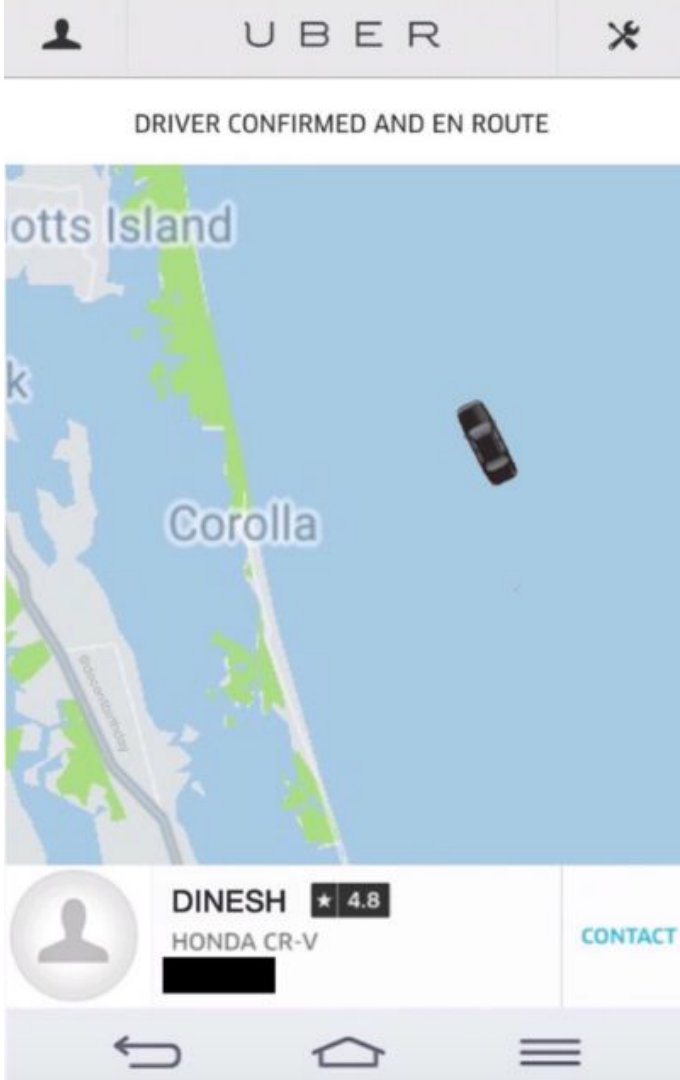
Computer Vision Example

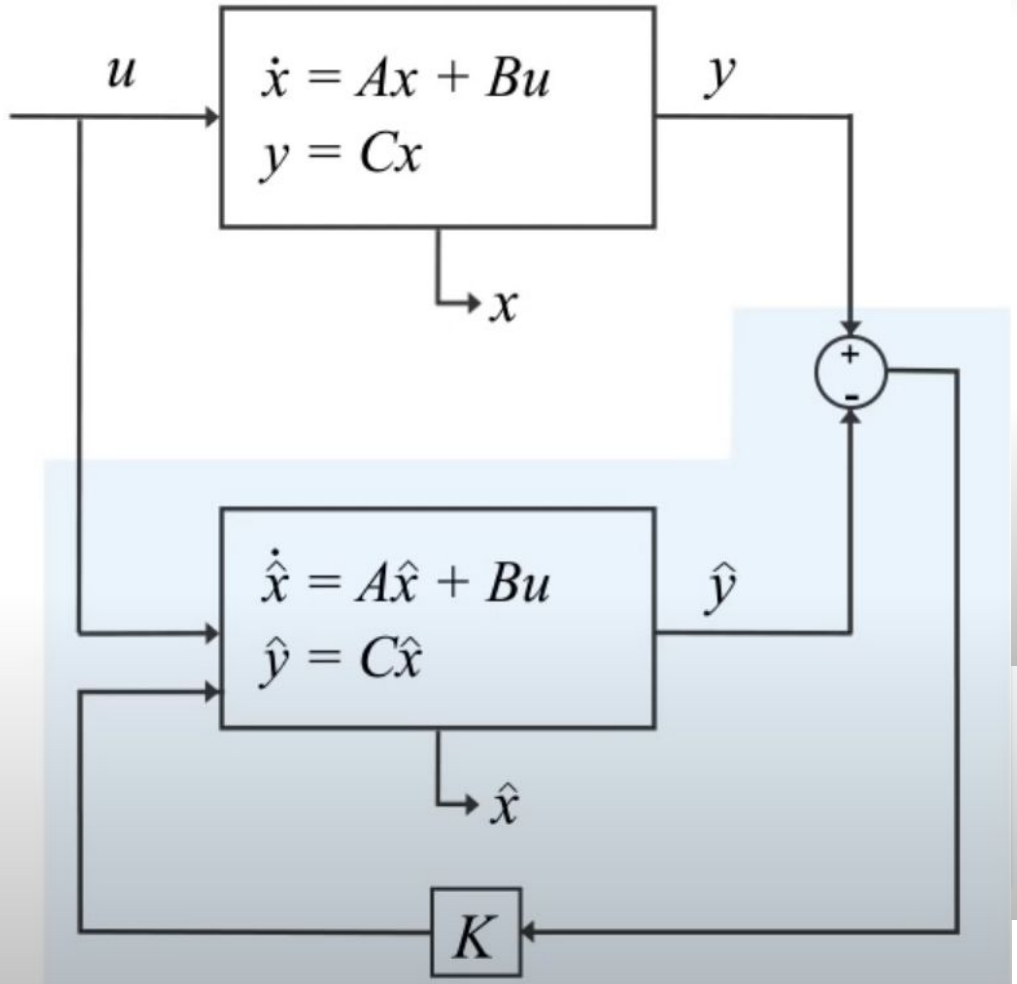
Onboard sensors:

- Acceleration and angular velocity (gyroscopes etc)
- Relative position (odometer)
- Accurate but prone to drift

GPS:

- Absolute position
- Noisy





$e_{obs} = x - \hat{x}$

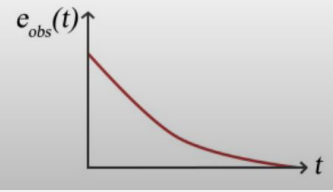
$\dot{x} = Ax + Bu$ $\quad y = Cx$

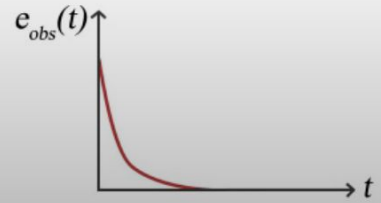
$\dot{\hat{x}} = A\hat{x} + Bu + \cancel{K(y - \hat{y})}$ $\quad \hat{y} = C\hat{x}$

$\dot{e}_{obs} = (A - \cancel{KC})e_{obs}$ $\quad y - \hat{y} = Ce_{obs}$

$\hookrightarrow e_{obs}(t) = e^{(A - \cancel{KC})t} e_{obs}(0)$

If $(A - \cancel{KC}) < 0$, then $e_{obs} \to 0$ as $t \to \infty$. So, $\hat{x} \to x$.

