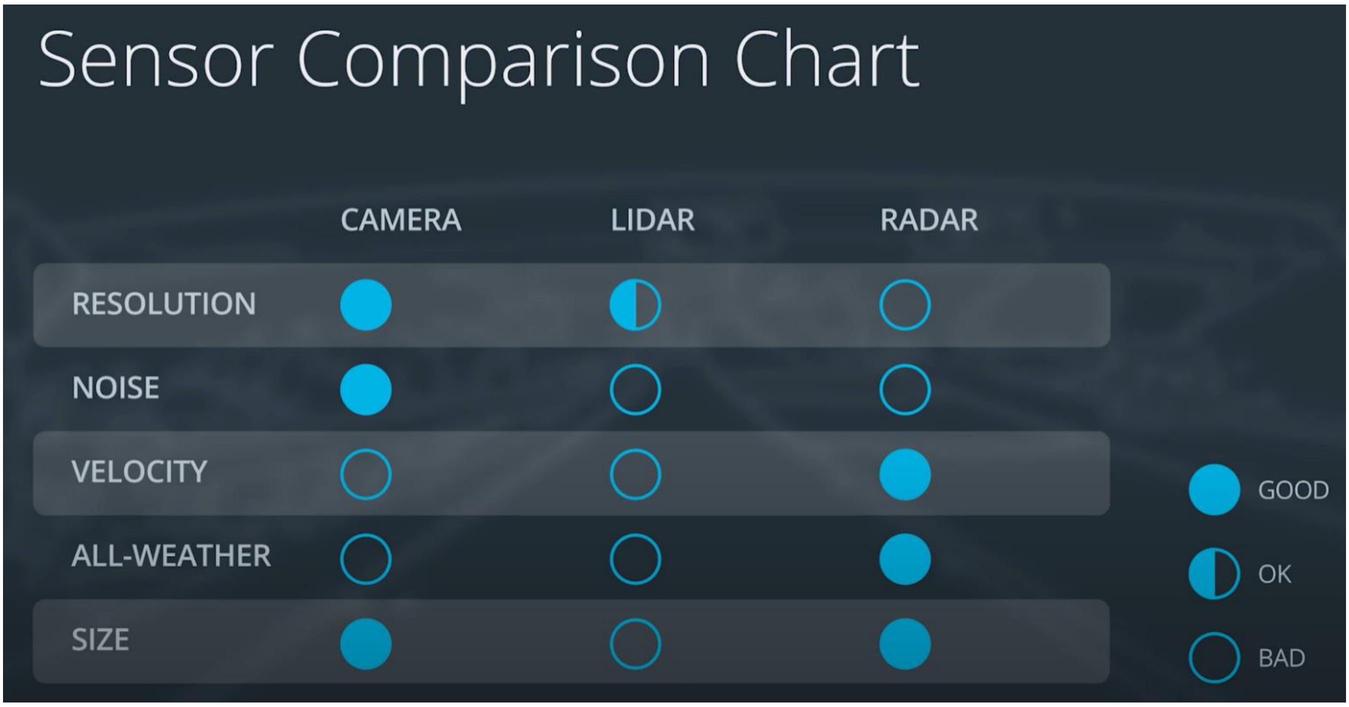


Sensor Comparison Chart

	CAMERA	LIDAR	RADAR
RESOLUTION	●	◐	○
NOISE	●	○	○
VELOCITY	○	○	●
ALL-WEATHER	○	○	●
SIZE	●	○	●

● GOOD

◐ OK

○ BAD

QUIZ QUESTION

Here are a few properties of lidar and radar sensors. Drag and drop the sensor labels to the properties that they describe.

Submit to check your answer choices!

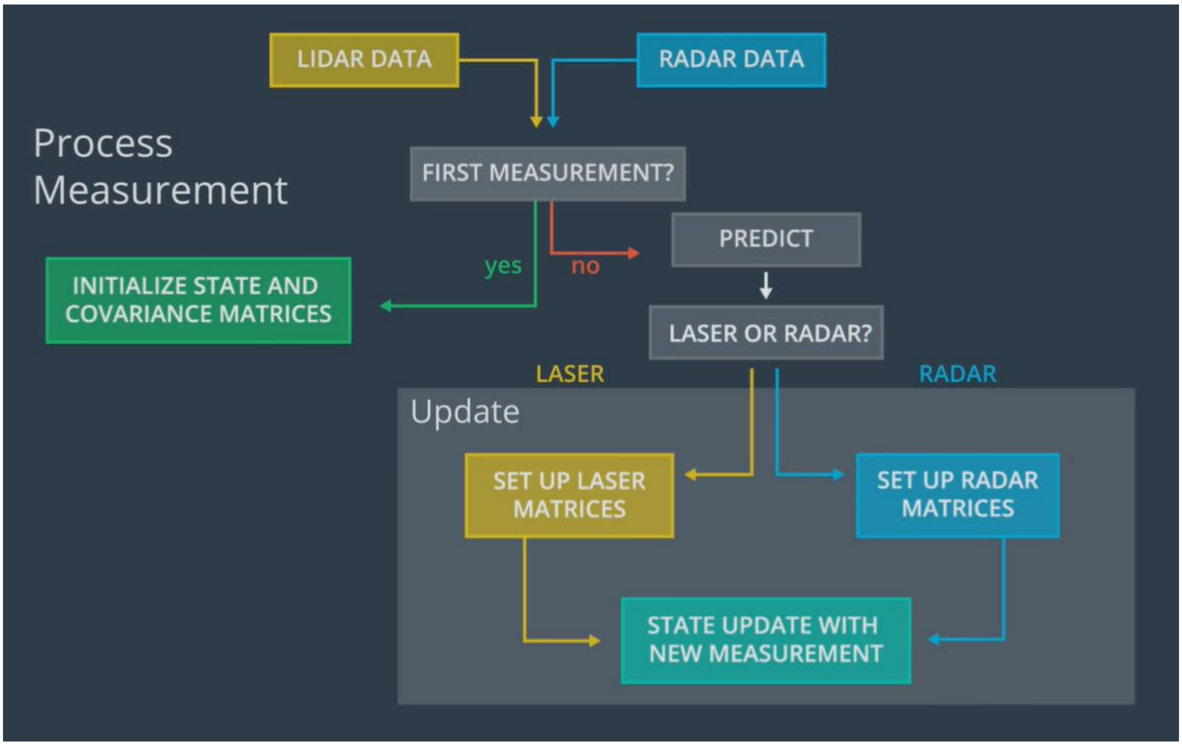
PROPERTIES	LIDAR OR RADAR?
Wavelength in mm	Radar
Wavelength in infrared	Lidar
Can sense non-line of sight objects	Radar
Most affected by dirt and small debris	Lidar
Creates a point cloud	Lidar
Can currently directly measure velocity	Radar
Higher Resolution	Lidar

The first cycle is the **Measurement Update**.

- Requires a product
- Uses Bayes rule.

The second cycle is the **Motion Update**.

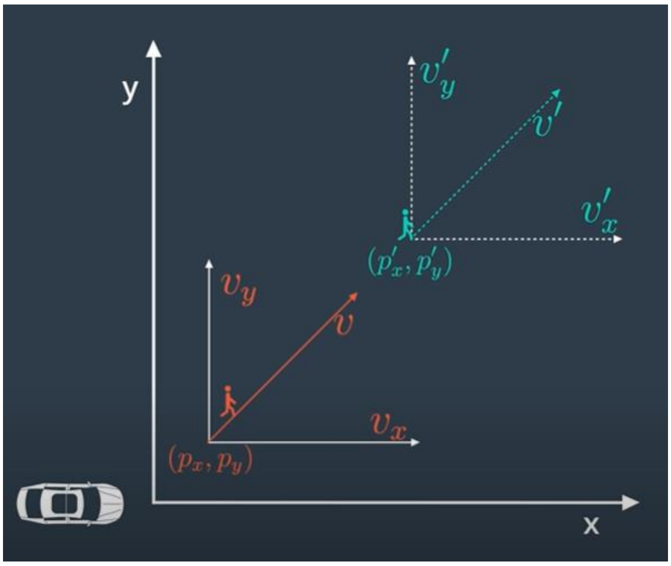
- Involves a convolution
- Uses total probability.

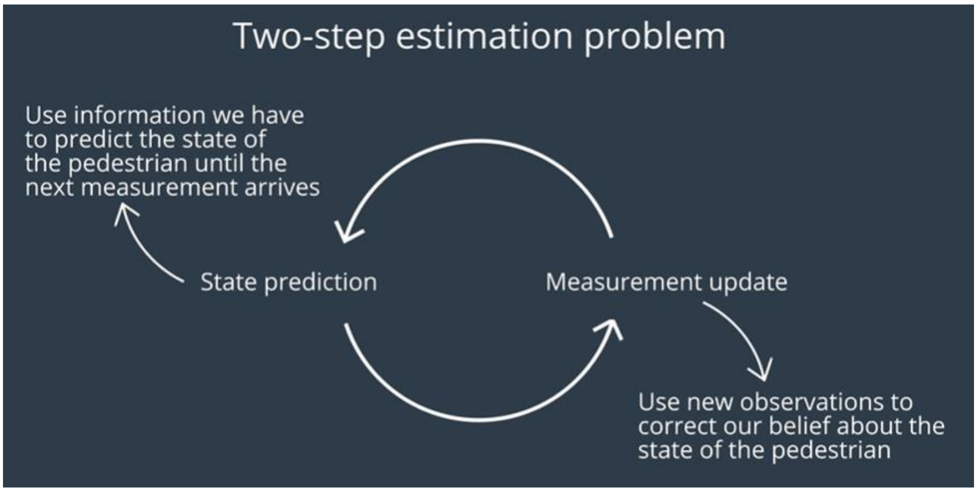


a bicycle. For variety, let's step through the Kalman Filter algorithm using the bicycle example.

The Kalman Filter algorithm will go through the following steps:

- **first measurement** - the filter will receive initial measurements of the bicycle's position relative to the car. These measurements will come from a radar or lidar sensor.
- **initialize state and covariance matrices** - the filter will initialize the bicycle's position based on the first measurement.
- then the car will receive another sensor measurement after a time period Δt.
- **predict** - the algorithm will predict where the bicycle will be after time Δt. One basic way to predict the bicycle location after Δt is to assume the bicycle's velocity is constant; thus the bicycle will have moved velocity * Δt. In the extended Kalman filter lesson, we will assume the velocity is constant.
- **update** - the filter compares the "predicted" location with what the sensor measurement says. The predicted location and the measured location are combined to give an updated location. The Kalman filter will put more weight on either the predicted location or the measured location depending on the uncertainty of each value.
- then the car will receive another sensor measurement after a time period Δt. The algorithm then does another **predict** and **update** step.





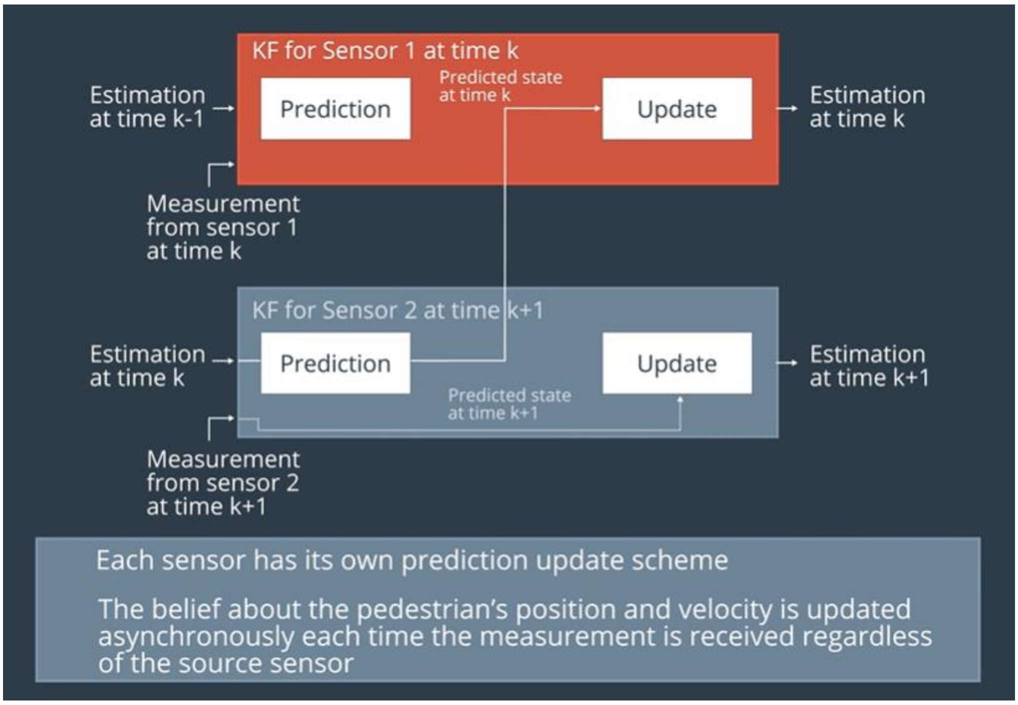
Two-step estimation problem

Use information we have to predict the state of the pedestrian until the next measurement arrives

State prediction

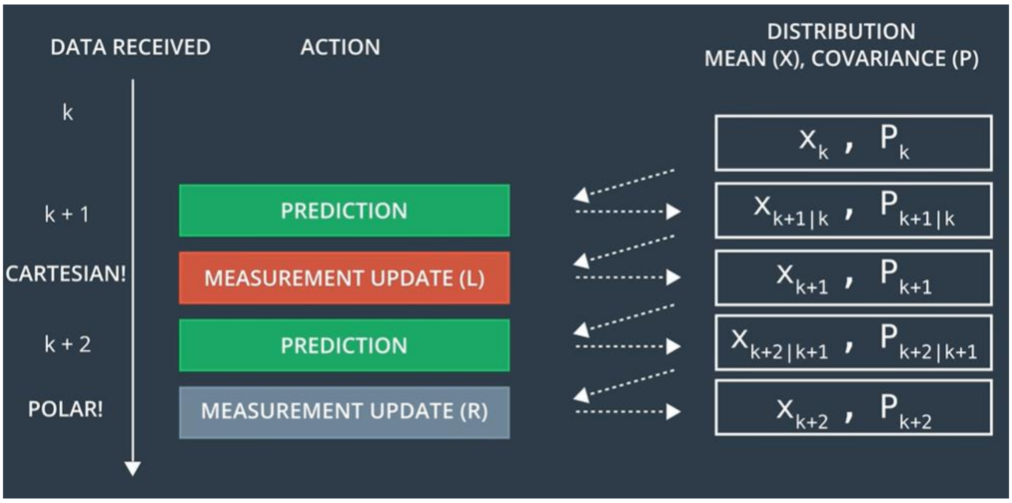
Measurement update

Use new observations to correct our belief about the state of the pedestrian


KF for Sensor 1 at time k
Estimation at time k-1
Prediction
Predicted state at time k
Update
Estimation at time k
Measurement from sensor 1 at time k
KF for Sensor 2 at time k+1
Estimation at time k
Prediction
Update
Estimation at time k+1
Predicted state at time k+1
Measurement from sensor 2 at time k+1

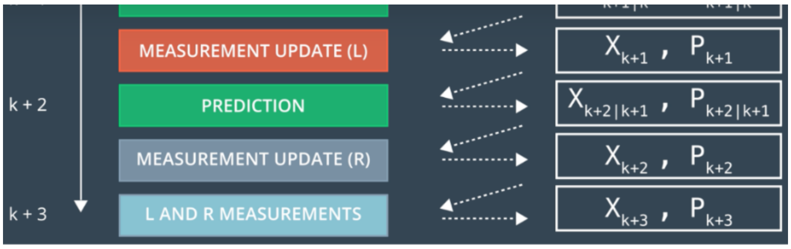
Each sensor has its own prediction update scheme

The belief about the pedestrian's position and velocity is updated asynchronously each time the measurement is received regardless of the source sensor



Definition of Variables

- x is the mean state vector. For an extended Kalman filter, the mean state vector contains information about the object's position and velocity that you are tracking. It is called the "mean" state vector because position and velocity are represented by a gaussian distribution with mean x.
- P is the state covariance matrix, which contains information about the uncertainty of the object's position and velocity. You can think of it as containing standard deviations.
- k represents time steps. So x_k refers to the object's position and velocity vector at time k.
- The notation $k+1|k$ refers to the prediction step. At time $k+1$, you receive a sensor measurement. Before taking into account the sensor measurement to update your belief about the object's position and velocity, you predict where you think the object will be at time $k+1$. You can predict the position of the object at $k+1$ based on its position and velocity at time k. Hence $x_{k+1|k}$ means that you have predicted where the object will be at $k+1$ but have not yet taken the sensor measurement into account.
- x_{k+1} means that you have now predicted where the object will be at time $k+1$ and then used the sensor measurement to update the object's position and velocity.



