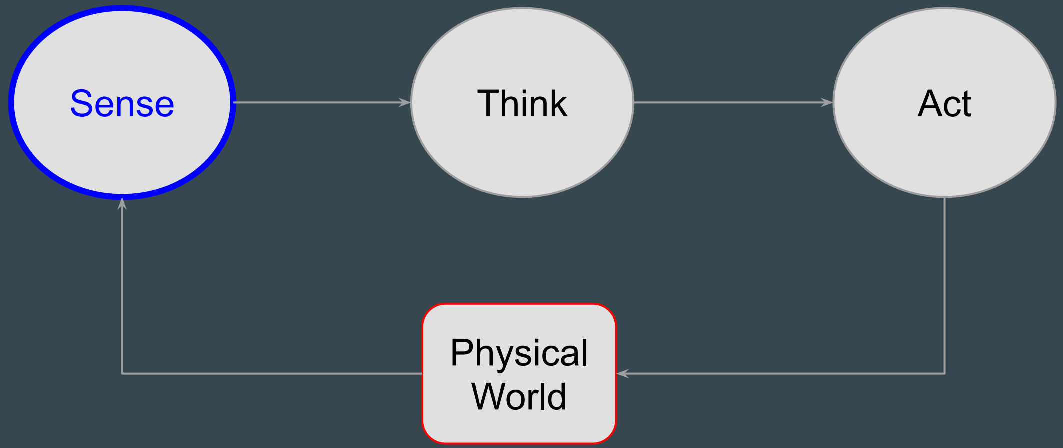
CS4501
Robotics for Soft Eng

•••

Sensors and Noise Management


Sense
Think
Act
Physical World

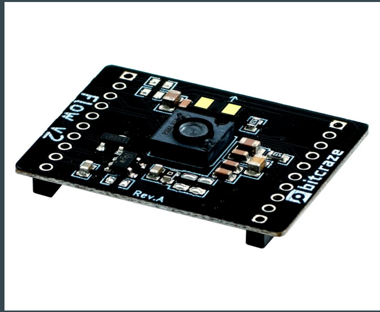
Sensor

- Transduces energy into measure
- Measures a physical quantity (light, force, speed, …)
- Provides window into the world and robot

Sensor

- Transduces energy
- Measures a physical quantity (light, force, speed, …)
- Provides window into the world and oneself

  - BMI088: 3 axis accelerometer / gyroscope ()
  - BMP388: high precision pressure sensor
  - VL53L1x ToF sensor to measure distance up to 4 meters
  - PMW3901 optical flow sensor



Sensor

- Transduces energy
- Measures a physical quantity (ligh
- Provides window into the world a

    - BMI088: 3 axis accelerometer / gyroscope ()
    - BMP388: high precision pressure sensor
    - VL53L1x ToF sensor to measure distance up to
    - PMW3901 optical flow sensor

Parameter	Technical data
Digital resolution	Accelerometer (A): 16-bit Gyroscope (G): 16-bit
Resolution	(A): 0.09 mg (G): 0.004°/s
Measurement range and sensitivity (calibrated)	(A): ± 3 g: 10920 LSB/g ± 6 g: 5460 LSB/g ± 12 g: 2730 LSB/g ± 24 g: 1365 LSB/g (G): ± 125°/s: 262.144 LSB/°/s ± 250°/s: 131.072 LSB/°/s ± 500°/s: 65.536 LSB/°/s ± 1000°/s: 32.768 LSB/°/s ± 2000°/s: 16.384 LSB/°/s
Zero offset (typ. over life-time)	(A): ± 20 mg (G): ± 1°/s
TCO	(A): ± 0.2 mg/K (G): ± 0.015 °/s/K
Noise density (typ.)	(A): 175 µg/√Hz (G): 0.014 °/s/√Hz
Bandwidths (progr.)	5 Hz ... 523Hz
Selectable output data rates	12.5 Hz ... 2 kHz
Digital inputs/outputs	SPI, I²C, 4x digital interrupts
Supply voltage (VDD)	2.4 ... 3.6 V
I/0 supply voltage (VDDIO)	1.2 ... 3.6 V
Temperature range	-40 ... +85°C
Current consumption (full operation)	5.15 mA
LGA package	3 x 4.5 x 0.95 mm³

Sensors in ROS

- Component support for
  - Range finders
  - Cameras
  - Audio
  - Force
  - Pose
  - Power
  - …
- Sensor messages

- Sensor messages
  - Example for images

sensor_msgs/Image Message

File: sensor_msgs/Image.msg

Raw Message Definition

```
# This message contains an uncompressed image
# (0, 0) is at top-left corner of image
#

Header header        # Header timestamp should be acquisition time of image
                     # Header frame_id should be optical frame of camera
                     # origin of frame should be optical center of camera
                     # +x should point to the right in the image
                     # +y should point down in the image
                     # +z should point into to plane of the image
                     # If the frame_id here and the frame_id of the CameraInfo
                     # message associated with the image conflict
                     # the behavior is undefined

uint32 height         # image height, that is, number of rows
uint32 width          # image width, that is, number of columns

# The legal values for encoding are in file src/image_encodings.cpp
# If you want to standardize a new string format, join
# ros-users@lists.sourceforge.net and send an email proposing a new encoding.

string encoding       # Encoding of pixels -- channel meaning, ordering, size
                      # taken from the list of strings in include/sensor_msgs/image_encodings.h

uint8 is_bigendian    # is this data bigendian?
uint32 step           # Full row length in bytes
uint8[] data          # actual matrix data, size is (step * rows)
```