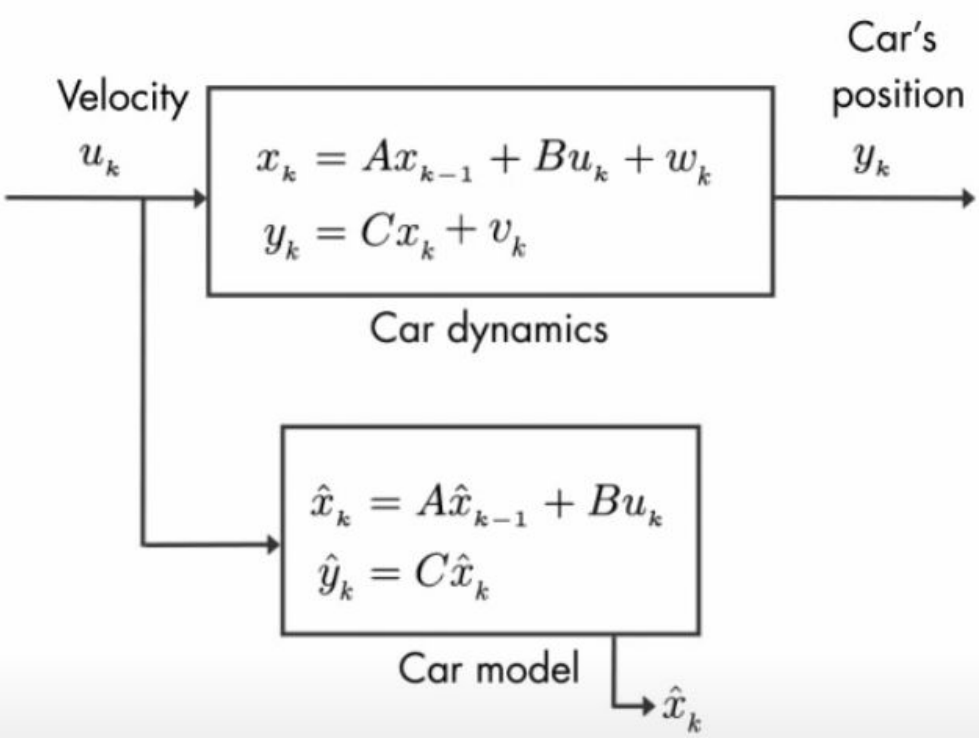




Stochastic Case

- Wk = process noise ~ N(0, Q)
- Vk = measurement noise ~ N(0,Q)
- Want as many parameters for Xk

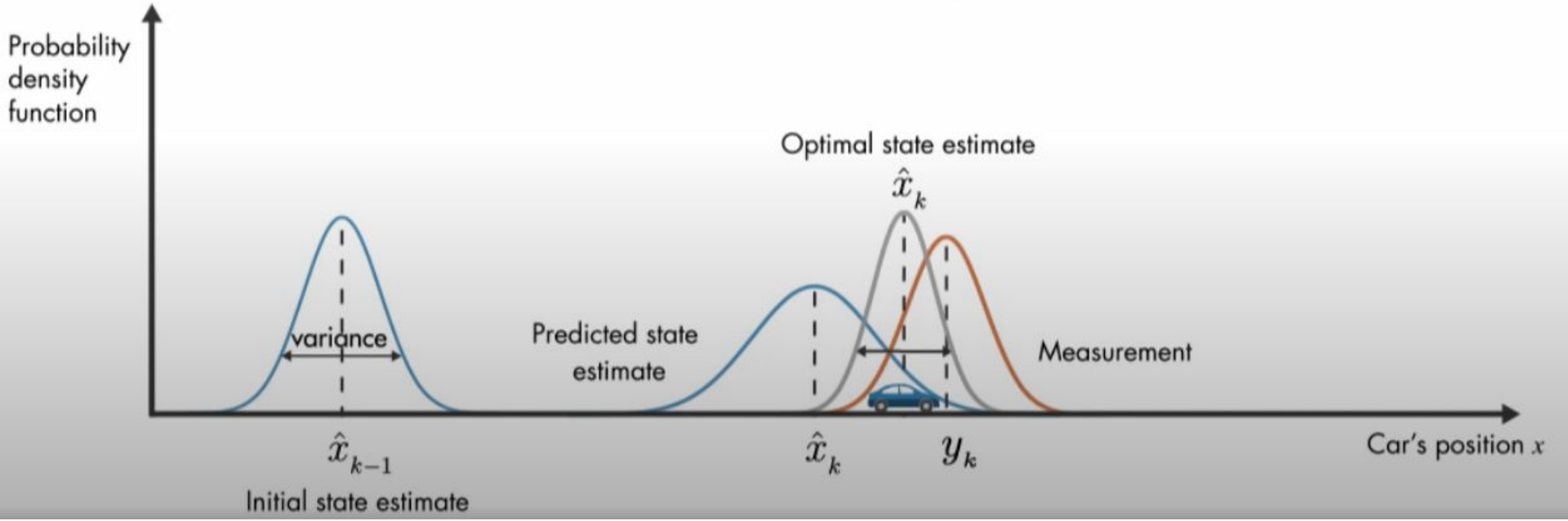
(Assumes linearity! Extended, unscented etc.)