QUIZ QUESTION

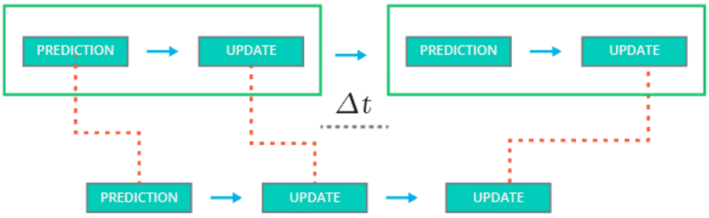
What should a Kalman Filter do if both the radar and laser measurements arrive at the same time, **k+3**? Hint: The Kalman filter algorithm predicts -> updates -> predicts -> updates, etc. If two sensor measurements come in simultaneously, the time step between the first measurement and the second measurement would be zero.

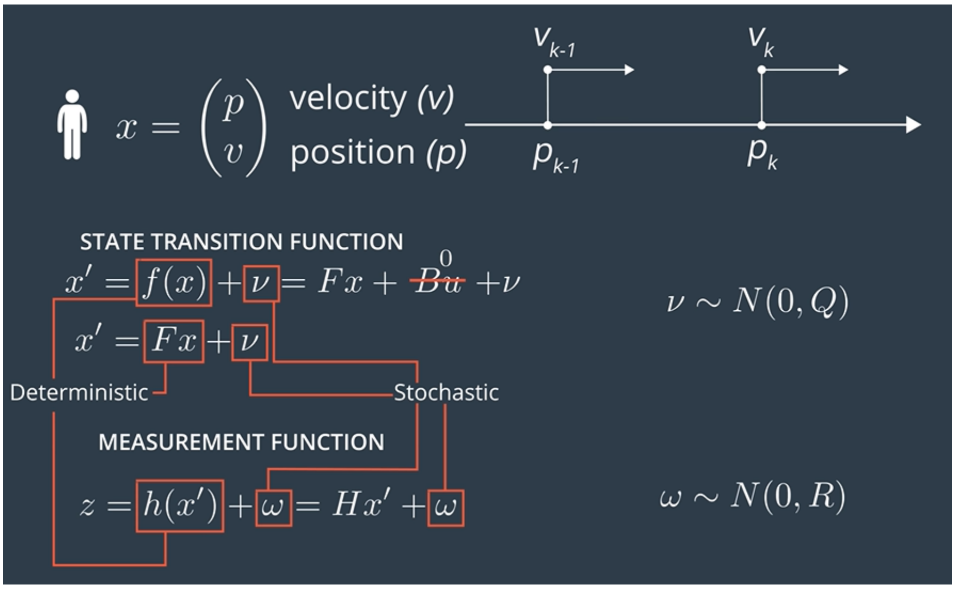
○ Predict the state to k+3, and then update the state with only one of those measurements, either laser or radar.

○ Predict the state to k+3, and then only update with the laser measurement because it is more accurate

✓ Predict the state to k+3 then use either one of the sensors to update. Then predict the state to k+3 again and update with the other sensor measurement.

○ Skip the prediction step because we have two measurements. Just update the prior probability distribution twice.





$x = \begin{pmatrix} p \\ v \end{pmatrix}$ velocity *(v)* position *(p)*

STATE TRANSITION FUNCTION

$$x' = \boxed{f(x)} + \boxed{\nu} = Fx + \overset{0}{\cancel{Bu}} + \nu$$

$$\nu \sim N(0, Q)$$

$$x' = \boxed{Fx} + \boxed{\nu}$$

Deterministic

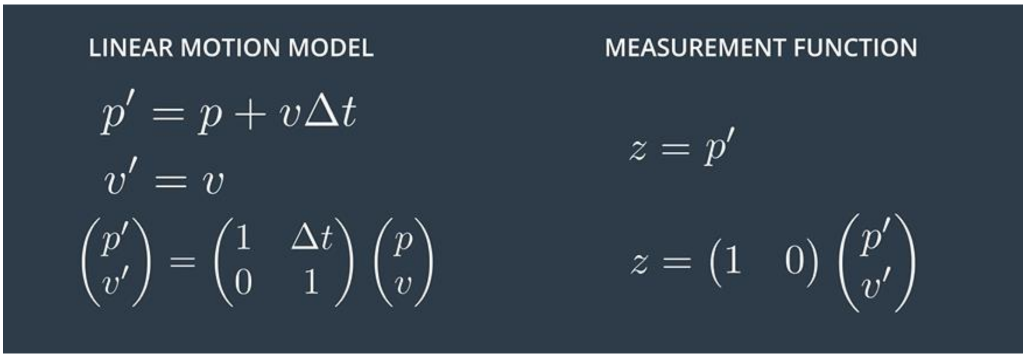
Stochastic

MEASUREMENT FUNCTION

$$z = \boxed{h(x')} + \boxed{\omega} = Hx' + \boxed{\omega}$$

$$\omega \sim N(0, R)$$

With v constant:



LINEAR MOTION MODEL

$$p' = p + v\Delta t$$

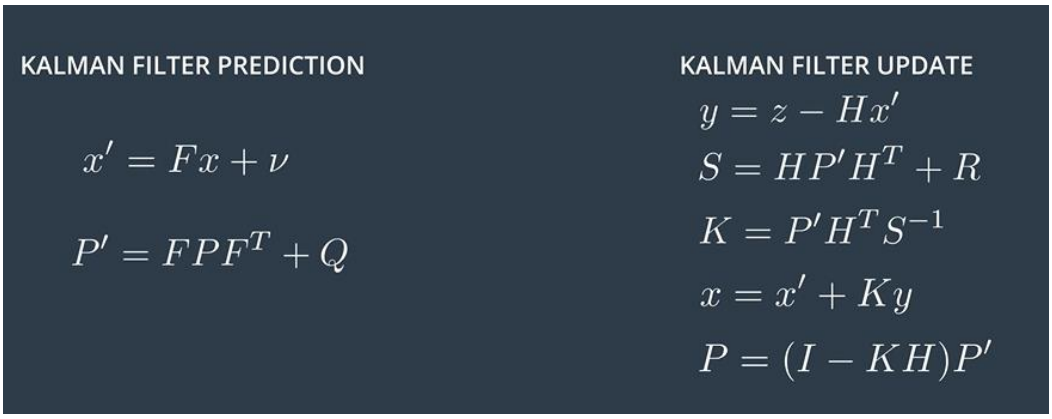
$$v' = v$$

$$\begin{pmatrix} p' \\ v' \end{pmatrix} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} p \\ v \end{pmatrix}$$

MEASUREMENT FUNCTION

$$z = p'$$

$$z = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} p' \\ v' \end{pmatrix}$$



KALMAN FILTER PREDICTION

$$x' = Fx + \nu$$

$$P' = FPF^T + Q$$

KALMAN FILTER UPDATE

$$y = z - Hx'$$

$$S = HP'H^T + R$$

$$K = P'H^TS^{-1}$$

$$x = x' + Ky$$

$$P = (I - KH)P'$$

Prediction

Let's say we know an object's current position and velocity , which we keep in the x variable. Now one second has passed. We can predict where the object will be one second later because we knew the object position and velocity one second ago; we'll just assume the object kept going at the same velocity.

The $x' = Fx + \nu$ equation does these prediction calculations for us.

But maybe the object didn't maintain the exact same velocity. Maybe the object changed direction, accelerated or decelerated. So when we predict the position one second later, our uncertainty increases. $P' = FPF^T + Q$ represents this increase in uncertainty.

Process noise refers to the uncertainty in the prediction step. We assume the object travels at a constant velocity, but in reality, the object might accelerate or decelerate. The notation $\nu \sim N(0, Q)$ defines the process noise as a gaussian distribution with mean zero and covariance Q.

Update

Now we get some sensor information that tells where the object is relative to the car. First we compare where we think we are with what the sensor data tells us $y = z - Hx'$.

The K matrix, often called the Kalman filter gain, combines the uncertainty of where we think we are P' with the uncertainty of our sensor measurement R. If our sensor measurements are very uncertain (R is high relative to P'), then the Kalman filter will give more weight to where we think we are: x'. If where we think we are is uncertain (P' is high relative to R), the Kalman filter will put more weight on the sensor measurement: z.

Measurement noise refers to uncertainty in sensor measurements. The notation $\omega \sim N(0, R)$ defines the measurement noise as a gaussian distribution with mean zero and covariance R. Measurement noise comes from uncertainty in sensor measurements.

A Note About the State Transition Function: Bu

If you go back to the video, you'll notice that the state transition function was first given as $x' = Fx + Bu + \nu$.

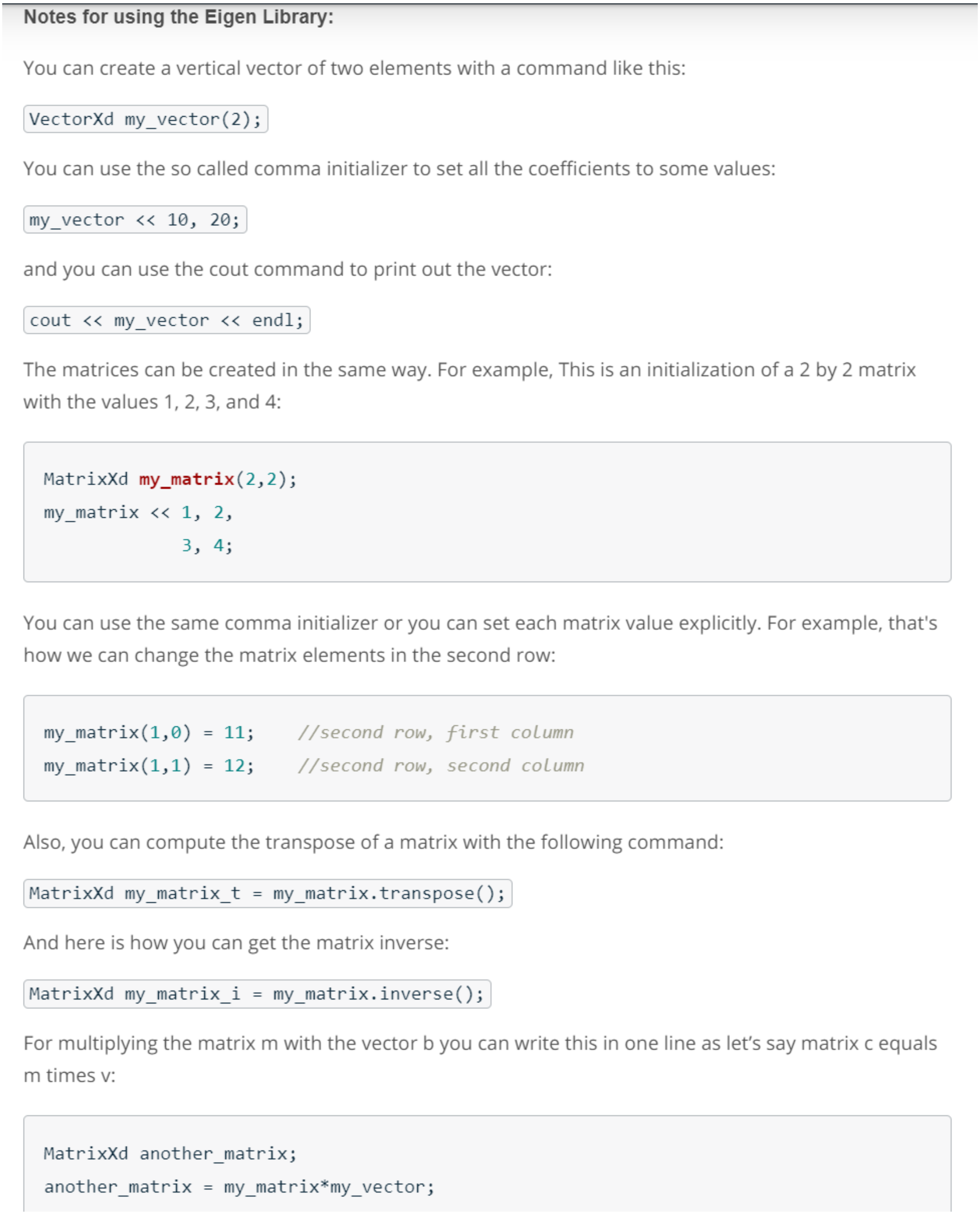
But then Bu was crossed out leaving $x' = Fx + \nu$.

B is a matrix called the control input matrix and u is the control vector.

As an example, let's say we were tracking a car and we knew for certain how much the car's motor was going to accelerate or decelerate over time; in other words, we had an equation to model the exact amount of acceleration at any given moment. Bu would represent the updated position of the car due to the internal force of the motor. We would use ν to represent any random noise that we could not precisely predict like if the car slipped on the road or a strong wind moved the car.