Compact Message Definition

```
std_msgs/Header header
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

autogenerated on Mon, 13 Jan 2020 18:40:17

Sensor Classification

- Proprioceptive (internal state) - sense of itself
  - Measures values internally to the system
  - Battery level, wheel position, gyro
- Exteroceptive (external state) - sense the world
  - Observations of environment
  - Compass, cameras, lidars

Sensor Classification

- Proprioceptive (internal state)
  - Measures values internally to the system
  - Battery level, wheel position, gyro
- Exteroceptive (external state)
  - Observations of environment
  - Compass, cameras, lidars

- Active (emits energy)
  - Optical encoder
  - Radar
- Passive (passively receives energy)
  - Camera
  - Bump

Sensor Classification

- Proprioceptive (internal state)
  - Measures values internally to the system
  - Battery level, wheel position, gyro
- Exteroceptive (external state)
  - Observations of environment
  - Compass

- Active (emits energy)
  - Optical encoder
  - Radar
- Passive (passively receives energy)
  - Camera
  - Bump

Question: Examples of inter, external, active, passive in your body?

Sensor Classification

- Proprioceptive (internal state)
  - Measures values internally to the system
  - Battery level, wheel position, gyro
- Exteroceptive (external state)
  - Observations of environment
  - Compass

- Active (emits energy)
  - Optical encoder
  - Radar
- Passive (passively receives energy)
  - Camera
  - Bump