$$x_k = [position]$$

$$C = 1$$

Visual


Probability density function
Optimal state estimate
$\hat{x}_k$
variance
Predicted state estimate
Measurement
$\hat{x}_{k-1}$
Initial state estimate
$\hat{x}_k$
y_k
Car's position x

Kalman Algo

Predict:

- Prior state estimate
- Error covariance

Update:

- Use priori estimate to find posterior estimate (state and error covariance)
- Kalman gain minimizes posterior error covariance

Prediction

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$$

$$P_k^- = AP_{k-1}A^T + Q$$

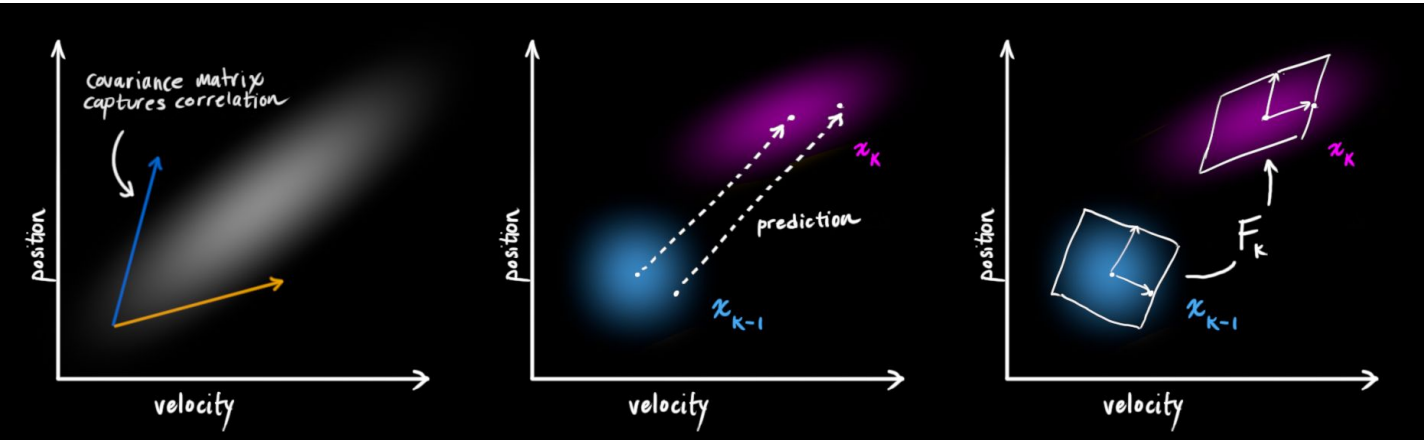
Update

$$K_k = \frac{P_k^- C^T}{CP_k^- C^T + R}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-)$$