For the Kalman filter lessons, we will assume that there is no way to measure or know the exact acceleration of a tracked object. For example, if we were in an autonomous vehicle tracking a bicycle, pedestrian or another car, we would not be able to model the internal forces of the other object; hence, we do not know for certain what the other object's acceleration is. Instead, we will set $Bu = 0$ and represent acceleration as a random noise with mean ν.

```
#include “Eigen/Dense”
using namespace Eigen;
```



Notes for using the Eigen Library:

You can create a vertical vector of two elements with a command like this:

```
VectorXd my_vector(2);
```

You can use the so called comma initializer to set all the coefficients to some values:

```
my_vector << 10, 20;
```

and you can use the cout command to print out the vector:

```
cout << my_vector << endl;
```

The matrices can be created in the same way. For example, This is an initialization of a 2 by 2 matrix with the values 1, 2, 3, and 4:

```
MatrixXd my_matrix(2,2);
my_matrix << 1, 2,
             3, 4;
```

You can use the same comma initializer or you can set each matrix value explicitly. For example, that's how we can change the matrix elements in the second row:

```
my_matrix(1,0) = 11;    //second row, first column
my_matrix(1,1) = 12;    //second row, second column
```

Also, you can compute the transpose of a matrix with the following command:

```
MatrixXd my_matrix_t = my_matrix.transpose();
```

And here is how you can get the matrix inverse:

```
MatrixXd my_matrix_i = my_matrix.inverse();
```

For multiplying the matrix m with the vector b you can write this in one line as let’s say matrix c equals m times v:

```
MatrixXd another_matrix;
another_matrix = my_matrix*my_vector;
```

QUESTION 2 OF 2

Why do we not use the process noise in the state prediction function, even though the state transition equation has one? In other words, why does the code set u << 0, 0 for the equation x = F * x + u?

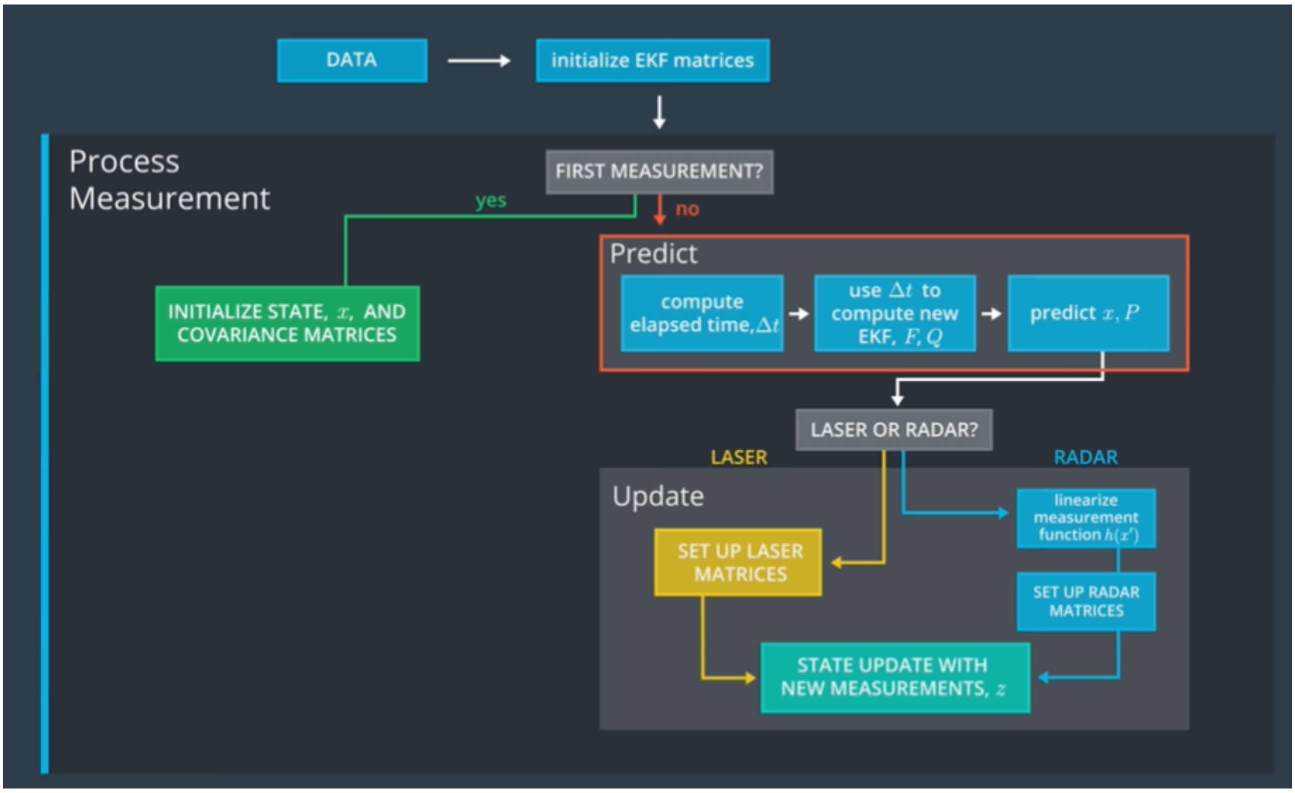
○ Because the process noise is just an approximation and we do not include the noise.

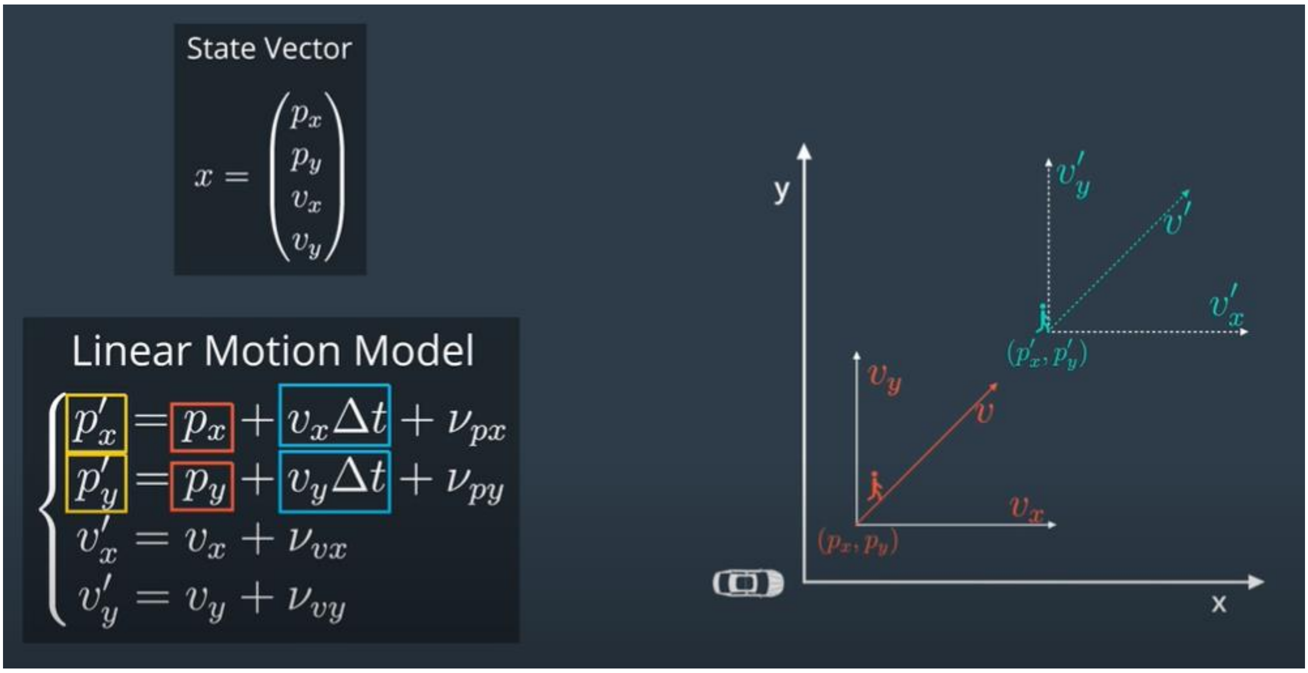
✓ The noise mean is zero.

○ We should! There's a bug in the function.

○ The noise mean is too large.

Looking closely at the process noise, we know from the Kalman Filter algorithm that its mean is zero and its covariance matrix is usually noted by $Q * N(0, Q)$. The first equation only predicts the mean state. As the mean value of the noise is zero, it does not directly affect the predicted state. However, we can see that the noise covariance Q is added here to the state covariance prediction so that the state uncertainty always increases through the process noise.


DATA
initialize EKF matrices
Process Measurement
FIRST MEASUREMENT?
yes
no
INITIALIZE STATE, x, AND COVARIANCE MATRICES
Predict
compute elapsed time, Δt
use Δt to compute new EKF, F, Q
predict x, P
LASER OR RADAR?
LASER
RADAR
Update
linearize measurement function $h(x')$
SET UP LASER MATRICES
SET UP RADAR MATRICES
STATE UPDATE WITH NEW MEASUREMENTS, z

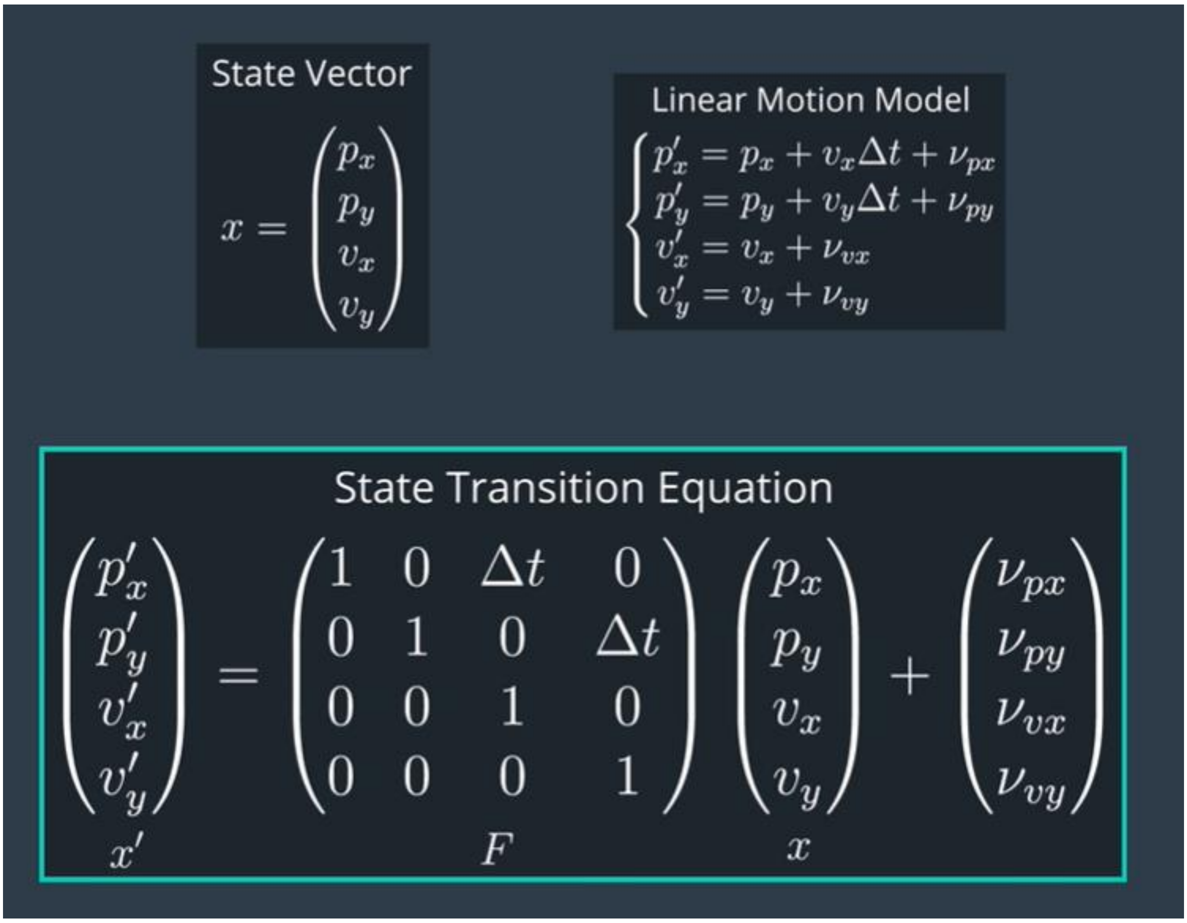


State Vector

$$x = \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix}$$

Linear Motion Model

$$\begin{cases} \boxed{p'_x} = \boxed{p_x} + \boxed{v_x \Delta t} + \nu_{px} \\ \boxed{p'_y} = \boxed{p_y} + \boxed{v_y \Delta t} + \nu_{py} \\ v'_x = v_x + \nu_{vx} \\ v'_y = v_y + \nu_{vy} \end{cases}$$



State Vector

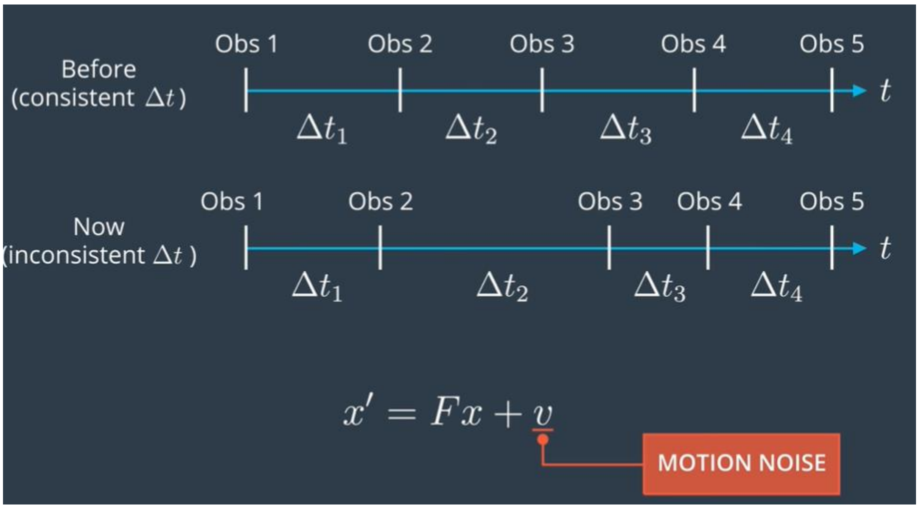
$$x = \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix}$$