Question: how the Crazyflie senses?

- BMI088: 3 axis accelerometer / gyroscope (): Pr/Pa
- BMP388: high precision pressure sensor: E/Pa
- VL53L1x ToF sensor to measure distance up to 4 meters: E/Ac
- PMW3901 optical flow sensor: E/Pa

Sensor Classification

- Proprioceptive (internal state)
  - Measures values internally to the system
  - Battery level, wheel position, gyro
- Exteroceptive (external state)
  - Observations of environment
  - Compass

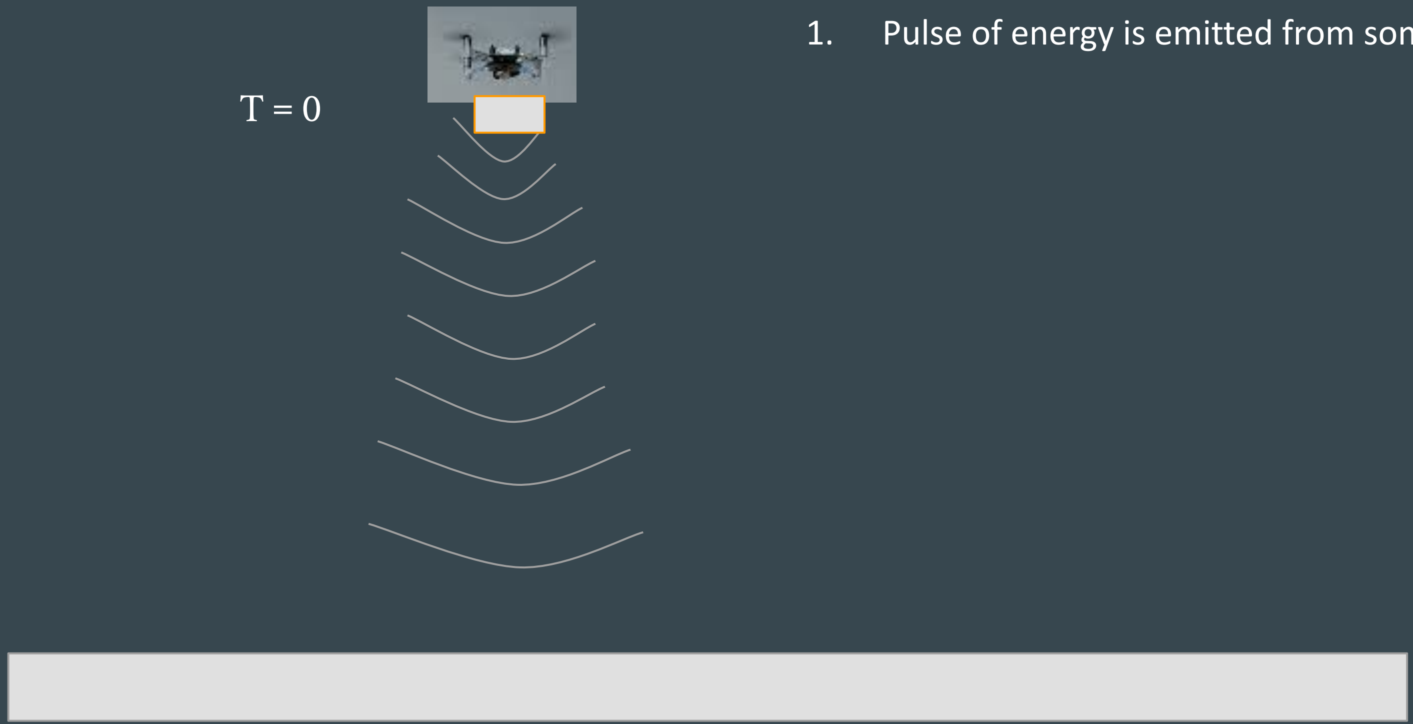
- Active (emits energy)
  - Optical encoder
  - Radar
- Passive (passively receives energy)
  - Camera
  - Bump

Question: how the Crazyflie sensors?

- BMI088: 3 axis accelerometer / gyroscope (): Pr/Pa
- BMP388: high precision pressure sensor: E/Pa
- VL53L1x ToF sensor to measure distance up to 4 meters: E/Ac
- PMW3901 optical flow sensor: E/Pa

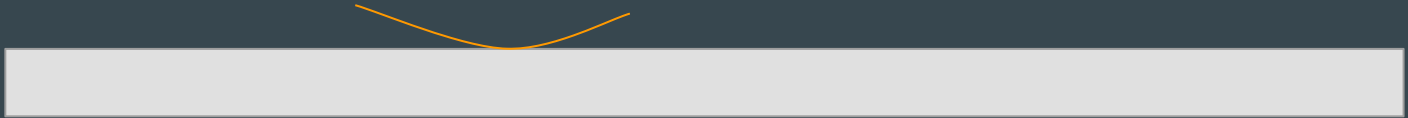
Sonar, ultrasonic, range scanners:

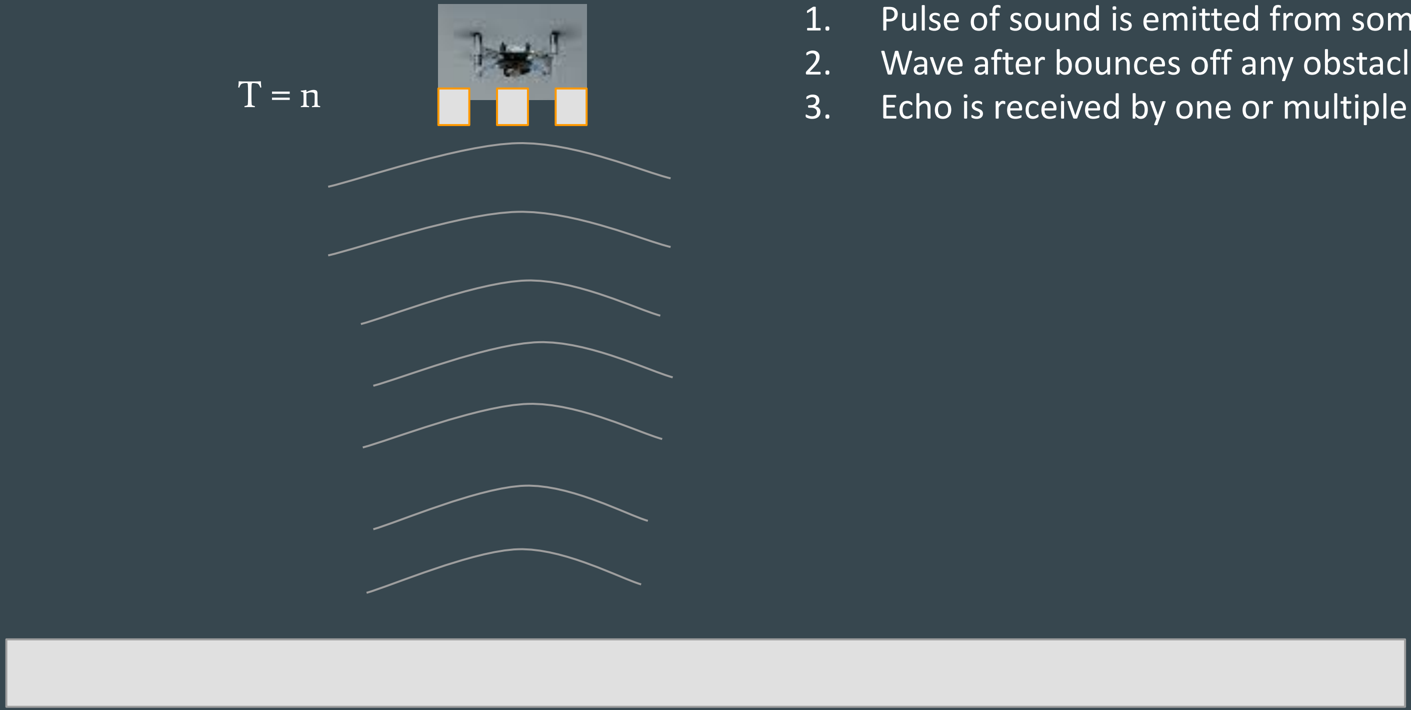
1. Pulse of energy is emitted from some source



Sonar, ultrasonic, range scanners:

1. Pulse of sound is emitted from some source
2. Wave after bounces off any obstacles





Sonar, ultrasonic, range scanners:

1. Pulse of sound is emitted from some source
2. Wave after bounces off any obstacles
3. Echo is received by one or multiple receptors

Sonar, ultrasonic, range scanners:

1. Pulse of sound is emitted from some source
2. Wave after bounces off any obstacles
3. Echo is received by one or multiple receptors
4. Signal is interpreted in various ways to obtain information about an obstacle