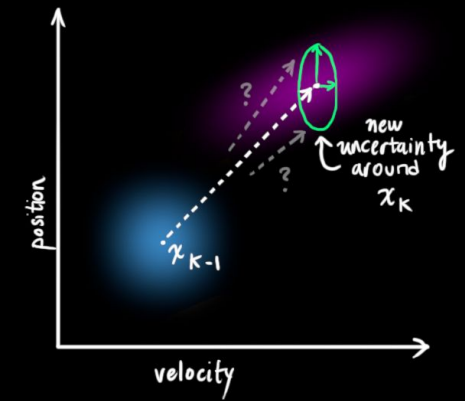
$$P_k = (I - K_k C)P_k^-$$

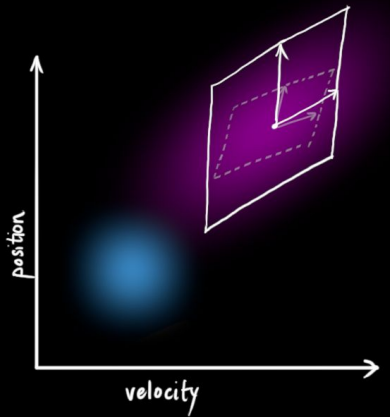
Pictures

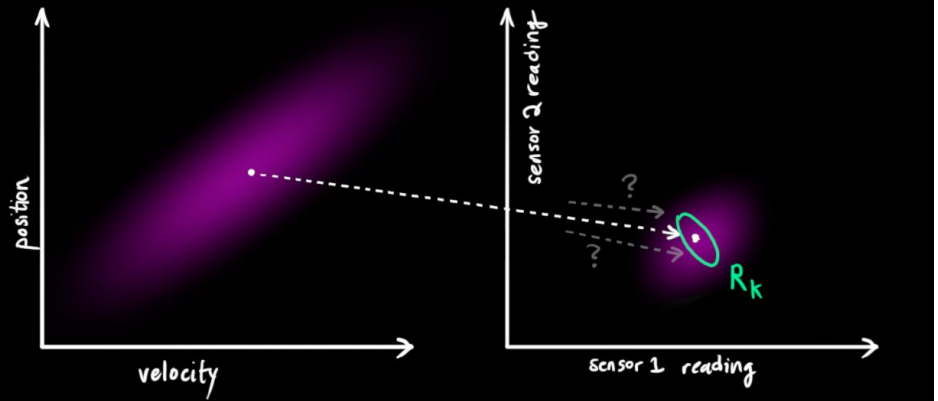

covariance matrix captures correlation
position
velocity

x_K
prediction
x_{K-1}
position
velocity

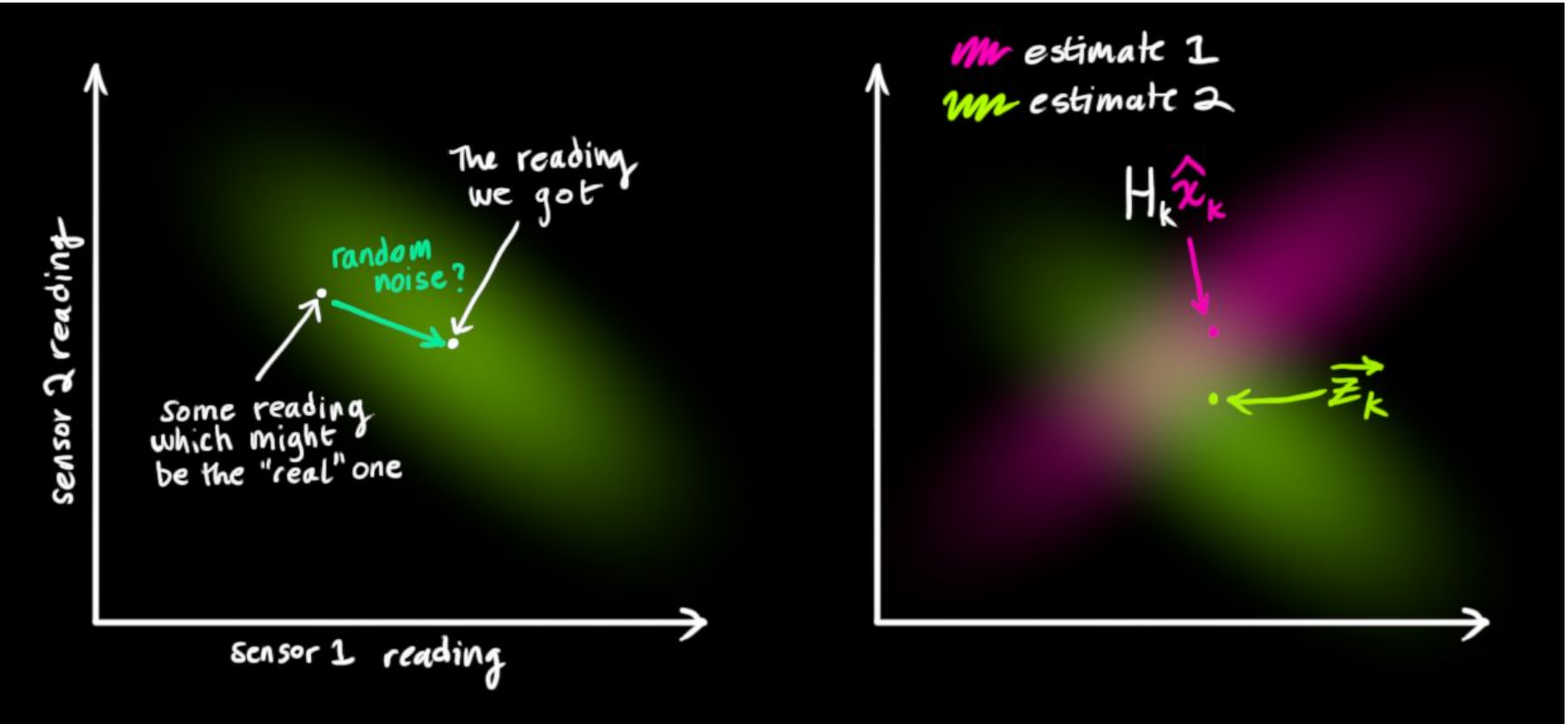
x_K
F_k
x_{K-1}
position
velocity


new uncertainty around x_K
x_{K-1}
position
velocity


position
velocity


sensor 2 reading
R_k
position
velocity
sensor 1 reading

More Pictures



Actual Algo

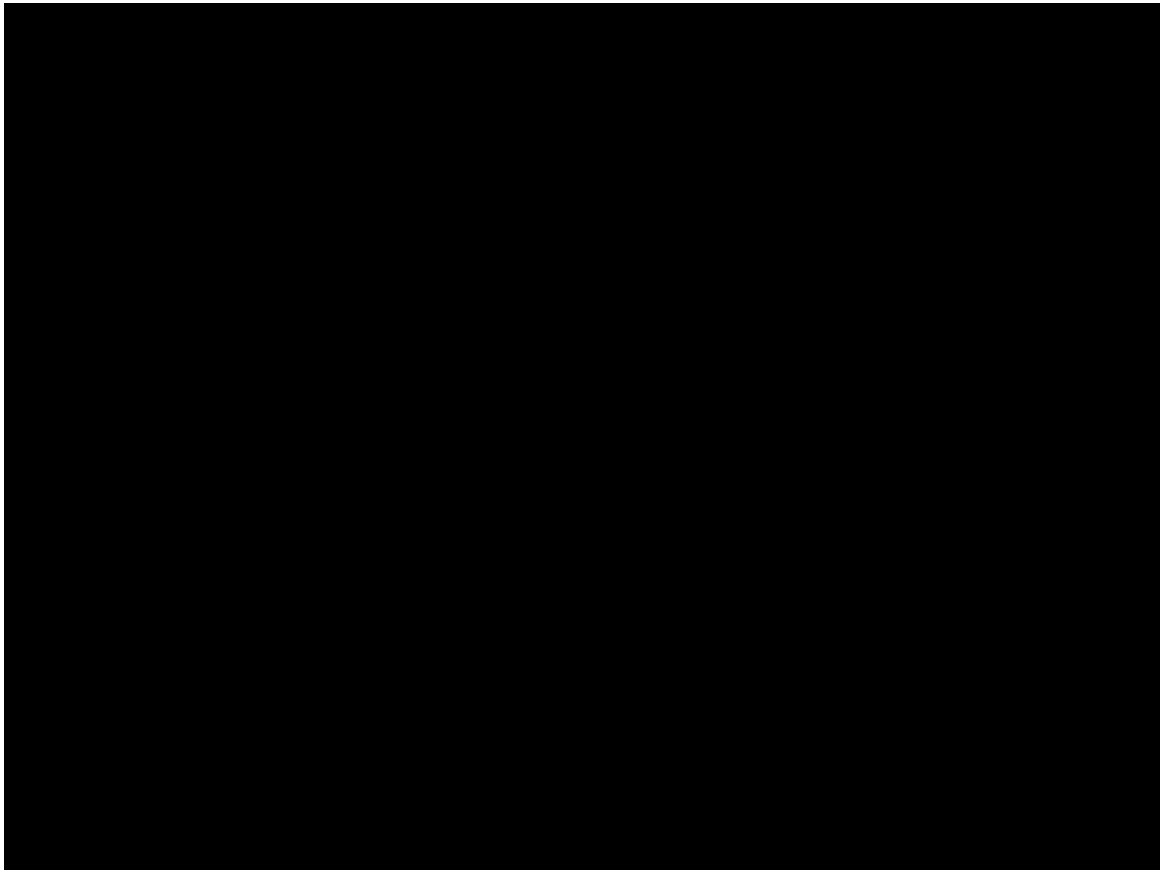
- Kalman class
  - Measurement matrix
  - Prediction matrix
  - Process noise covariance (default 1)
  - Measurement noise covariance (default 1)
  - Error covariance (default 0)
- Update predictions
  - Prediction matrix = [x, y, x_velocity, y_velocity]^T

Code

```python
#init Kalman
kalman = cv2.KalmanFilter(4,2)