Linear Motion Model

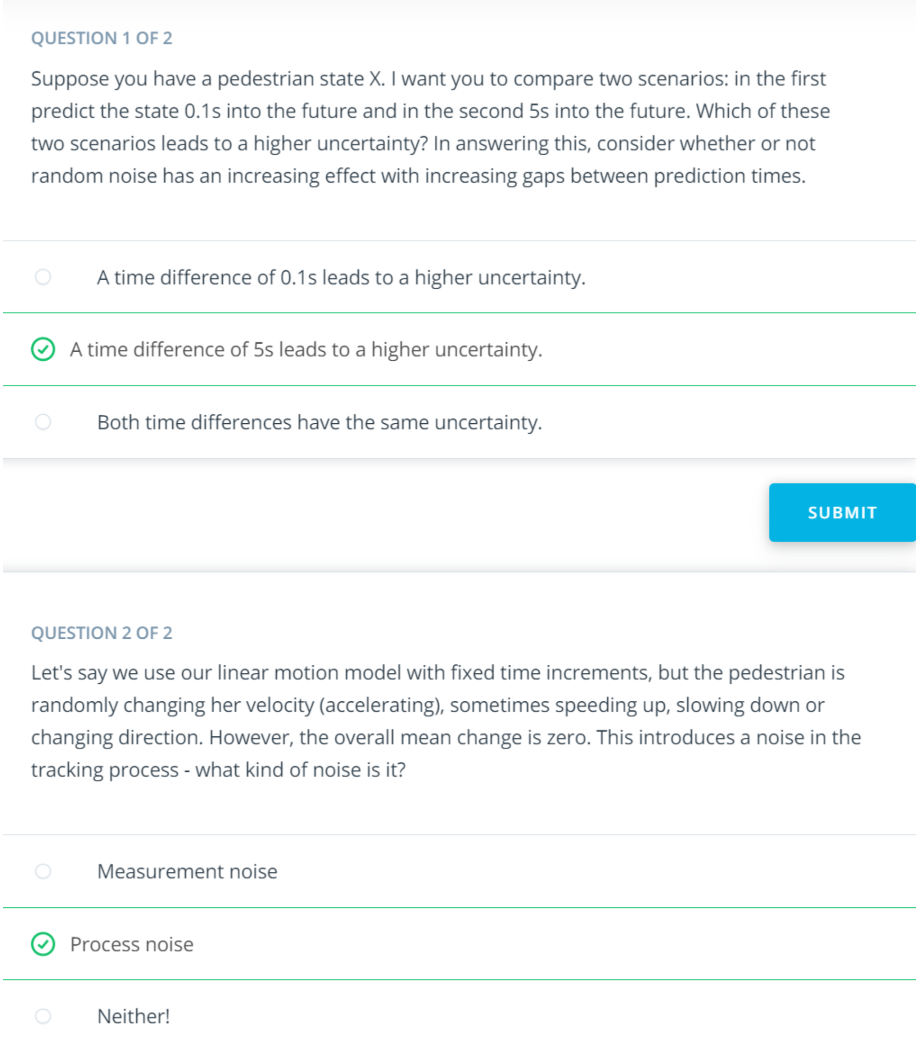
$$\begin{cases} p'_x = p_x + v_x \Delta t + \nu_{px} \\ p'_y = p_y + v_y \Delta t + \nu_{py} \\ v'_x = v_x + \nu_{vx} \\ v'_y = v_y + \nu_{vy} \end{cases}$$

State Transition Equation

$$\underbrace{\begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix}}_{x'} = \underbrace{\begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{F} \underbrace{\begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix}}_{x} + \begin{pmatrix} \nu_{px} \\ \nu_{py} \\ \nu_{vx} \\ \nu_{vy} \end{pmatrix}$$



$$x' = Fx + \underline{v}$$



QUESTION 1 OF 2

Suppose you have a pedestrian state X. I want you to compare two scenarios: in the first predict the state 0.1s into the future and in the second 5s into the future. Which of these two scenarios leads to a higher uncertainty? In answering this, consider whether or not random noise has an increasing effect with increasing gaps between prediction times.

○ A time difference of 0.1s leads to a higher uncertainty.

✓ A time difference of 5s leads to a higher uncertainty.

○ Both time differences have the same uncertainty.

QUESTION 2 OF 2

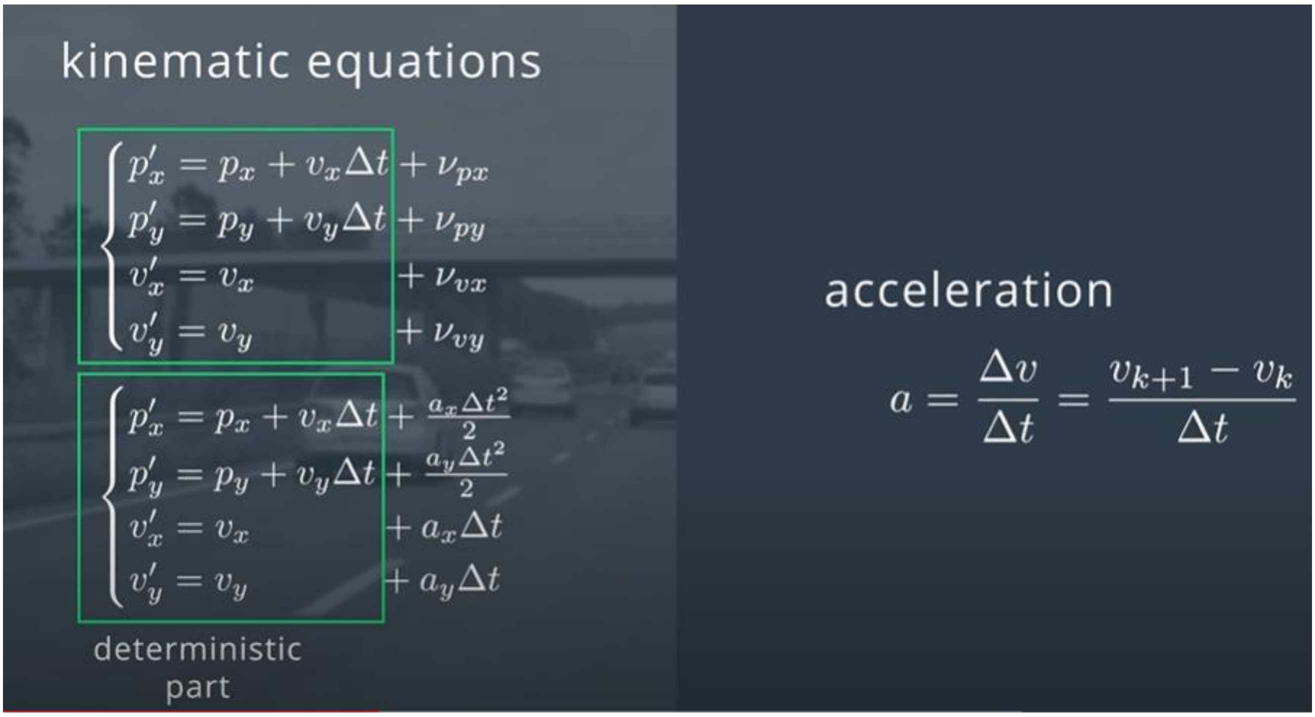
Let's say we use our linear motion model with fixed time increments, but the pedestrian is randomly changing her velocity (accelerating), sometimes speeding up, slowing down or changing direction. However, the overall mean change is zero. This introduces a noise in the tracking process - what kind of noise is it?

○ Measurement noise

✓ Process noise

○ Neither!

The prediction equation treats the pedestrian's velocity as constant. As such, a randomly accelerating pedestrian creates a process noise.



kinematic equations

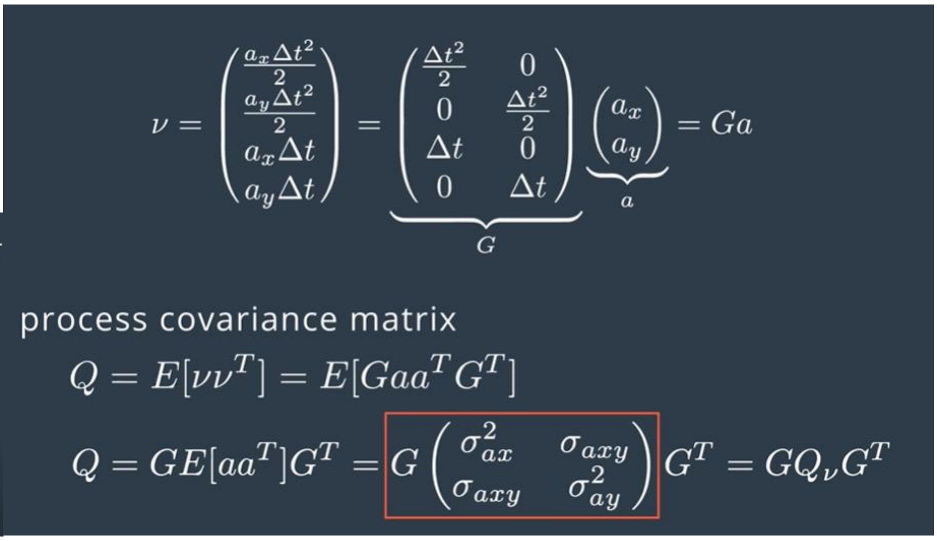
$$\begin{cases} p_x' = p_x + v_x\Delta t + \nu_{px} \\ p_y' = p_y + v_y\Delta t + \nu_{py} \\ v_x' = v_x + \nu_{vx} \\ v_y' = v_y + \nu_{vy} \end{cases}$$

$$\begin{cases} p_x' = p_x + v_x\Delta t + \frac{a_x\Delta t^2}{2} \\ p_y' = p_y + v_y\Delta t + \frac{a_y\Delta t^2}{2} \\ v_x' = v_x + a_x\Delta t \\ v_y' = v_y + a_y\Delta t \end{cases}$$

deterministic part

acceleration

$$a = \frac{\Delta v}{\Delta t} = \frac{v_{k+1} - v_k}{\Delta t}$$

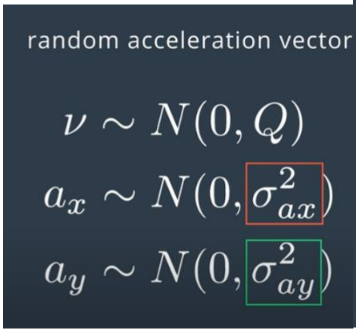


$$\nu = \begin{pmatrix} \frac{a_x\Delta t^2}{2} \\ \frac{a_y\Delta t^2}{2} \\ a_x\Delta t \\ a_y\Delta t \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix}}_{G} \underbrace{\begin{pmatrix} a_x \\ a_y \end{pmatrix}}_{a} = Ga$$

process covariance matrix

$$Q = E[\nu\nu^T] = E[Gaa^TG^T]$$

$$Q = GE[aa^T]G^T = \boxed{G\begin{pmatrix} \sigma_{ax}^2 & \sigma_{axy} \\ \sigma_{axy} & \sigma_{ay}^2 \end{pmatrix}}G^T = GQ_\nu G^T$$

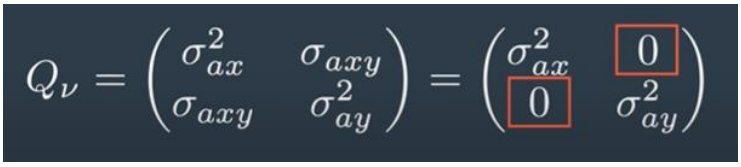


random acceleration vector

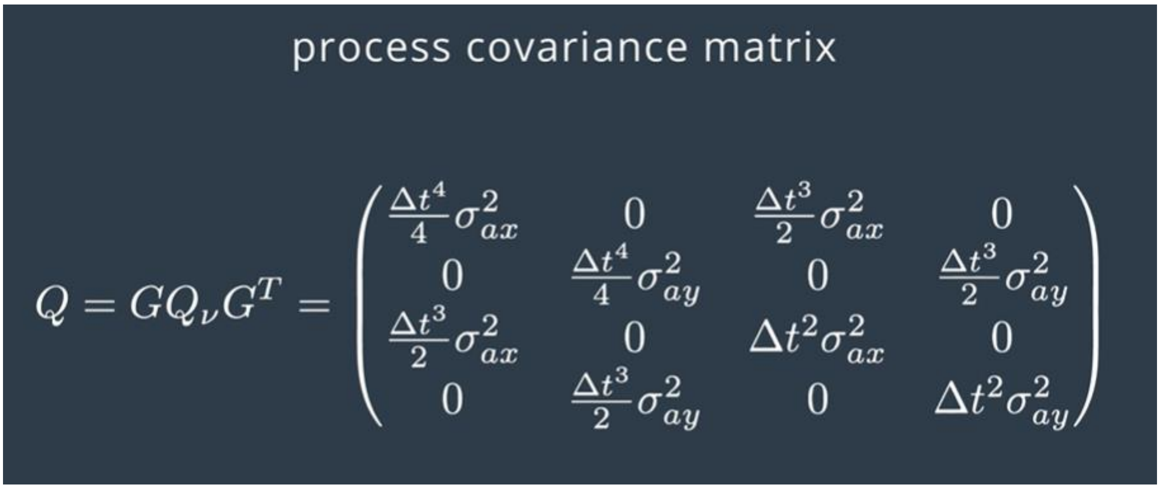
$$\nu \sim N(0, Q)$$

$$a_x \sim N(0, \boxed{\sigma_{ax}^2})$$

$$a_y \sim N(0, \boxed{\sigma_{ay}^2})$$

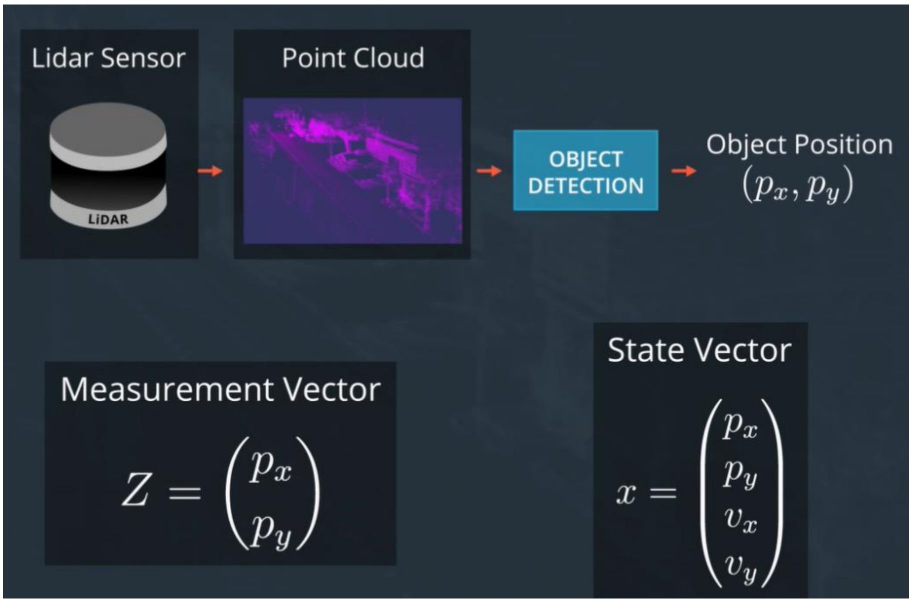


$$Q_\nu = \begin{pmatrix} \sigma_{ax}^2 & \sigma_{axy} \\ \sigma_{axy} & \sigma_{ay}^2 \end{pmatrix} = \begin{pmatrix} \sigma_{ax}^2 & \boxed{0} \\ \boxed{0} & \sigma_{ay}^2 \end{pmatrix}$$



process covariance matrix

$$Q = GQ_\nu G^T = \begin{pmatrix} \frac{\Delta t^4}{4}\sigma_{ax}^2 & 0 & \frac{\Delta t^3}{2}\sigma_{ax}^2 & 0 \\ 0 & \frac{\Delta t^4}{4}\sigma_{ay}^2 & 0 & \frac{\Delta t^3}{2}\sigma_{ay}^2 \\ \frac{\Delta t^3}{2}\sigma_{ax}^2 & 0 & \Delta t^2\sigma_{ax}^2 & 0 \\ 0 & \frac{\Delta t^3}{2}\sigma_{ay}^2 & 0 & \Delta t^2\sigma_{ay}^2 \end{pmatrix}$$