MODEL

$d = \frac{1}{2} * v * t$

t is measured

v is known cnst

Let's say this is ultrasonic sensor:

- v= 344 m/s
- If t=0.05 s then d= 8.6m

Sonar, ultrasonic, range scanners:

1. Pulse of sound is emitted from some source
2. Wave after bounces off any obstacles
3. Echo is received by one or multiple receptors
4. Signal is interpreted in various ways to obtain information about an obstacle

MODEL

$d = \frac{1}{2} * v * t$

t is measured

v is known cnst

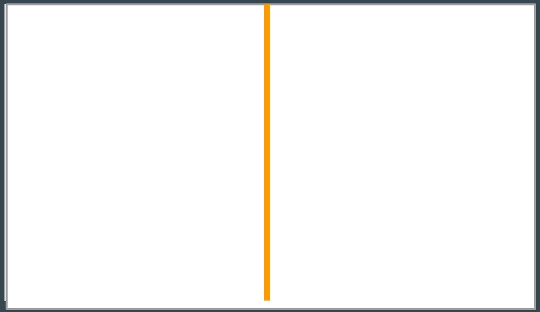
Let's say this is ultrasonic sensor:

- v= 344 m/s
- If t=0.05 s then d= 8.6m

Assumptions - leaky abstractions

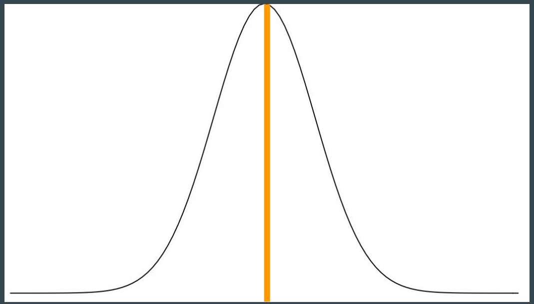
- v= 344 m/s with dry air, 21 C, sea level
- Surfaces are …

Sensor Noise - Modified Signal



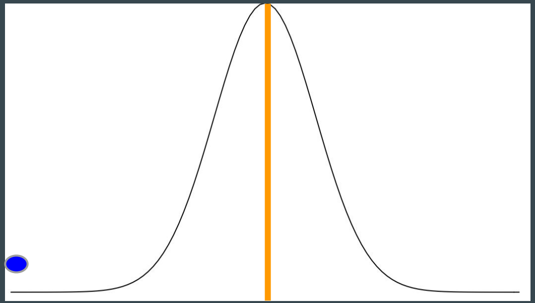
- Single reading

Sensor Noise - Modified Signal



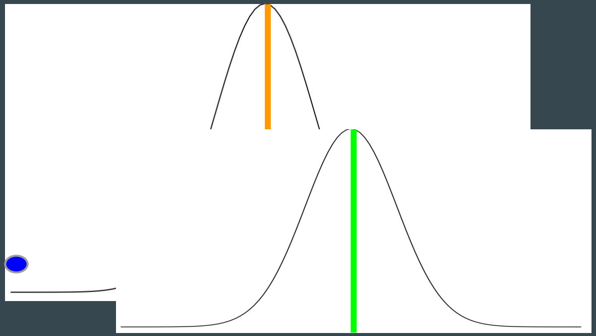
- Single reading
- Belongs belong to a distribution

Sensor Noise - Modified Signal



- Single reading
- Belong belong to a distribution
- Outliers

Sensor Noise - Modified Signal



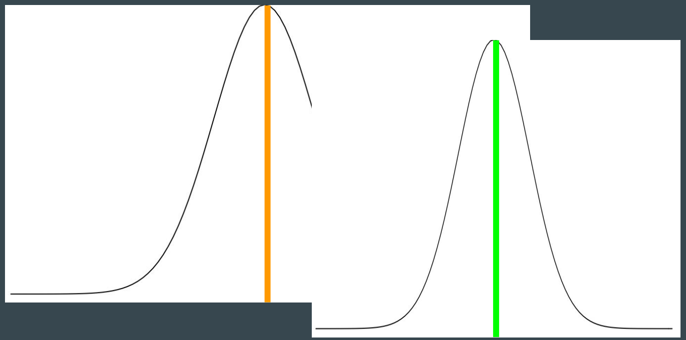
- Single reading
- Belong belong to a distribution
- Outliers
- Shifts

Managing Sensor Noise

- Calibration
- Filtering
- Fusing

Calibration

- Shifts in distribution due to environmental assumptions
- Adjusting sensor for more accurate physical measurements within context
- Process
    a. Conduct standardized tests
    b. Recompute constants and error estimates
    c. Redefine model parameters