dt = 1 #step interval
kalman.measurementMatrix = np.array([[1,0,0,0],
                                     [0,1,0,0]],np.float32)

kalman.transitionMatrix = np.array([[1,0,dt,0],
                                    [0,1,0,dt],
                                    [0,0,1,0],
                                    [0,0,0,1]],np.float32)

kalman.processNoiseCov = np.array([[1,0,0,0],
                                   [0,1,0,0],
                                   [0,0,1,0],
                                   [0,0,0,1]],np.float32) * 0.01 #smoothing

f = 0.1845 #from paper
kalman.measurementNoiseCov = np.array([[f,f/40],
                                       [f/40,f/4]],np.float32)

prediction = np.zeros((4,1), np.float32)
```

```
[[0.]
 [0.]
 [0.]
 [0.]]

[[6.079208 ]
 [2.5445545]
 [0.       ]
 [0.       ]]

[[31.278164 ]
 [12.189723 ]
 [ 6.315371 ]
 [ 2.4172752]]

[[104.84807]
 [ 36.91534]
 [ 24.76144]
 [  8.53584]]

[[255.91873 ]
 [ 81.72509 ]
 [ 56.760105]
 [ 17.725325]]
```

Kalman vs OpenCV Camshift



One Euro Filter

- Another way to control for noise
- Uses:
  - smoothing factor
  - cut off frequency
- Less intensive version of Kalman
- Useful for high lag (adjust cut off frequency)

Neural Network (Attempt)

- CNN for gesture recognition
    - Simple classification
    - VGG 16
- Severe issues with lighting
- F1 score < 60%
    - Transformations did little
- NOT PRESENT IN PROJECT BUT WAS EXPLORED

Future Considerations

- Adaptively adjust R and Q
- Integrate optimizations to improve detection
- Test post-processing instead of real-time
- Add more sounds and related

References

- A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos by Andrews Sobral, Antoine Vacavant