Lidar Sensor

Point Cloud

OBJECT DETECTION

Object Position (p_x, p_y)

State Vector

$$x = \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix}$$

Measurement Vector

$$Z = \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$

Variable Definitions

To reinforce was what discussed in the video, here is an explanation of what each variable represents:

- **z** is the measurement vector. For a lidar sensor, the z vector contains the $position - x$ and $position - y$ measurements.
- **H** is the matrix that projects your belief about the object's current state into the measurement space of the sensor. For lidar, this is a fancy way of saying that we discard velocity information from the state variable since the lidar sensor only measures position: The state vector x contains information about $[p_x, p_y, v_x, v_y]$ whereas the z vector will only contain $[px, py]$. Multiplying Hx allows us to compare x, our belief, with z, the sensor measurement.
- What does the prime notation in the x vector represent? The prime notation like p'_x means you have already done the prediction step but have not done the measurement step yet. In other words, the object was at p_x. After time Δt, you calculate where you believe the object will be based on the motion model and get p'_x.

H Matrix Quiz

Find the right H matrix to project from a 4D state to a 2D observation space, as follows:

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = H \begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix}$$

Here are your options:

A. $H = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

B. $H = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$

C. $H = \begin{pmatrix} 1 & 1 \end{pmatrix}$

D. $H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$

(Hint: first consider the matrix dimensions, then try to use a 0 or 1 to correctly project the components into the measurement space.)

QUIZ QUESTION

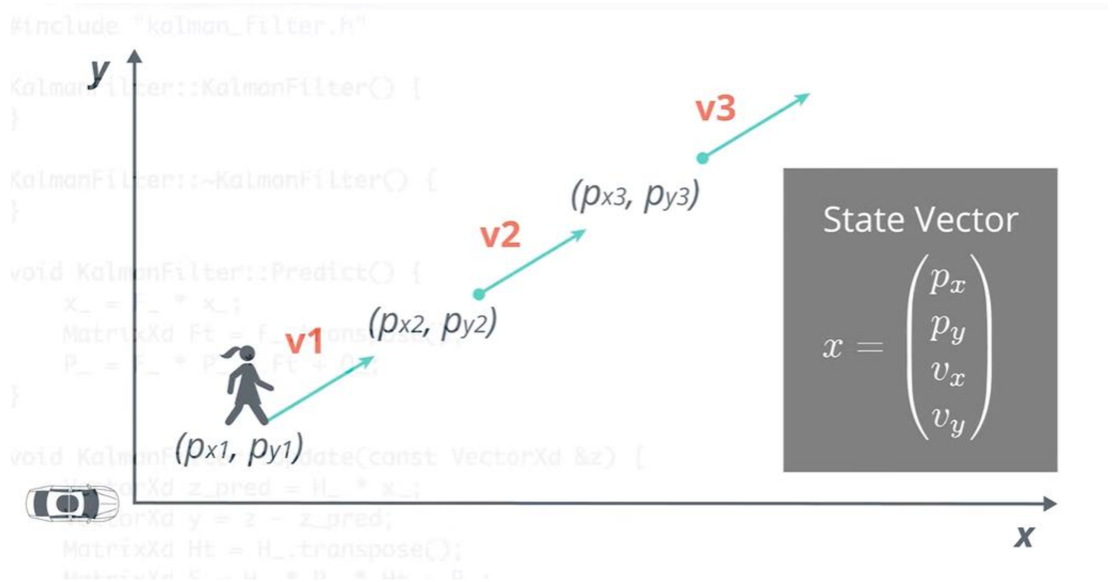
Select the correct H matrix given the projection above.

○ A

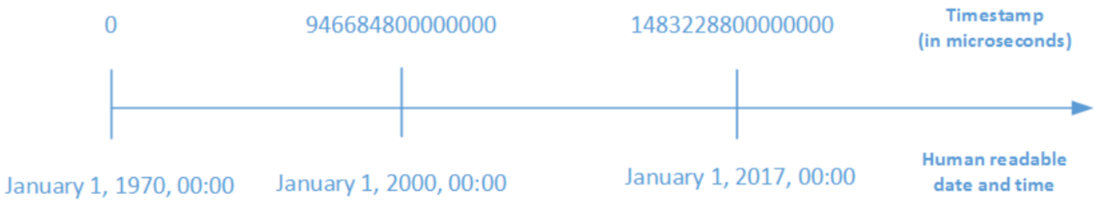
○ B

○ C

✓ D


y
x
v1
v2
v3
(px1, py1)
(px2, py2)
(px3, py3)
State Vector
$x = \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix}$

More Info on Timestamps



Time keeps on tickin', tickin', tickin'...

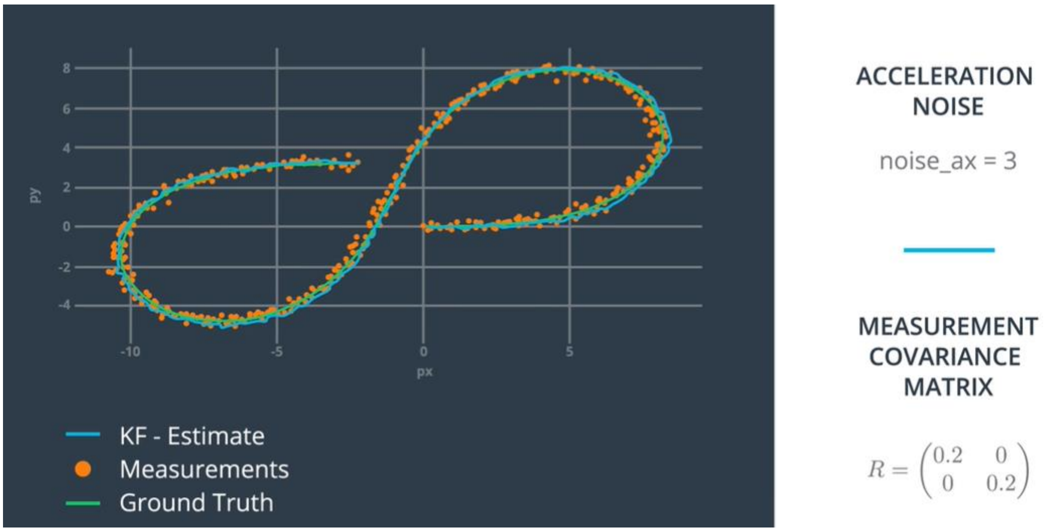
Timestamps are often used for logging a sequence of events, so that we know exactly, for example, in our case when the measurements were generated.

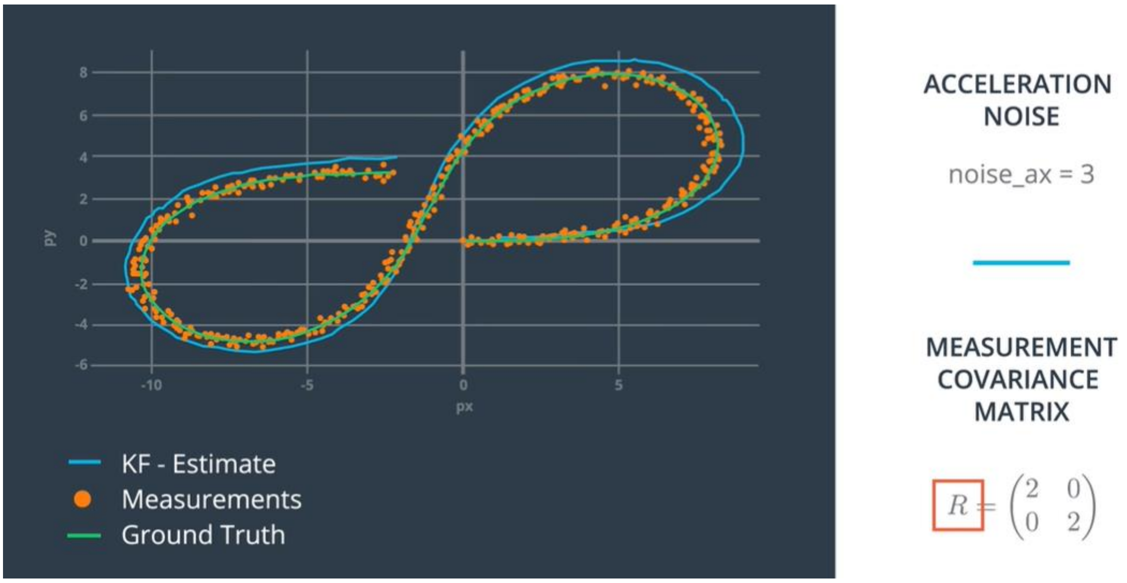
We can use the timestamp values to compute the elapsed time between two consecutive observations as:

```
float delta_t = ( timestamp(k+1) - timestamp(k) ) / 1000000.0.
```

Additionally we divide the result by 10^6 to transform it from microseconds to seconds.

```
float dt = (measurement_pack.timestamp_ - previous_timestamp_) / 1000000.0;
```