Calibration Problem

MODEL
$d = \frac{1}{2} * v * t$
t is measured
v is known cnst

Let's say this is ultrasonic sensor:

- v= 344 m/s
- If t=0.05 seconds, then d= 8.6m

Assumptions

- v= 344 m/s with dry air, 21 C, sea level
- Surfaces are …
- When assumptions break, measures are off!

Calibration Problem

MODEL

$d = \frac{1}{2} * v * t$

t is measured

v is known cnst

Let's say this is ultrasonic sensor:

- v= 344 m/s
- If t=0.05 seconds, then d= 8.6m

Assumptions

- v= 344 m/s with dry air, ~~21 C, sea level~~ -4.8
- Surfaces are …
- When assumptions break, measures are off!

ALTITUDE	TEMPERATURE	SPEED OF SOUND
Meter (m)	Celcius (°C)	m/s
0 (sea level)	21	344
3048 (10k ft)	-4.8	328
6096 (20k ft)	-24.6	316
9144 (30k ft)	-44.4	303

Calibration Problem

MODEL

$d = \frac{1}{2} * v * t$

t is measured

v is known cnst

Let's say this is ultrasonic sensor:

- ~~v= 344 m/s~~ 328
- If t=0.05 seconds, then d = ~~8.6m~~ 8.2m

Assumptions

- v= 344 m/s with dry air, ~~21 C, sea level~~ -4.8
- Surfaces are ...
- When assumptions break, measures are off!

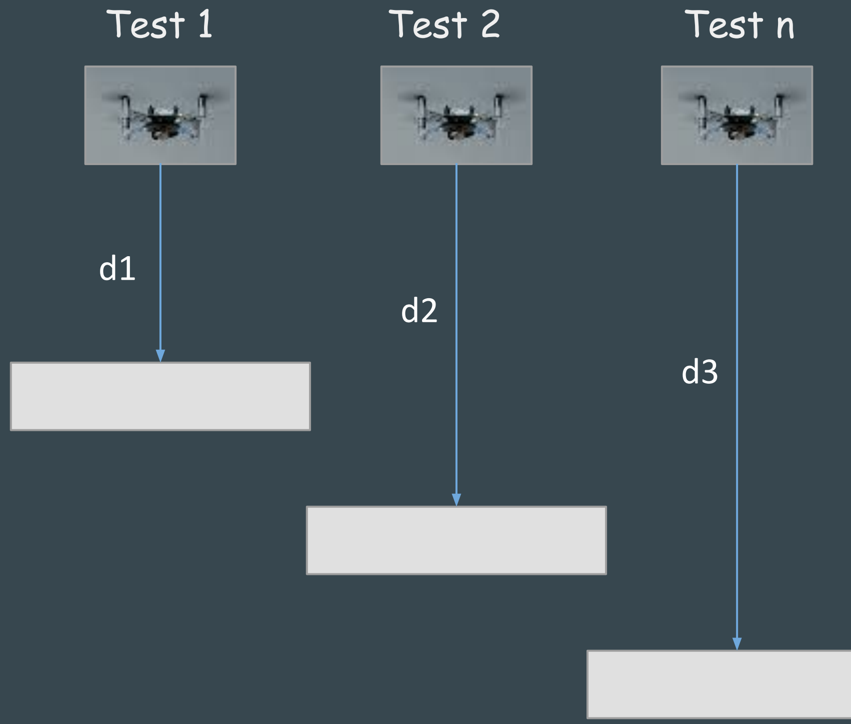
ALTITUDE	TEMPERATURE	SPEED OF SOUND
Meter (m)	Celcius (°C)	m/s
0 (sea level)	21	344
3048 (10k ft)	-4.8	328
6096 (20k ft)	-24.6	316
9144 (30k ft)	-44.4	303

Calibration

MODEL
$d = \frac{1}{2} * v * t$
t is measured
v is unknown cnst

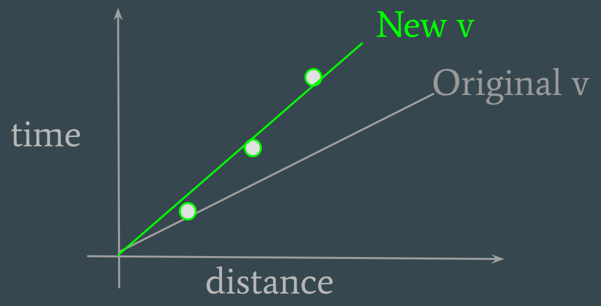
So let's fix d to find v

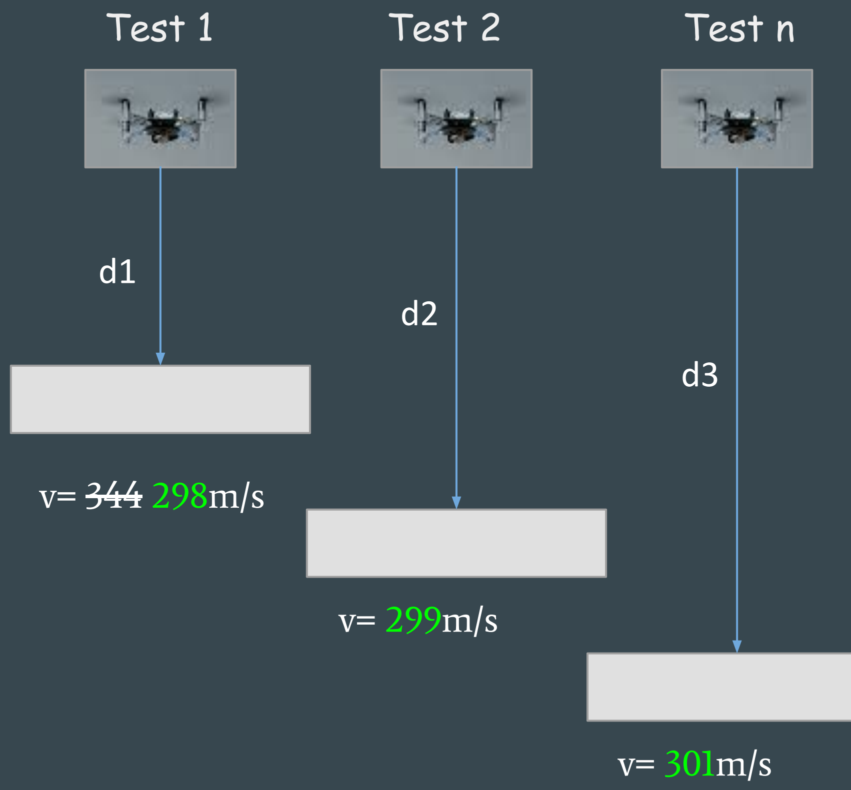
Test 1: d = d1, v = d1 * 2 / t1
Test 2: d = d2, v = d2 * 2 / t2
...