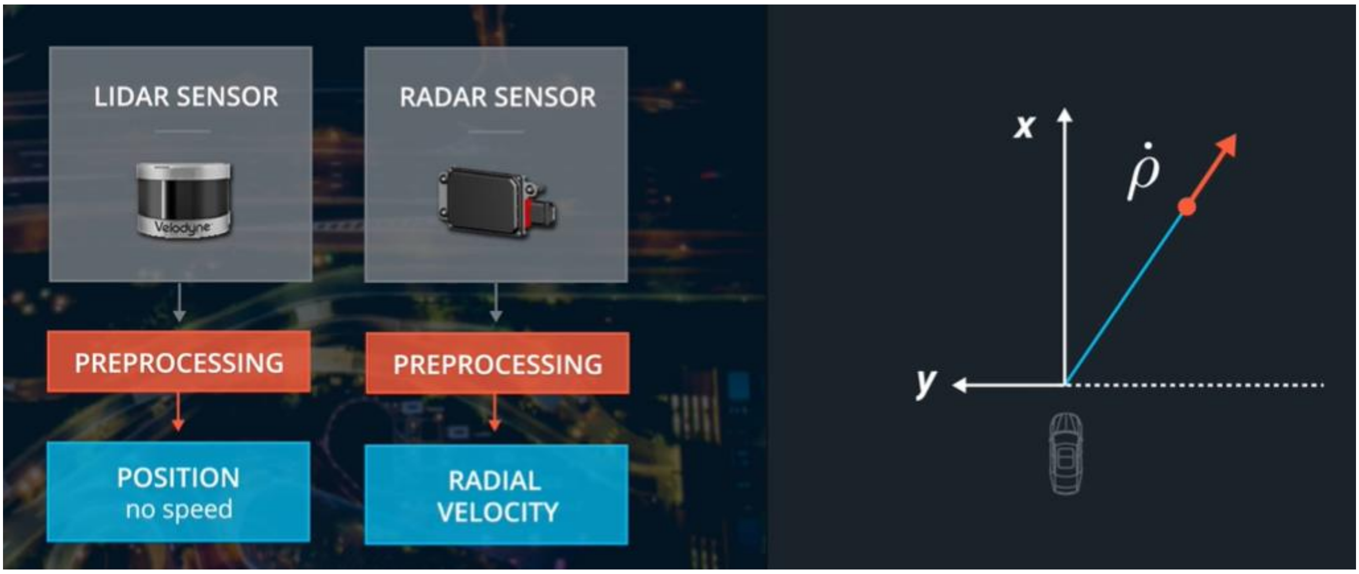


ACCELERATION NOISE

noise_ax = 3

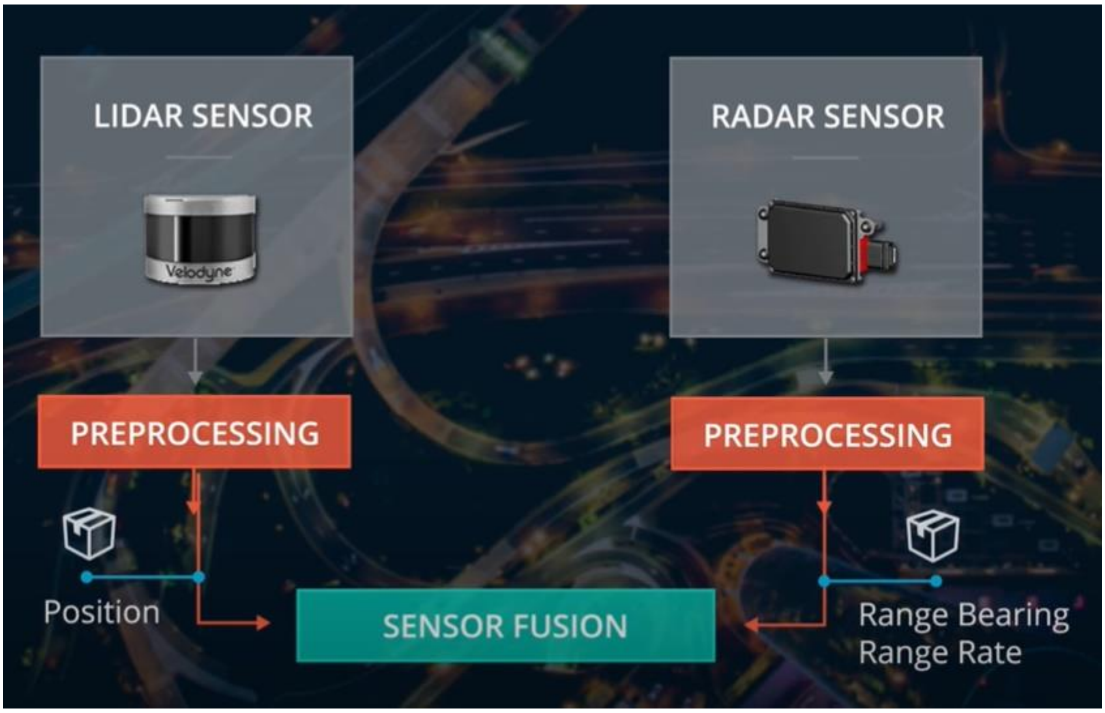
MEASUREMENT COVARIANCE MATRIX

$$\boxed{R} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$



LIDAR SENSOR → PREPROCESSING → POSITION no speed

RADAR SENSOR → PREPROCESSING → RADIAL VELOCITY



LIDAR SENSOR → PREPROCESSING → Position → SENSOR FUSION

RADAR SENSOR → PREPROCESSING → Range Bearing Range Rate → SENSOR FUSION

STATE TRANSITION FUNCTION

$$x' = f(x) + \nu$$

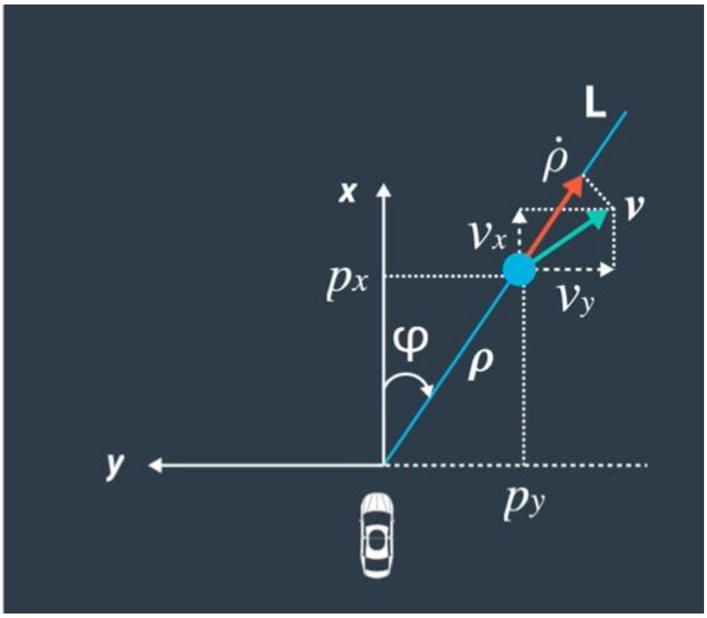
STATE VECTOR

$$x = \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix} \qquad z = \underbrace{h(x')}_{\text{LINEAR MOTION}} + \underbrace{\omega}_{\text{PROCESS NOISE}}$$

RANGE: *ρ* (rho)
radial distance from origin

BEARING: φ (phi)
angle between ***ρ*** and ***x***

RADIAL VELOCITY: $\dot{\rho}$ (rho dot)
change of ***ρ*** (range rate)



MEASUREMENT FUNCTION

$$z = h(x') + \underbrace{\omega}_{\text{NOISE}}$$

MEASUREMENT VECTOR

$$z = \begin{pmatrix} \rho \\ \varphi \\ \dot{\rho} \end{pmatrix}$$

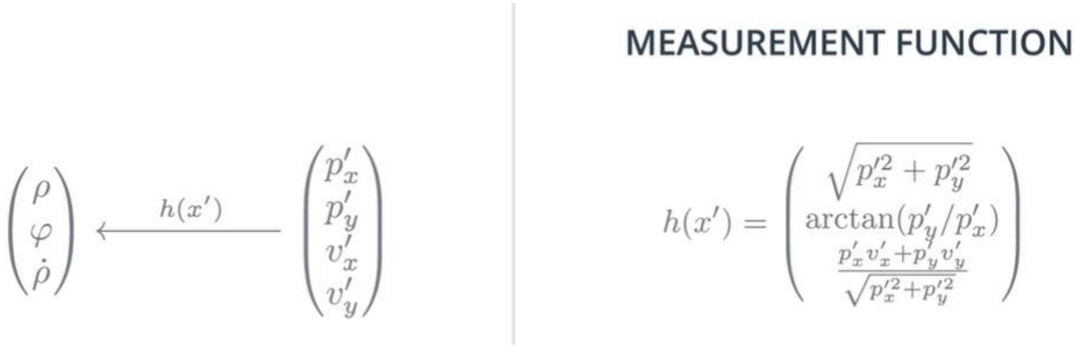
$$\omega \sim N(0, R)$$

RADAR MEASUREMENT COVARIANCE MATRIX

$$R = \begin{pmatrix} \sigma_\rho^2 & 0 & 0 \\ 0 & \sigma_\varphi^2 & 0 \\ 0 & 0 & \sigma_{\dot{\rho}}^2 \end{pmatrix}$$

MEASUREMENT FUNCTION

$$\begin{pmatrix} \rho \\ \varphi \\ \dot{\rho} \end{pmatrix} \xleftarrow{\boxed{h(x')}} \begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix}$$



MEASUREMENT FUNCTION

$$\begin{pmatrix} \rho \\ \varphi \\ \dot{\rho} \end{pmatrix} \xleftarrow{h(x')} \begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix}$$

$$h(x') = \begin{pmatrix} \sqrt{p'^2_x + p'^2_y} \\ \arctan(p'_y/p'_x) \\ \frac{p'_x v'_x + p'_y v'_y}{\sqrt{p'^2_x + p'^2_y}} \end{pmatrix}$$

H versus h(x)

The H matrix from the lidar lesson and $h(x)$ equations from the radar lesson are actually accomplishing the same thing; they are both needed to solve $y = z - Hx'$ in the update step.

But for radar, there is no H matrix that will map the state vector x into polar coordinates; instead, you need to calculate the mapping manually to convert from cartesian coordinates to polar coordinates.

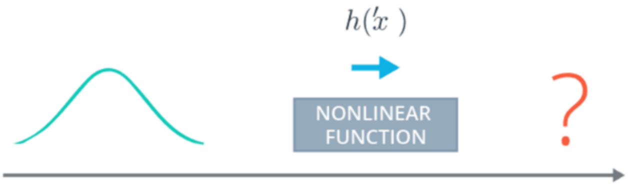
Here is the h function that specifies how the predicted position and speed get mapped to the polar coordinates of range, bearing and range rate.

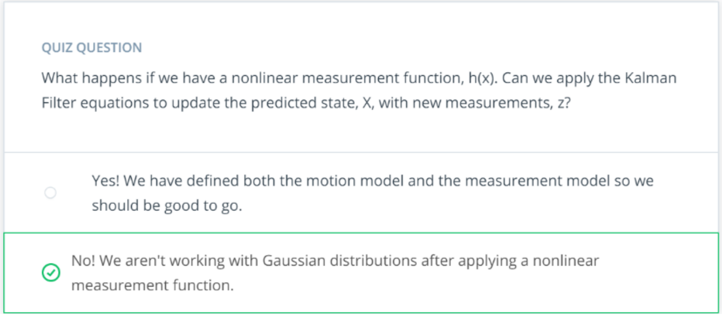
$$h(x') = \begin{pmatrix} \rho \\ \phi \\ \dot{\rho} \end{pmatrix} = \begin{pmatrix} \sqrt{{p'}_x^2 + {p'}_y^2} \\ \arctan(p'_y/p'_x) \\ \frac{p'_x v'_x + p'_y v'_y}{\sqrt{{p'}_x^2 + {p'}_y^2}} \end{pmatrix}$$

Hence for radar $y = z - Hx'$ becomes $y = z - h(x')$.

Definition of Radar Variables

- The range, (ρ), is the distance to the pedestrian. The range is basically the magnitude of the position vector ρ which can be defined as $\rho = sqrt(p_x^2 + p_y^2)$.
- $\varphi = atan(p_y/p_x)$. Note that φ is referenced counter-clockwise from the x-axis, so φ from the video clip above in that situation would actually be negative.
- The range rate, $\dot{\rho}$, is the projection of the velocity, v, onto the line, L.



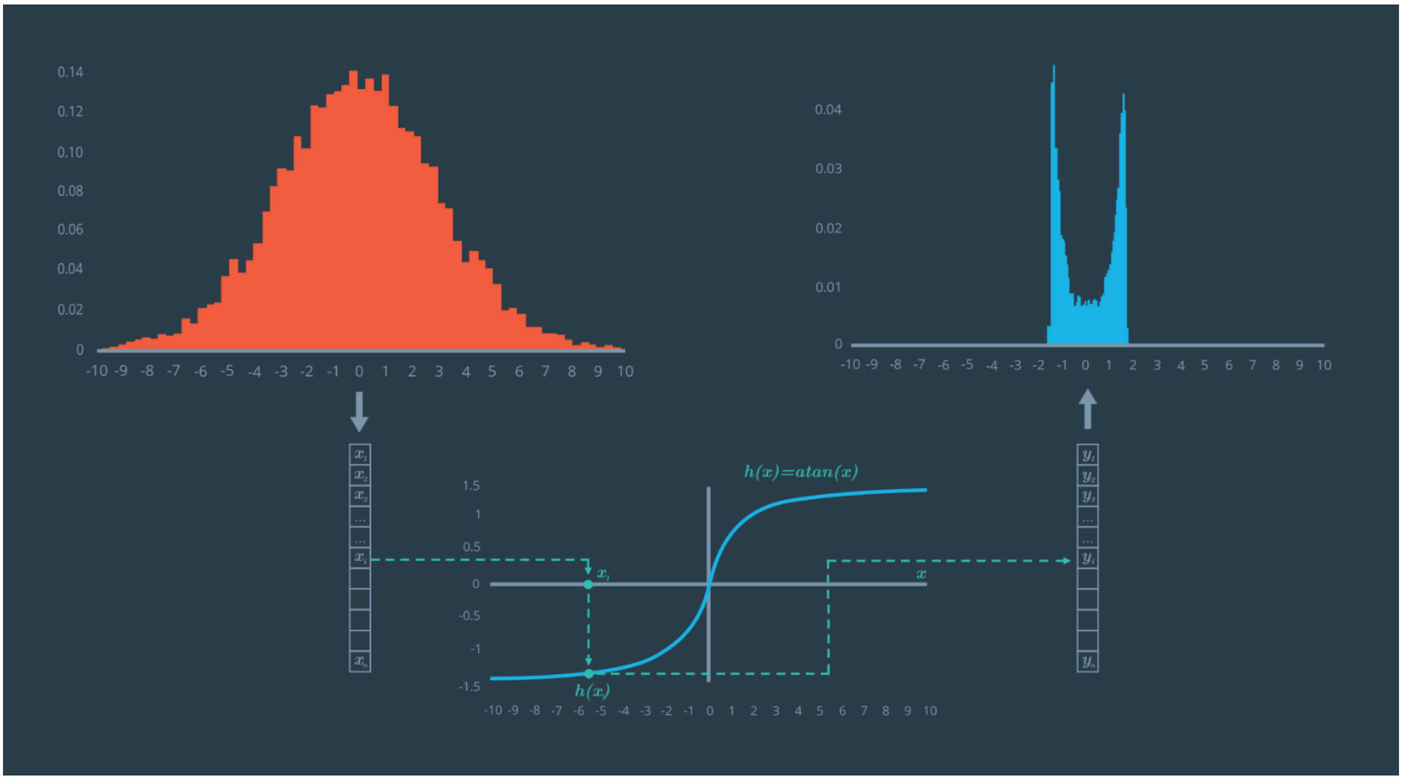


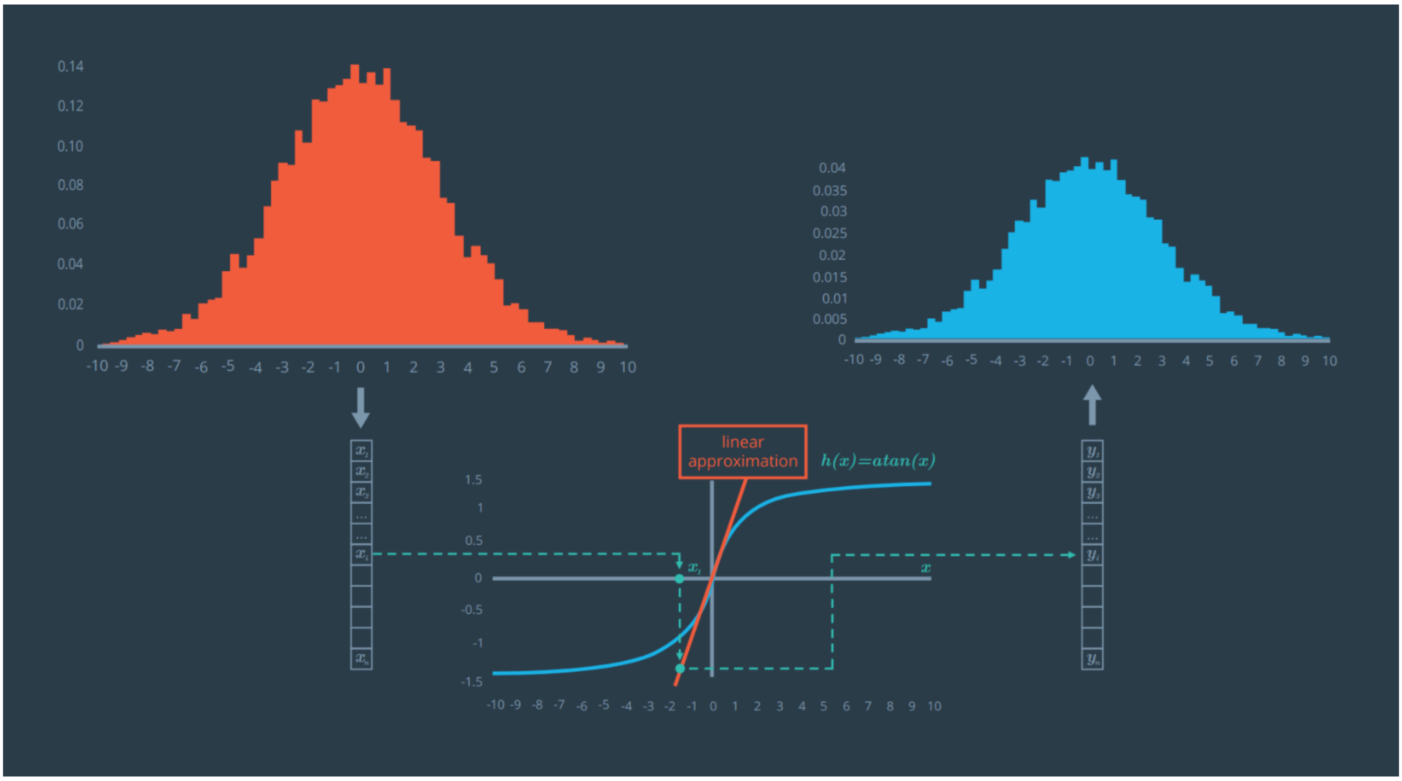
QUIZ QUESTION

What happens if we have a nonlinear measurement function, h(x). Can we apply the Kalman Filter equations to update the predicted state, X, with new measurements, z?

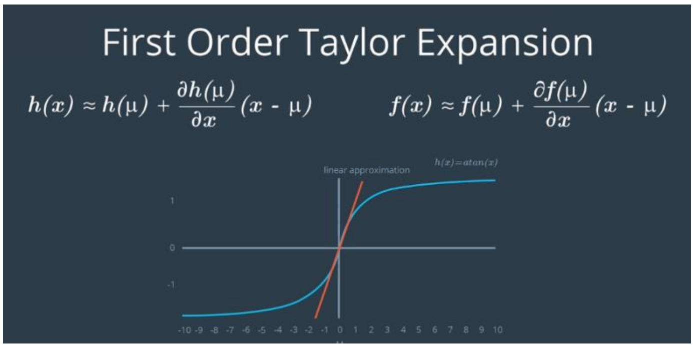
○ Yes! We have defined both the motion model and the measurement model so we should be good to go.

✓ No! We aren't working with Gaussian distributions after applying a nonlinear measurement function.



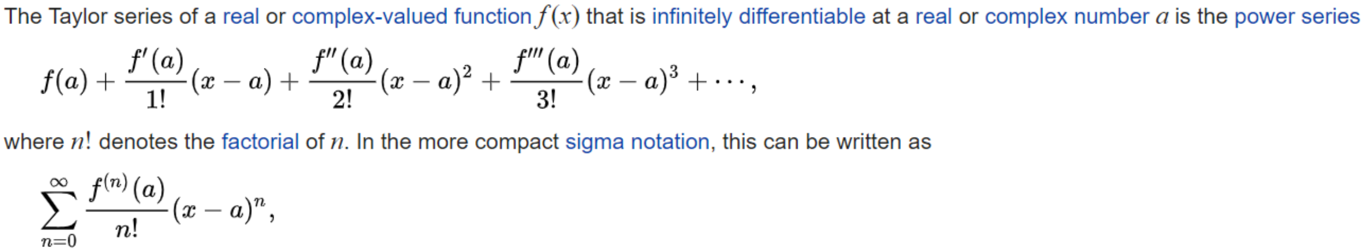


This one looks much better! Notice how the blue graph, the output, remains a Gaussian after applying a first order Taylor expansion.



First Order Taylor Expansion

$h(x) \approx h(\mu) + \frac{\partial h(\mu)}{\partial x}(x - \mu)$ $\qquad$ $f(x) \approx f(\mu) + \frac{\partial f(\mu)}{\partial x}(x - \mu)$

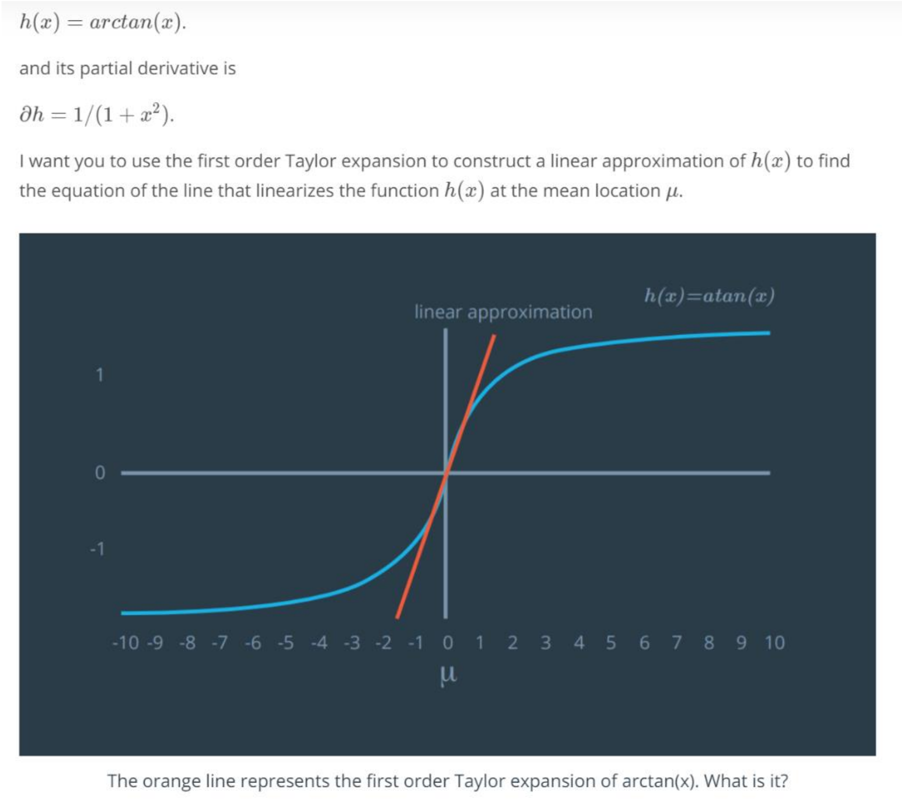


The Taylor series of a real or complex-valued function $f(x)$ that is infinitely differentiable at a real or complex number a is the power series

$$f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\cdots,$$

where $n!$ denotes the factorial of n. In the more compact sigma notation, this can be written as

$$\sum_{n=0}^{\infty}\frac{f^{(n)}(a)}{n!}(x-a)^n,$$



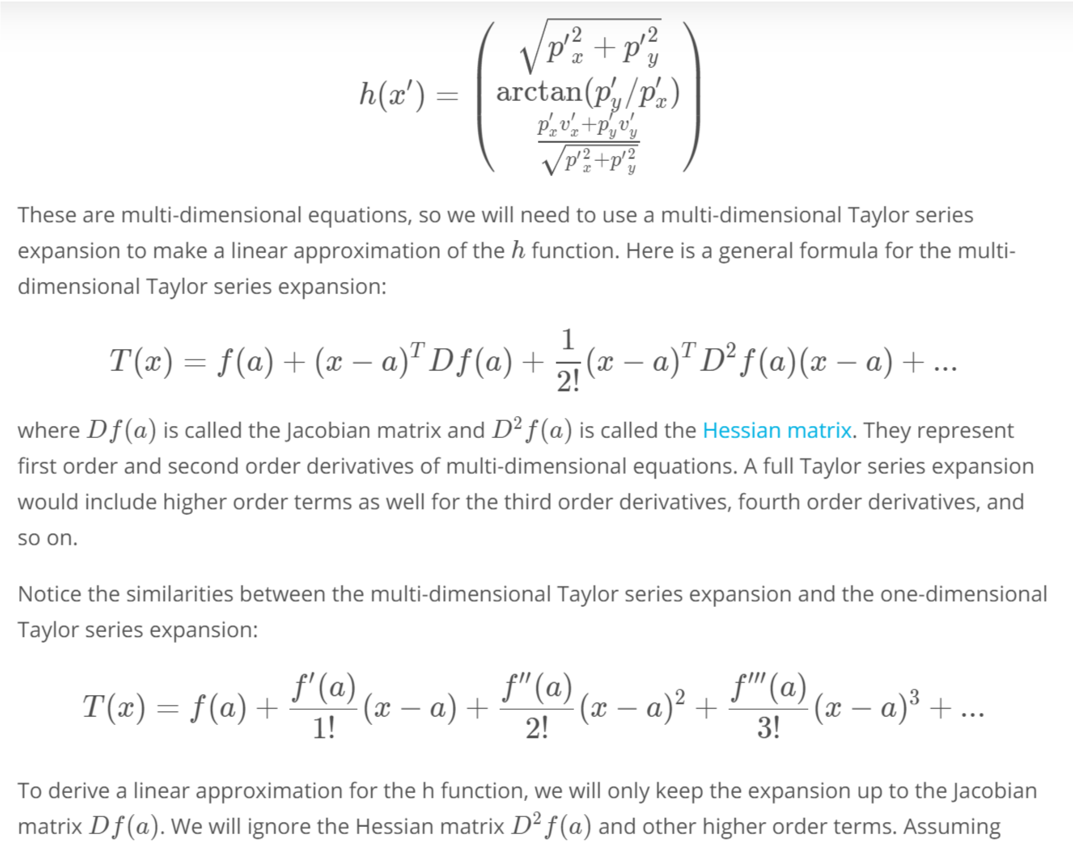
$h(x) = arctan(x).$

and its partial derivative is

$\partial h = 1/(1+x^2).$

I want you to use the first order Taylor expansion to construct a linear approximation of $h(x)$ to find the equation of the line that linearizes the function $h(x)$ at the mean location μ.

The orange line represents the first order Taylor expansion of arctan(x). What is it?



$$h(x') = \begin{pmatrix} \sqrt{p'^2_x + p'^2_y} \\ \arctan(p'_y/p'_x) \\ \frac{p'_x v'_x + p'_y v'_y}{\sqrt{p'^2_x + p'^2_y}} \end{pmatrix}$$

These are multi-dimensional equations, so we will need to use a multi-dimensional Taylor series expansion to make a linear approximation of the h function. Here is a general formula for the multi-dimensional Taylor series expansion:

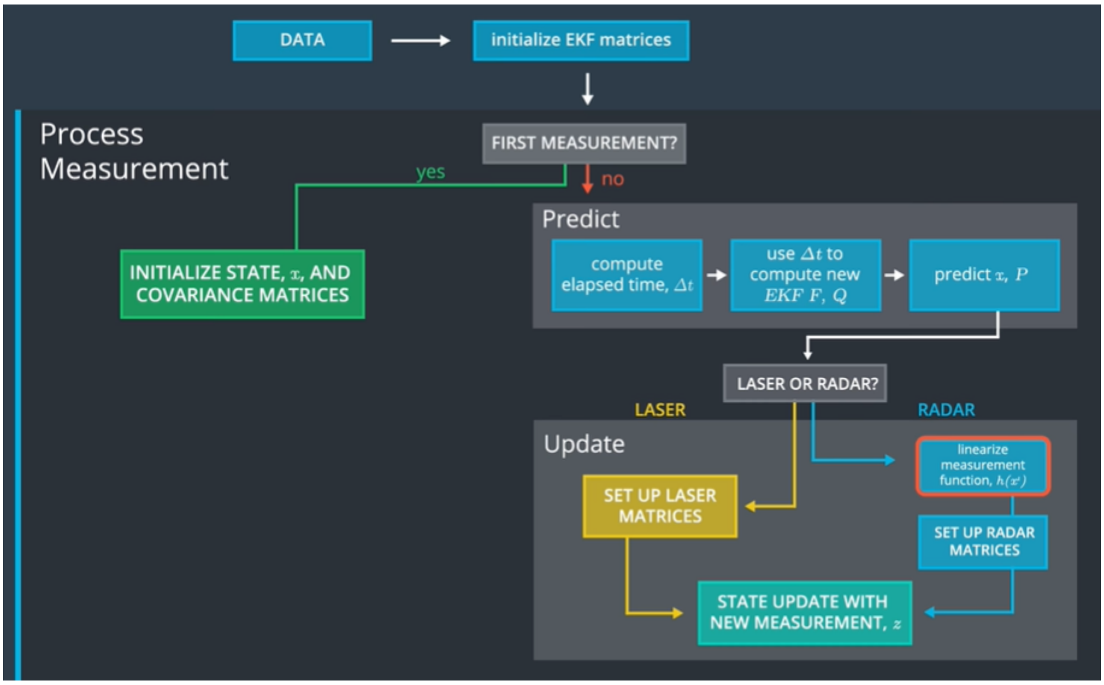
$$T(x) = f(a) + (x-a)^T Df(a) + \frac{1}{2!}(x-a)^T D^2 f(a)(x-a) + \ldots$$

where $Df(a)$ is called the Jacobian matrix and $D^2 f(a)$ is called the Hessian matrix. They represent first order and second order derivatives of multi-dimensional equations. A full Taylor series expansion would include higher order terms as well for the third order derivatives, fourth order derivatives, and so on.

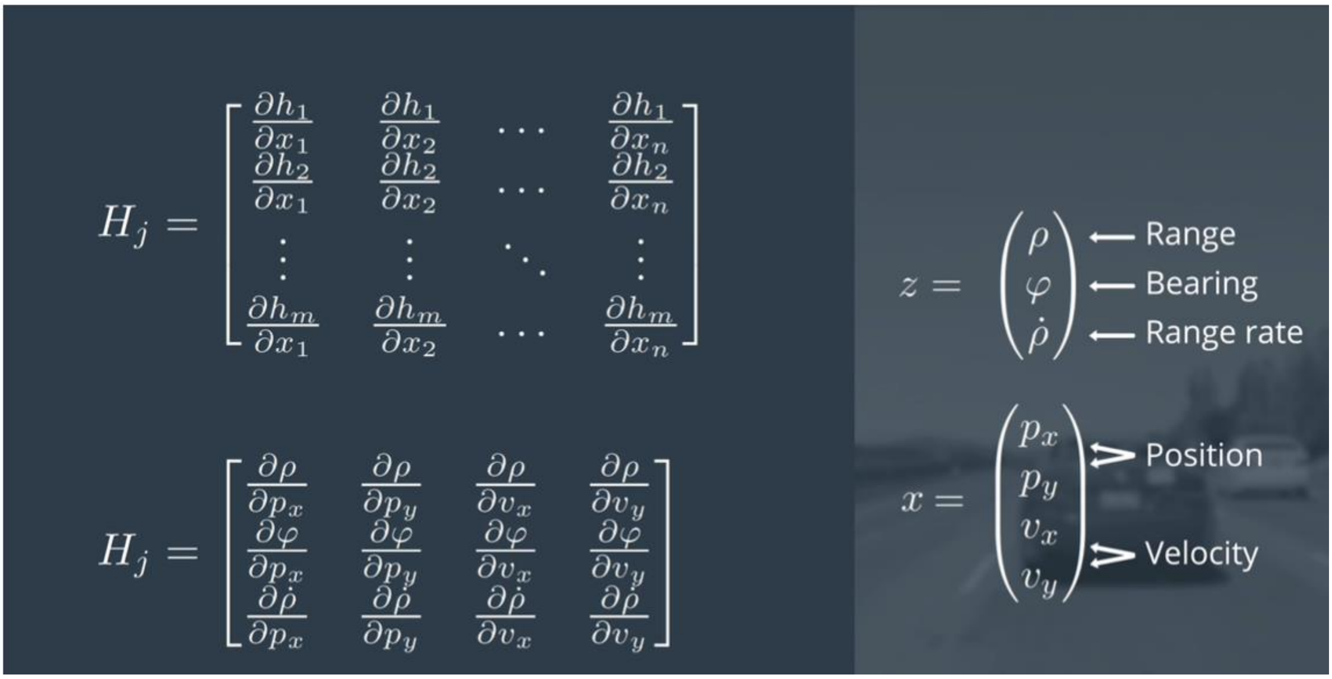
Notice the similarities between the multi-dimensional Taylor series expansion and the one-dimensional Taylor series expansion:

$$T(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \ldots$$

To derive a linear approximation for the h function, we will only keep the expansion up to the Jacobian matrix $Df(a)$. We will ignore the Hessian matrix $D^2 f(a)$ and other higher order terms. Assuming