Test n: d = dn, v = dn * 2 / tn



Calibration

MODEL
d = ½ * v * t
t is measured
v' is known contextualized cnst





Calibration

MODEL

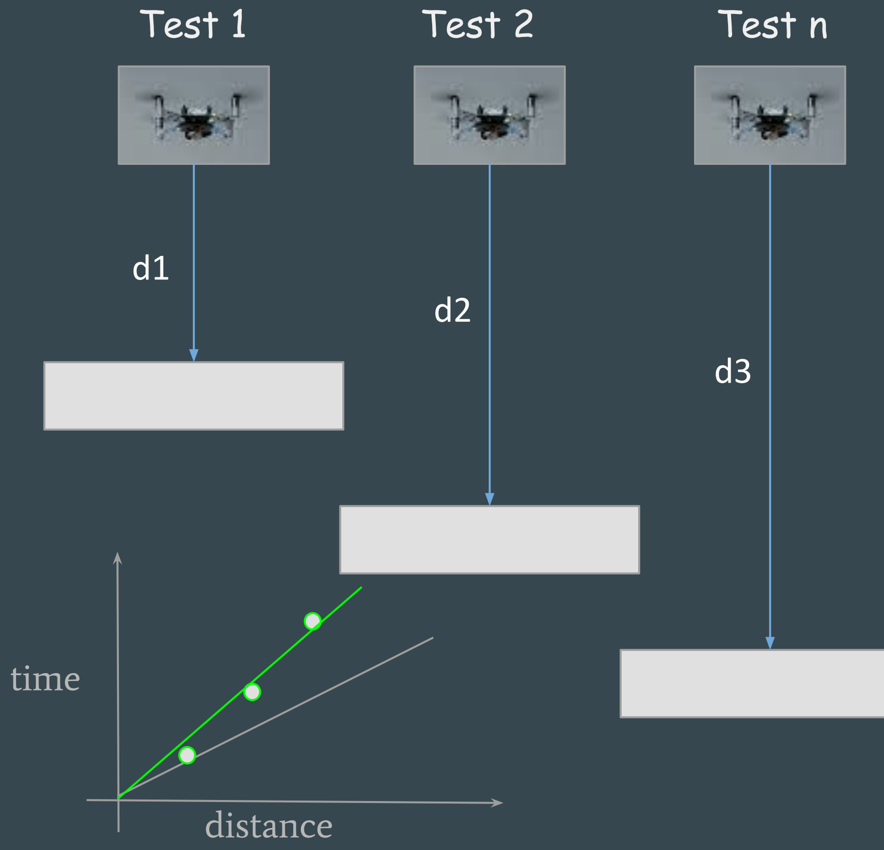
$d = \frac{1}{2} * v * t$

t is measured

v' is known contextualized cnst

Let's say this is ultrasonic sensor:

- v = 300 m/s
- If t=0.05 seconds, d= ~~8.6m~~ 7.5m

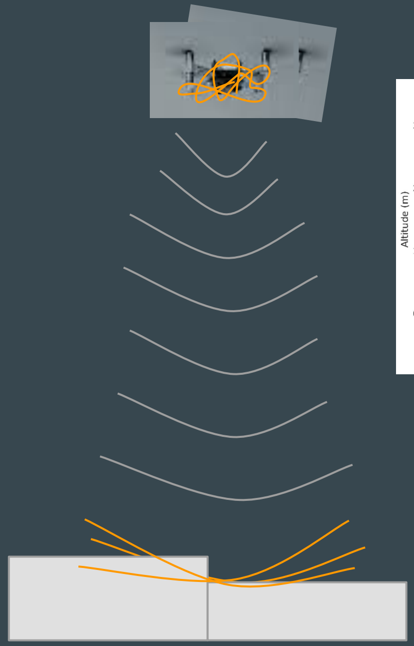


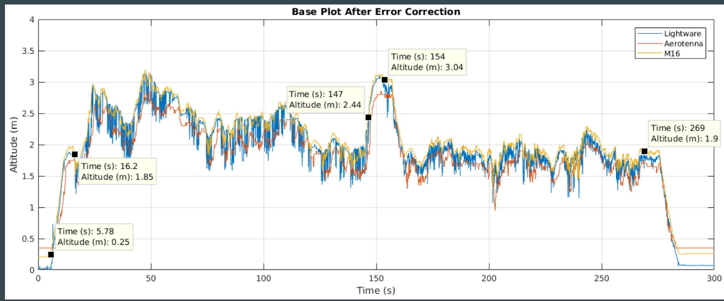
Filtering Problem

- Signal gets distorted
- Interference causes lost reading
- Sensor pose shifts

Filtering Problem

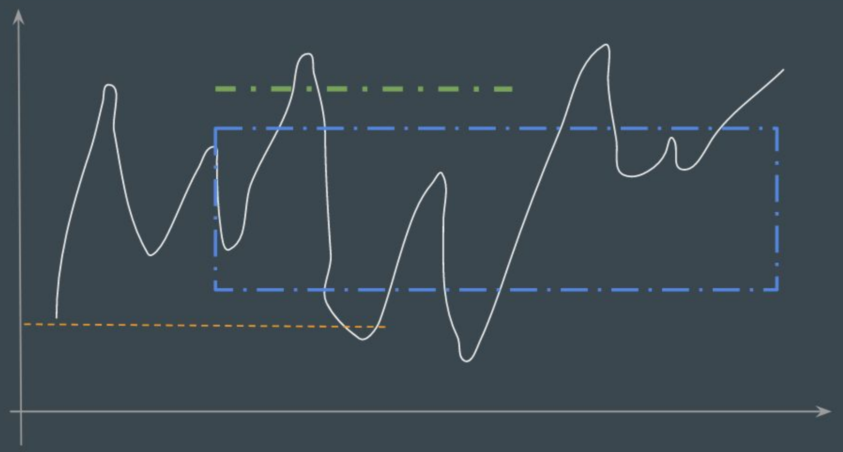
- Signal gets distorted
- Interference causes lost reading
- Sensor pose shifts





Basic Filters from Signal Processing

- Low-pass filters
- High-pass filters
- Band filters



Filtering as smoothing

- Moving averages

$$Y_t = (X_t + X_{t-1} + X_{t-2} + \ldots + X_{t-window}) / windowSize$$

Filtering as smoothing

- Moving averages

Yt = (Xt + Xt-1 + Xt-2 + … + Xt-window) / windowSize

… 64 65 64 78 65 66 67 64 …

64

windowSize = 3

Filtering as smoothing

- Moving averages

Yt = (Xt + Xt-1 + Xt-2 + … + Xt-window) / windowSize

…	64	65	64	78	65	66	67	64	…
		64	69						

windowSize = 3

Filtering as smoothing

- Moving averages

Yt = (Xt + Xt-1 + Xt-2 + … + Xt-window) / windowSize

…	64	65	64	78	65	66	67	64	…
		64	69	69					

windowSize = 3

Filtering as smoothing

- Moving averages

Yt = (Xt + Xt-1 + Xt-2 + … + Xt-window) / windowSize

… 64 65 64 78 65 66 67 64 …

64 69 69 70

windowSize = 3

Filtering as smoothing

- Moving averages

$$Y_t = (X_t + X_{t-1} + X_{t-2} + \dots + X_{t-window}) / windowSize$$

…	64	65	64	78	65	66	67	64	…

67

windowSize = 5

Larger windows stronger smoothing

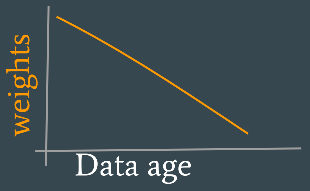
Filtering as smoothing

- Moving averages

$$Y_t = (X_t + X_{t-1} + X_{t-2} + \dots + X_{t-window}) / windowSize$$

- Generalized with weights for decaying

$$Y_t = a\,X_t + a_1\,X_{t-1} + a_2\,X_{t-2} + a_3\,X_{t-3} + \dots \quad \text{where } a+a_1+a_2+a\dots = 1$$



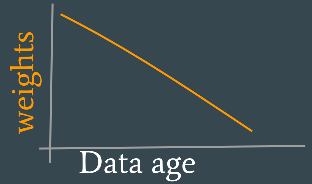
Filtering as smoothing

- Moving averages

Yt = (Xt + Xt-1 + Xt-2 + … + Xt-window) / windowSize

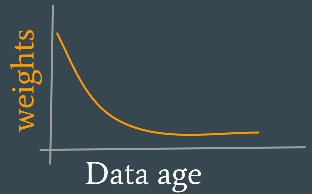
- Generalized with weights for decaying

Yt = α Xt + α1 Xt-1 + α2 Xt-2 + α3 Xt-3 + …   where α+α1+α2+α… = 1



- Generalized with exponential weights for decaying

Yt = α [Xt + (1 – α) Xt-1 + (1 - α)*2 Xt-2 + (1 - α)*3 Xt-3 + …]

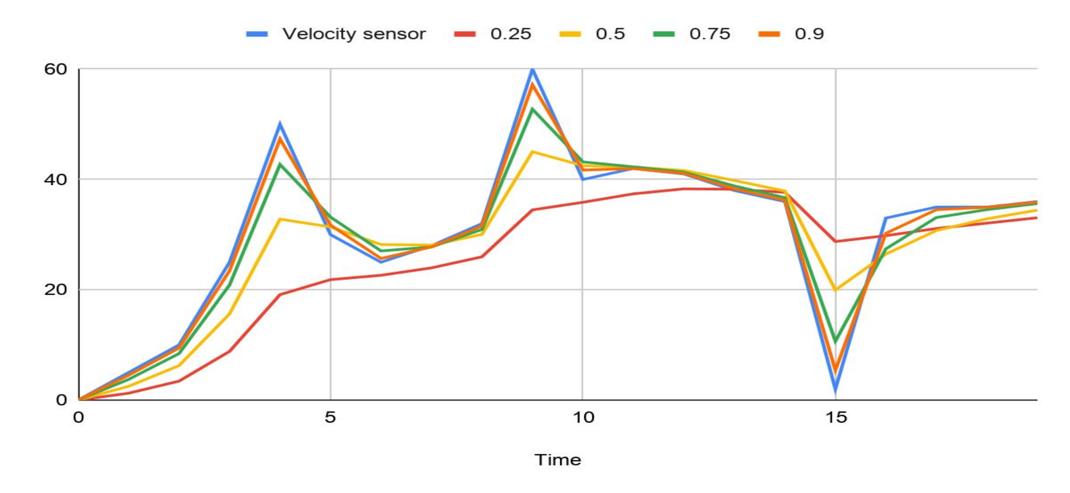


Filtering as smoothing

- Generalized with exponential weights for decaying

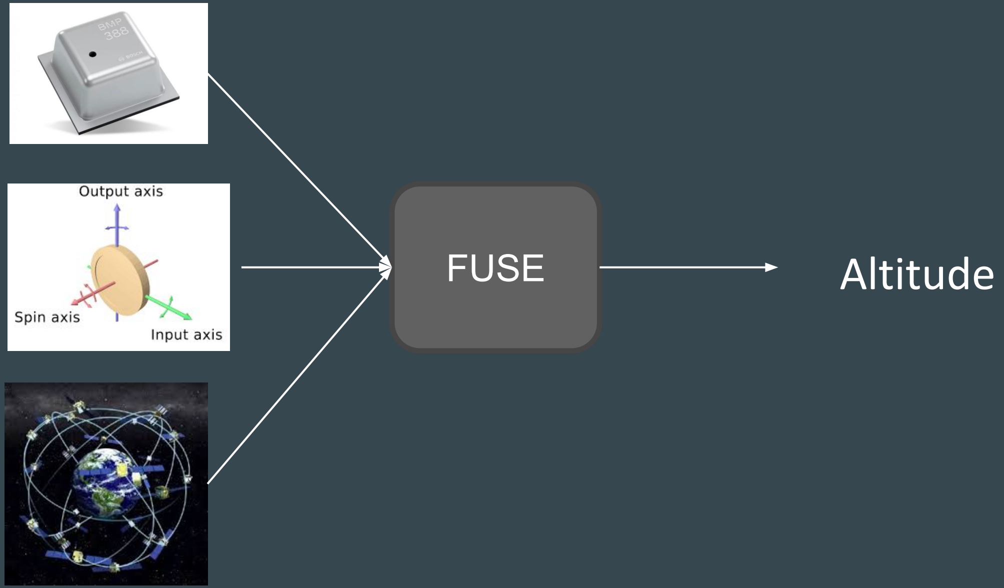
  Yt = α Xt + (1 – α) Xt-1 + (1 - α)*2 Xt-2 + (1 - α)*3 Xt-3 + …
  Or efficiently approximated
  Yt = Yt-1 + α (Xt - Yt-1)

- Selection of α is crucial
  - α closer to 0: closer to last value
  - α closer to 1: no filtering

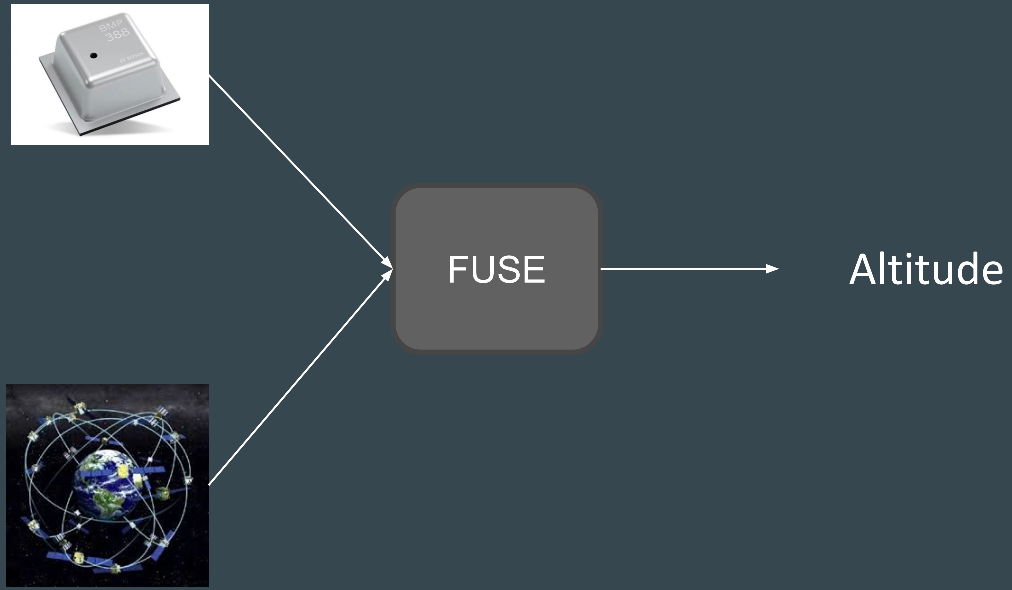


What if our sensor is still really noise?

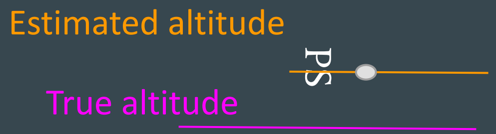
Add sensors with complementary attributes and fuse them



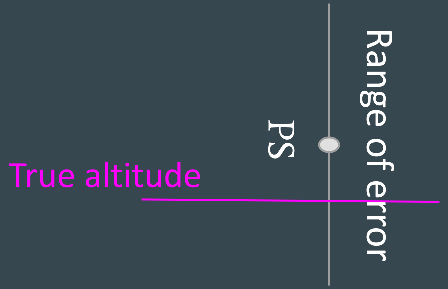
What if our sensor is still really noise?