DATA
initialize EKF matrices
Process Measurement
FIRST MEASUREMENT?
yes
no
INITIALIZE STATE, x, AND COVARIANCE MATRICES
Predict
compute elapsed time, Δt
use Δt to compute new EKF F, Q
predict x, P
LASER OR RADAR?
LASER
RADAR
Update
linearize measurement function, $h(x')$
SET UP LASER MATRICES
SET UP RADAR MATRICES
STATE UPDATE WITH NEW MEASUREMENT, z

Jacobian Matrix Part 1



$$H_j = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial x_1} & \frac{\partial h_m}{\partial x_2} & \cdots & \frac{\partial h_m}{\partial x_n} \end{bmatrix}$$

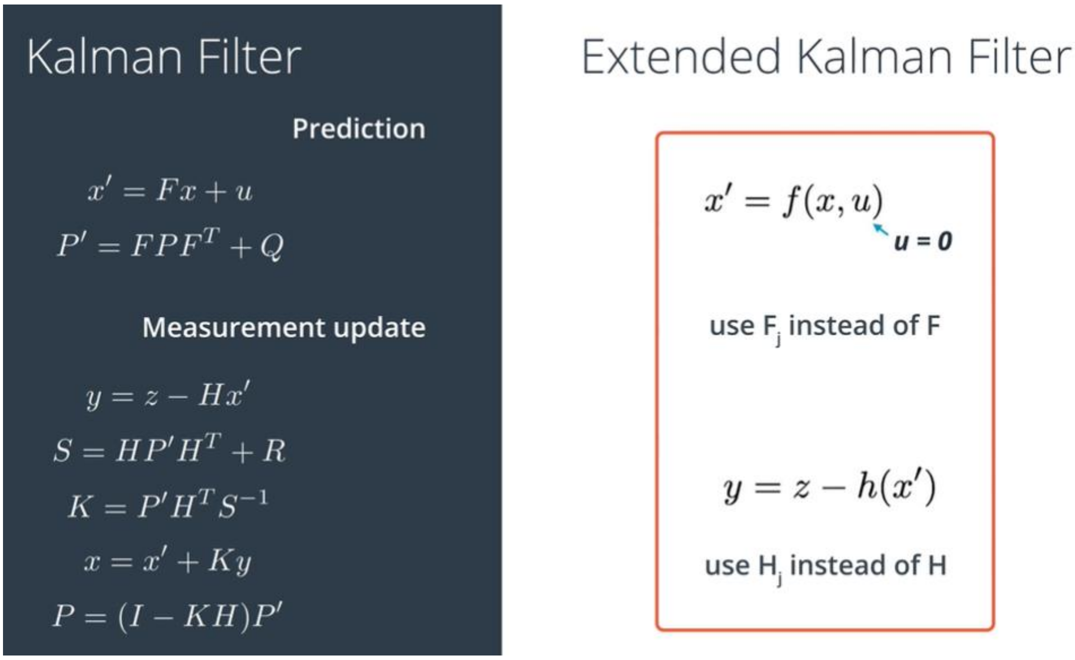
$$H_j = \begin{bmatrix} \frac{\partial \rho}{\partial p_x} & \frac{\partial \rho}{\partial p_y} & \frac{\partial \rho}{\partial v_x} & \frac{\partial \rho}{\partial v_y} \\ \frac{\partial \varphi}{\partial p_x} & \frac{\partial \varphi}{\partial p_y} & \frac{\partial \varphi}{\partial v_x} & \frac{\partial \varphi}{\partial v_y} \\ \frac{\partial \dot{\rho}}{\partial p_x} & \frac{\partial \dot{\rho}}{\partial p_y} & \frac{\partial \dot{\rho}}{\partial v_x} & \frac{\partial \dot{\rho}}{\partial v_y} \end{bmatrix}$$

$$z = \begin{pmatrix} \rho \\ \varphi \\ \dot{\rho} \end{pmatrix} \begin{array}{l} \longleftarrow \text{Range} \\ \longleftarrow \text{Bearing} \\ \longleftarrow \text{Range rate} \end{array}$$

$$x = \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix} \begin{array}{l} \left.\begin{array}{l} \\ \end{array}\right\rangle \text{Position} \\ \left.\begin{array}{l} \\ \end{array}\right\rangle \text{Velocity} \end{array}$$

Jacobian

$$H_j = \begin{bmatrix} \frac{p_x}{\sqrt{p_x^2+p_y^2}} & \frac{p_y}{\sqrt{p_x^2+p_y^2}} & 0 & 0 \\ -\frac{p_y}{p_x^2+p_y^2} & \frac{p_x}{p_x^2+p_y^2} & 0 & 0 \\ \frac{p_y(v_x p_y - v_y p_x)}{(p_x^2+p_y^2)^{3/2}} & \frac{p_x(v_y p_x - v_x p_y)}{(p_x^2+p_y^2)^{3/2}} & \frac{p_x}{\sqrt{p_x^2+p_y^2}} & \frac{p_y}{\sqrt{p_x^2+p_y^2}} \end{bmatrix}$$



Kalman Filter

Prediction

$$x' = Fx + u$$

$$P' = FPF^T + Q$$

Measurement update

$$y = z - Hx'$$

$$S = HP'H^T + R$$

$$K = P'H^TS^{-1}$$

$$x = x' + Ky$$

$$P = (I - KH)P'$$

Extended Kalman Filter

$$x' = f(x, \overset{u = 0}{u})$$

use F_j instead of F

$$y = z - h(x')$$

use H_j instead of H

Extended Kalman Filter Equations

Although the mathematical proof is somewhat complex, it turns out that the Kalman filter equations and extended Kalman filter equations are very similar. The main differences are:

- the F matrix will be replaced by F_j when calculating P'.
- the H matrix in the Kalman filter will be replaced by the Jacobian matrix H_j when calculating S, K, and P.
- to calculate x', the prediction update function, f, is used instead of the F matrix.
- to calculate y, the h function is used instead of the H matrix.

For this project, however, we do not need to use the f function or F_j. If we had been using a non-linear model in the prediction step, we would need to replace the F matrix with its Jacobian, F_j. However, we are using a linear model for the prediction step. So, for the prediction step, we can still use the regular Kalman filter equations and the F matrix rather than the extended Kalman filter equations.

The measurement update for lidar will also use the regular Kalman filter equations, since lidar uses linear equations. Only the measurement update for the radar sensor will use the extended Kalman filter equations.

One important point to reiterate is that the equation $y = z - Hx'$ for the Kalman filter does not become $y = z - H_jx$ for the extended Kalman filter. Instead, for extended Kalman filters, we'll use the h function directly to map predicted locations x' from Cartesian to polar coordinates.

Clarification of u = 0

In the above image, the prediction equation is written as $x' = Fx + u$ and $x' = f(x, u)$. Previously the equation was written $x' = Fx + \nu$.

It is just a question of notation where ν is the greek letter "nu" and "u" is used in the code examples. Remember that ν is represented by a gaussian distribution with mean zero. The equation $x' = Fx + u$ or the equivalent equation $x' = Fx + \nu$ calculates the mean value of the state variable x; hence we set u = 0. The uncertainty in the gaussian distribution shows up in the Q matrix.

More Details About Calculations with Radar Versus Lidar

In the radar update step, the Jacobian matrix H_j is used to calculate S, K and P. To calculate y, we use the equations that map the predicted location x' from Cartesian coordinates to polar coordinates:

$$h(x') = \begin{pmatrix} \sqrt{p'^2_x + p'^2_y} \\ \arctan(p'_y / p'_x) \\ \frac{p'_x v'_x + p'_y v'_y}{\sqrt{p'^2_x + p'^2_y}} \end{pmatrix}$$

The predicted measurement vector x' is a vector containing values in the form $\begin{bmatrix} p_x, p_y, v_x, v_y \end{bmatrix}$. The radar sensor will output values in polar coordinates:

$$\begin{pmatrix} \rho \\ \phi \\ \dot{\rho} \end{pmatrix}$$

In order to calculate y for the radar sensor, we need to convert x' to polar coordinates. In other words, the function $h(x)$ maps values from Cartesian coordinates to polar coordinates. So the equation for radar becomes $y = z_{radar} - h(x')$.

One other important point when calculating y with radar sensor data: the second value in the polar coordinate vector is the angle ϕ. You'll need to make sure to normalize ϕ in the y vector so that its angle is between $-\pi$ and π; in other words, add or subtract 2π from ϕ until it is between $-\pi$ and π.

To summarize:

- for measurement updates with lidar, we can use the H matrix for calculating y, S, K and P.
- for radar, H_j is used to calculate S, K and P.

QUIZ QUESTION

Compared to Kalman Filters, how would the Extended Kalman Filter result differ when the prediction function and measurement function are both linear?

✓ The Extended Kalman Filter's result would be the same as the standard Kalman Filter's result.

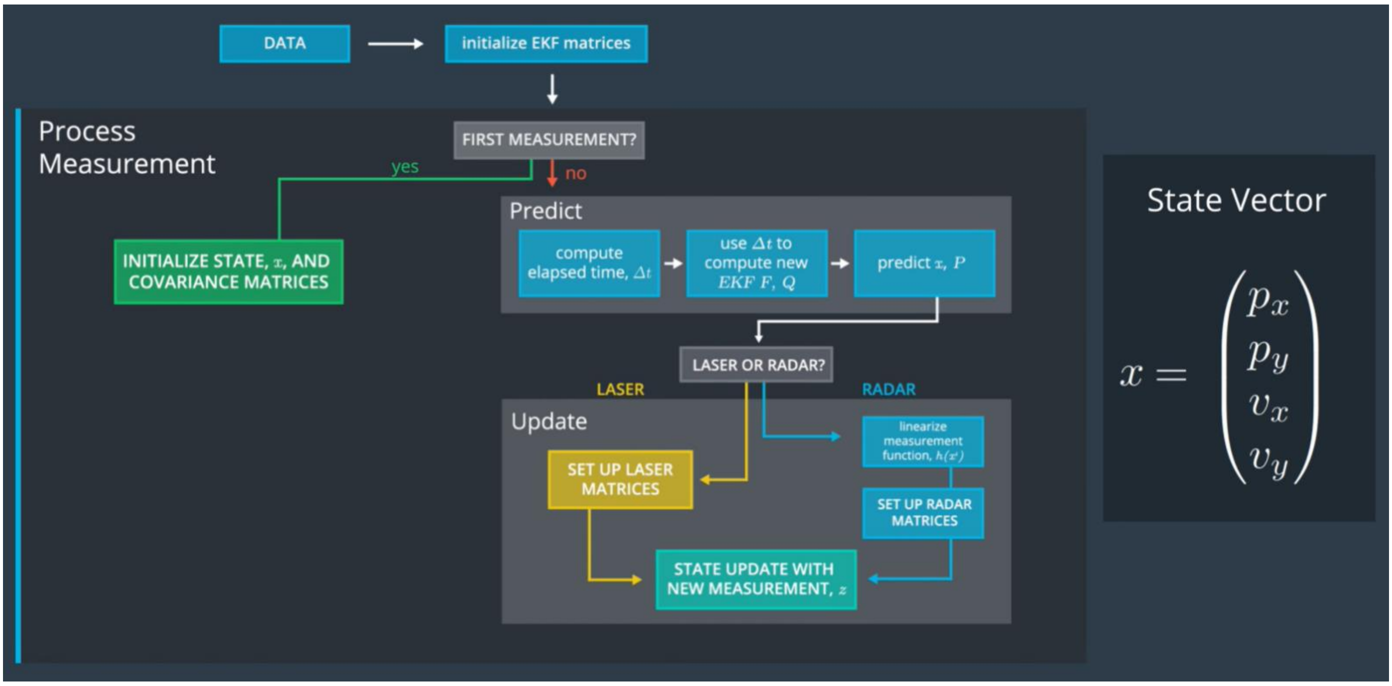
○ The Extended Kalman Filter's result would vary unpredictably compared to the Kalman Filter's result.

○ The Extended Kalman Filter's result would be less accurate than the Kalman Filter's result.

○ The Extended Kalman Filter's result would be more accurate than the Kalman Filter's result.

If f and h are linear functions, then the Extended Kalman Filter generates exactly the same result as the standard Kalman Filter. Actually, if f and h are linear then the Extended Kalman Filter F_j turns into f and H_j turns into h. All that's left is the same ol' standard Kalman Filter!

In our case we have a linear motion model, but a nonlinear measurement model when we use radar observations. So, we have to compute the Jacobian only for the measurement function.



DATA

initialize EKF matrices

Process Measurement

FIRST MEASUREMENT?

yes

no

INITIALIZE STATE, x, AND COVARIANCE MATRICES

Predict

compute elapsed time, Δt

use Δt to compute new EKF F, Q

predict x, P

LASER OR RADAR?

LASER

RADAR

Update

SET UP LASER MATRICES

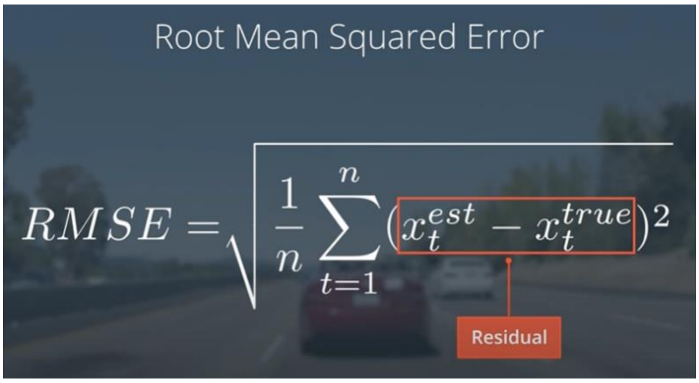
linearize measurement function, $h(x')$

SET UP RADAR MATRICES

STATE UPDATE WITH NEW MEASUREMENT, z

State Vector

$$x = \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix}$$

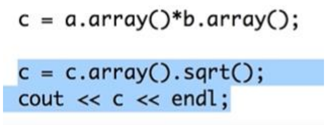


Root Mean Squared Error

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(\boxed{x_t^{est} - x_t^{true}}\right)^2}$$

Residual

Element-wise multiplication and square root



```
c = a.array()*b.array();

c = c.array().sqrt();
cout << c << endl;
```