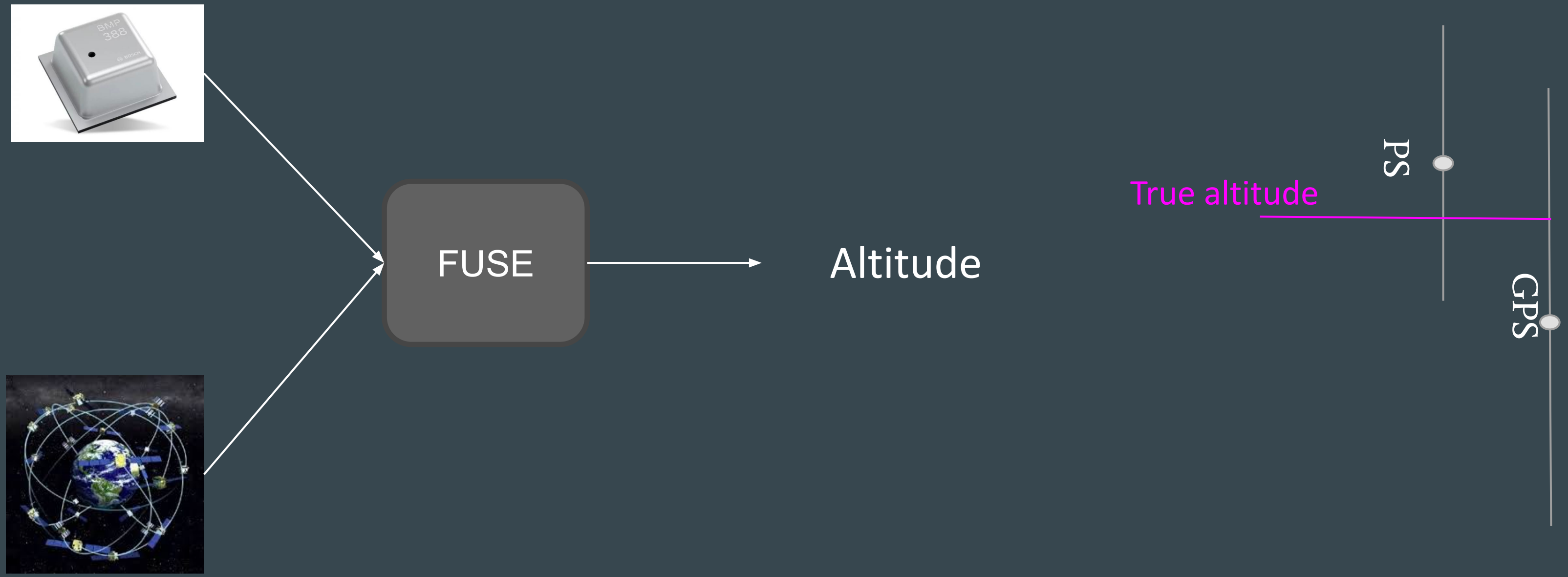
What if our sensor is still really noise?



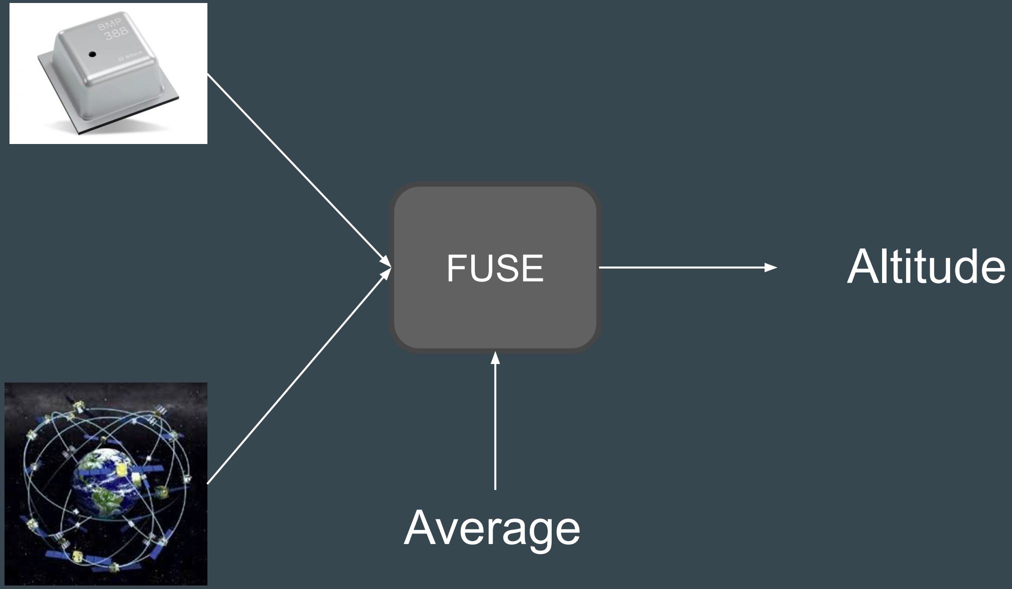
What if our sensor is still really noise?

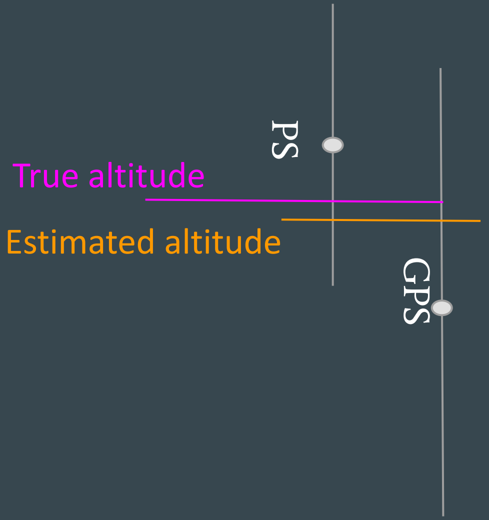


What if our sensor is still really noise?



What if our sensor is still really noise?

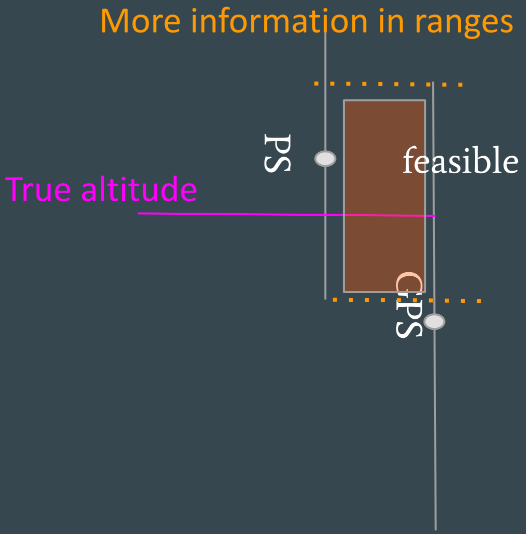




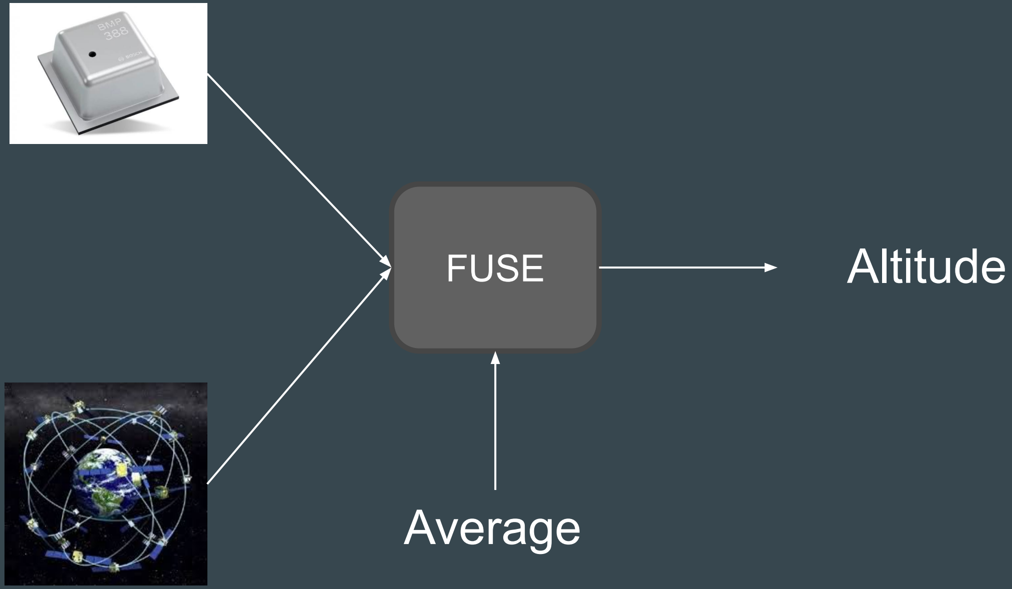
What if our sensor is still really noise?

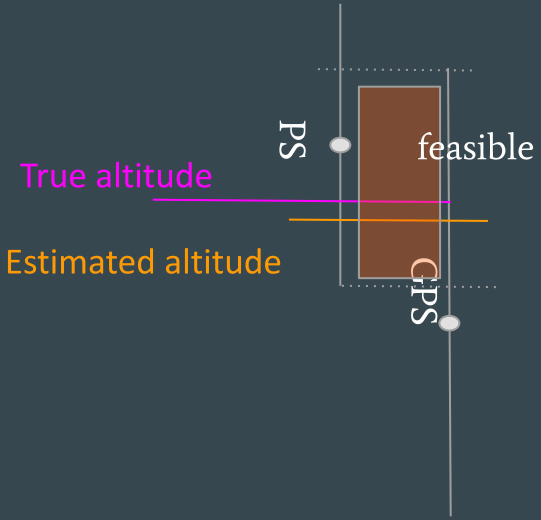
FUSE

Altitude



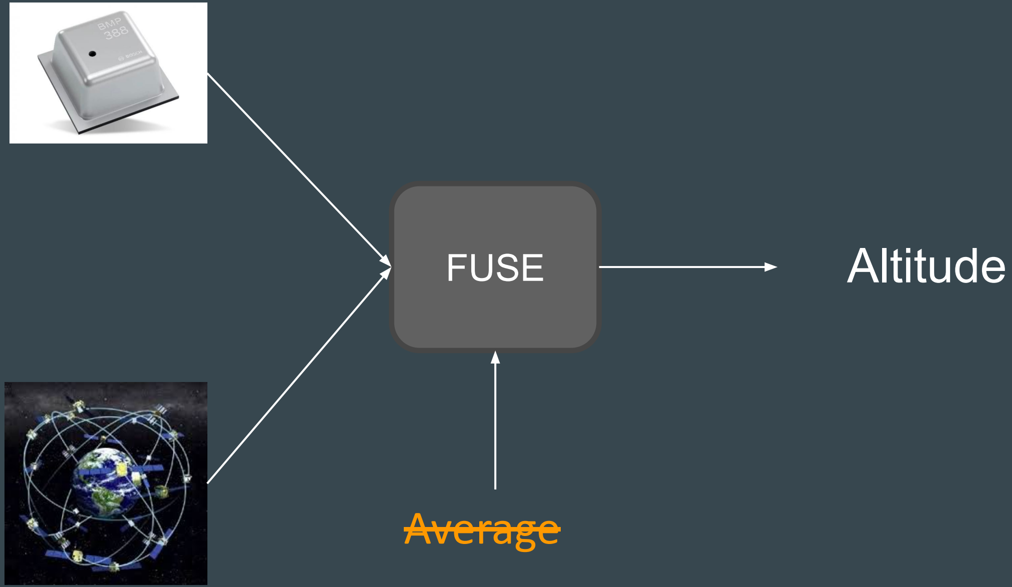
What if our sensor is still really noise?

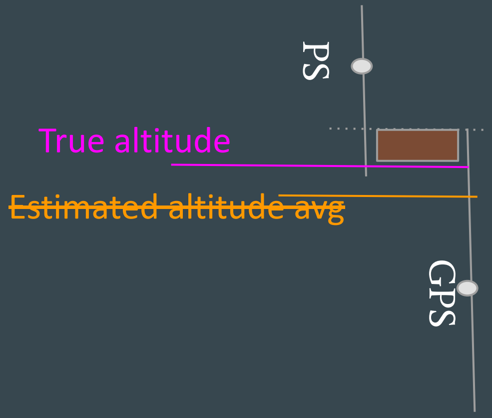




What if our sensor is still really noise?

What if overlapping range was tighter?





What if our sensor is still really noise?

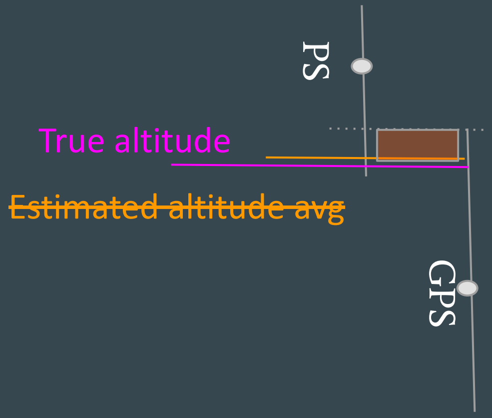
What if overlapping range was tighter?

- Within range, identify closest to average

FUSE

Altitude

~~Average~~



What if our sensor is still really noise?

What if overlapping range was tighter?

- Within range, identify closest to average
- Use different fusing function

FUSE

Altitude

Weighted Average

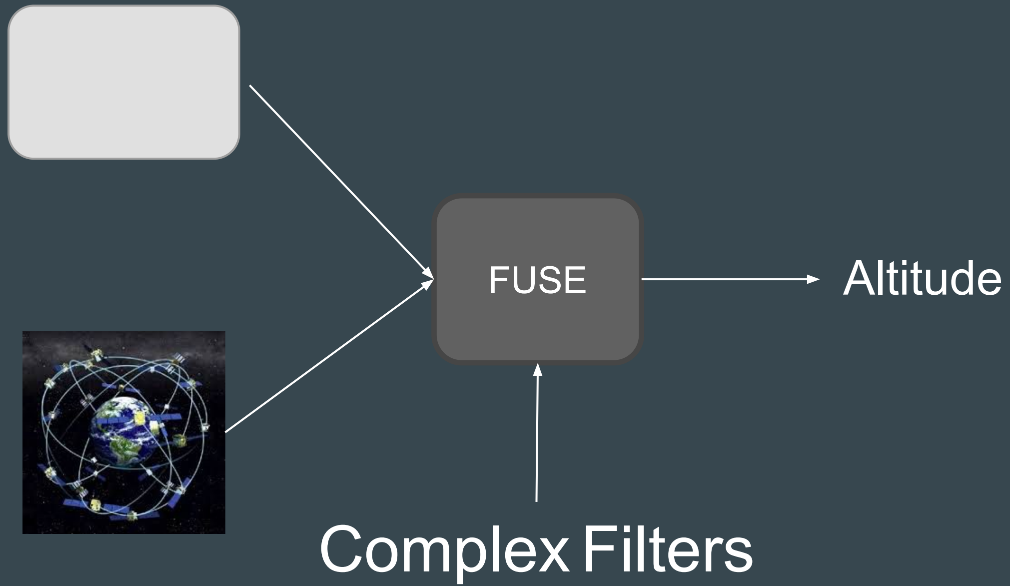
Let’s favor GPS… but what if flying indoors?

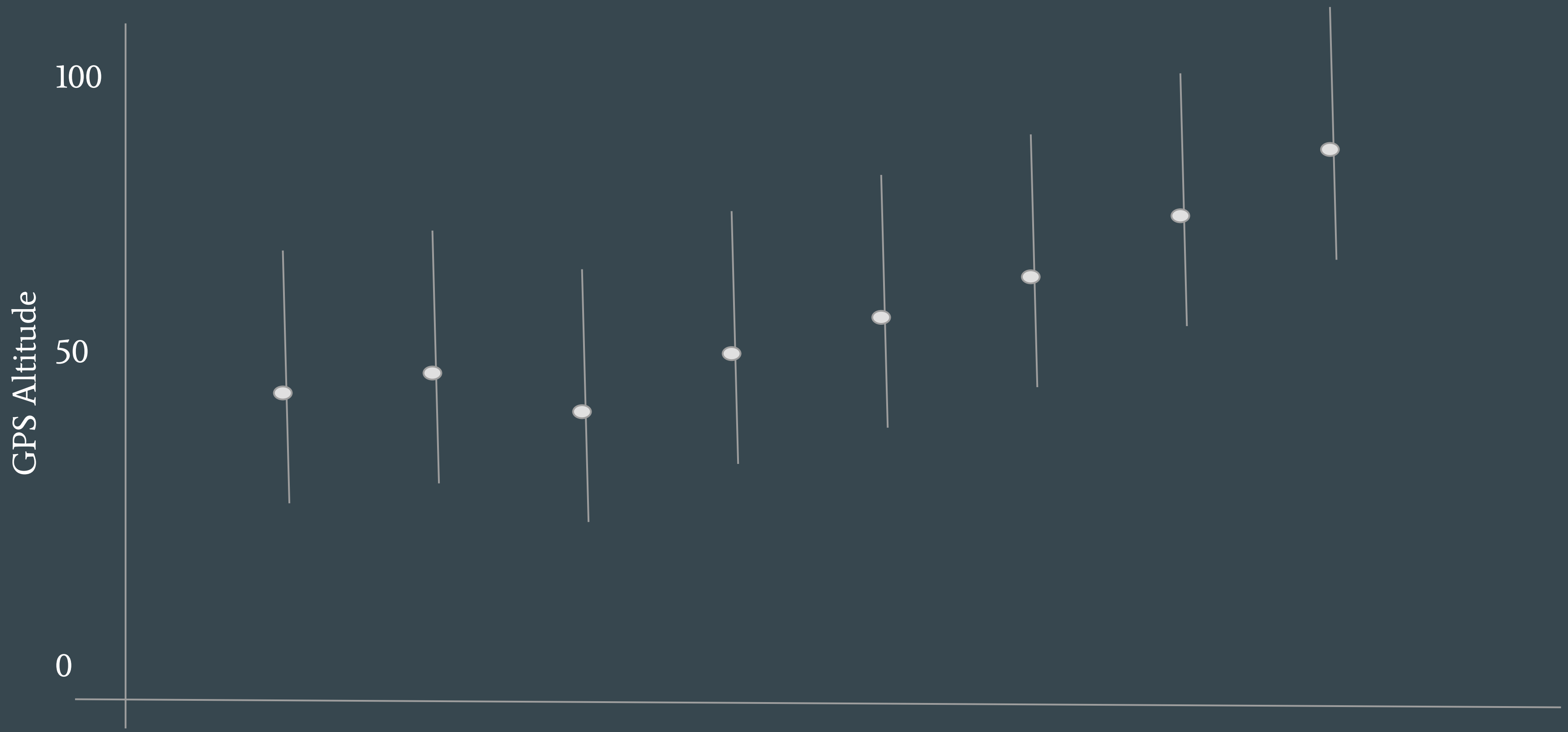
What if our sensor is still really noise?

FUSE → Altitude

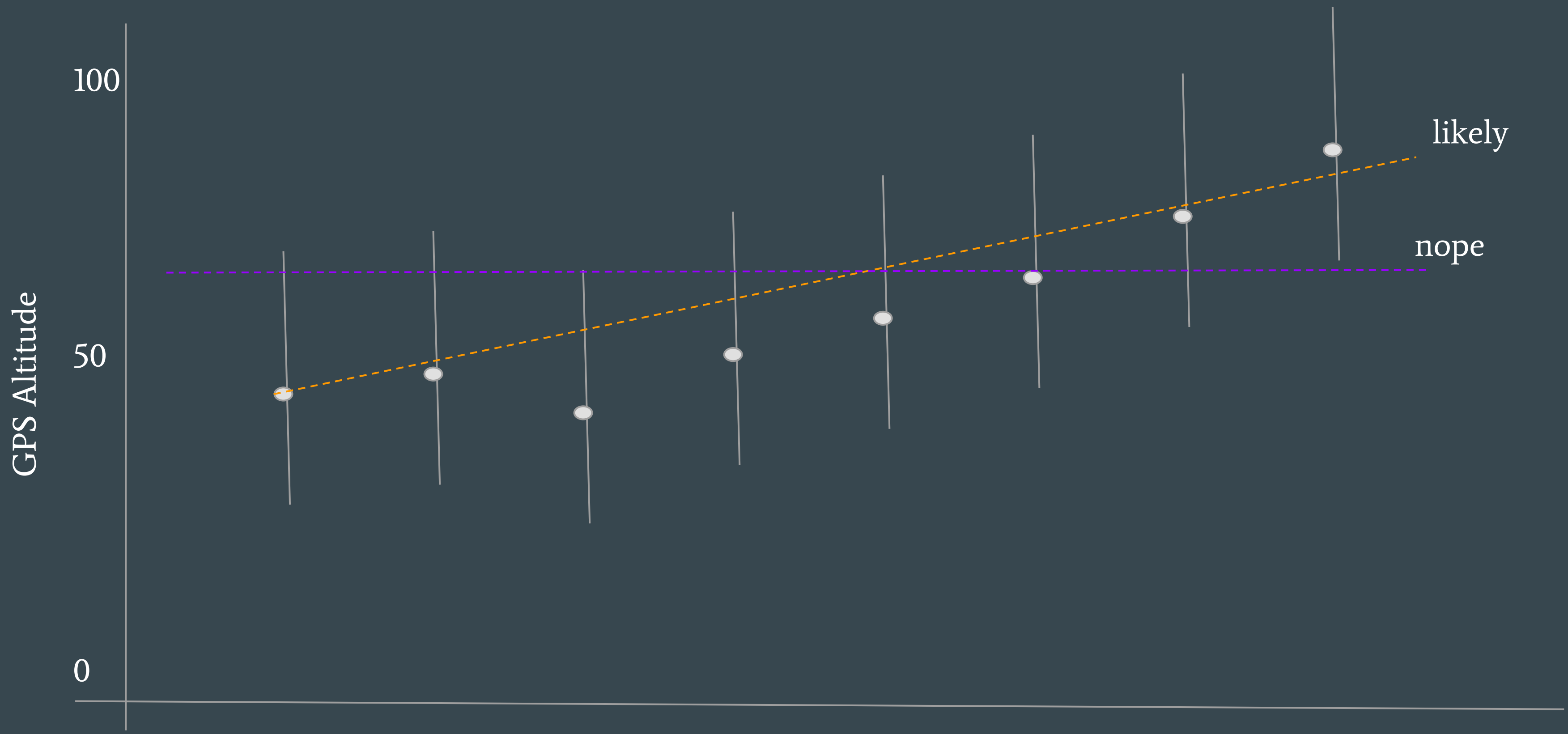
Multiple sensors are better than one. Need to learn how to combine them to leverage all information

What if our sensors are still noise?

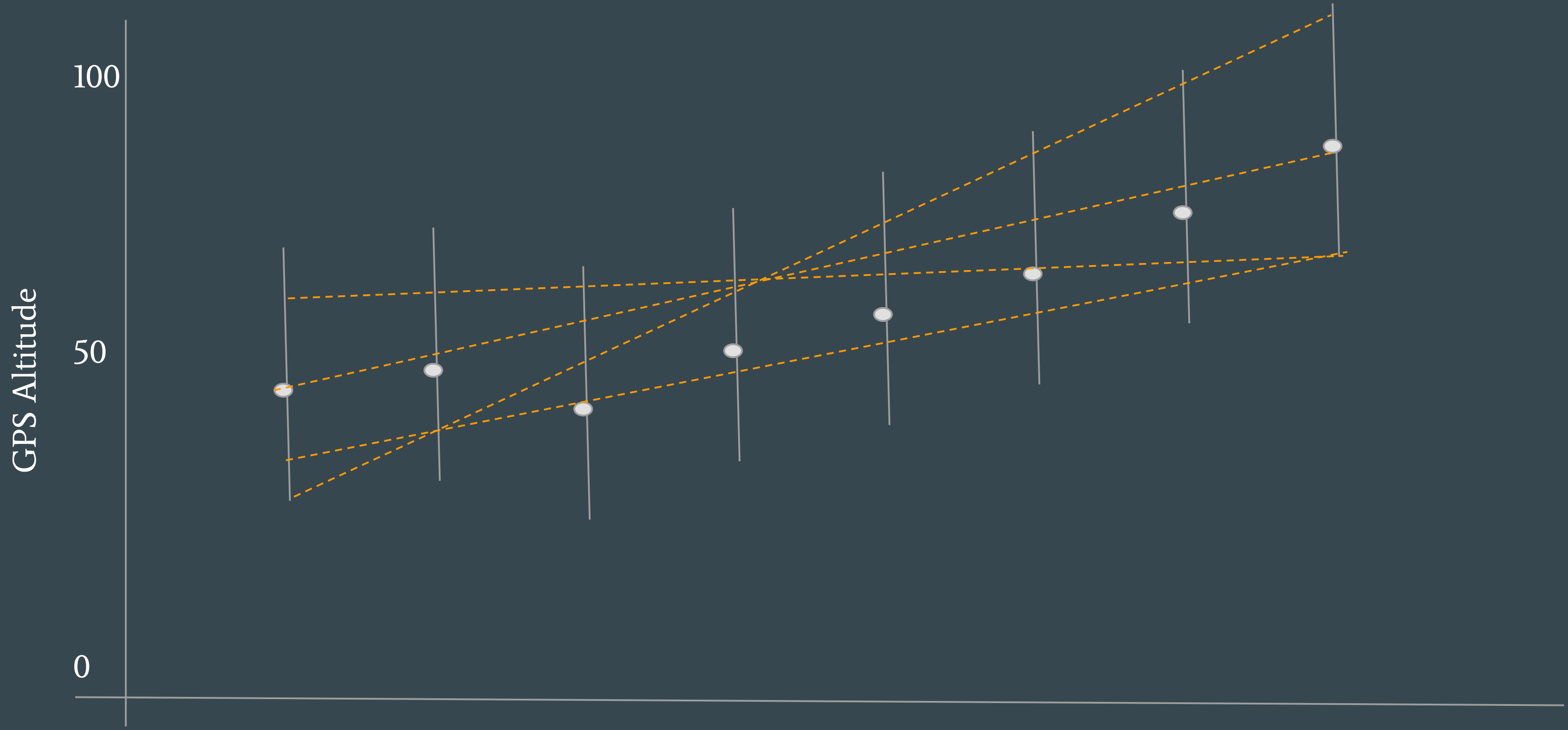




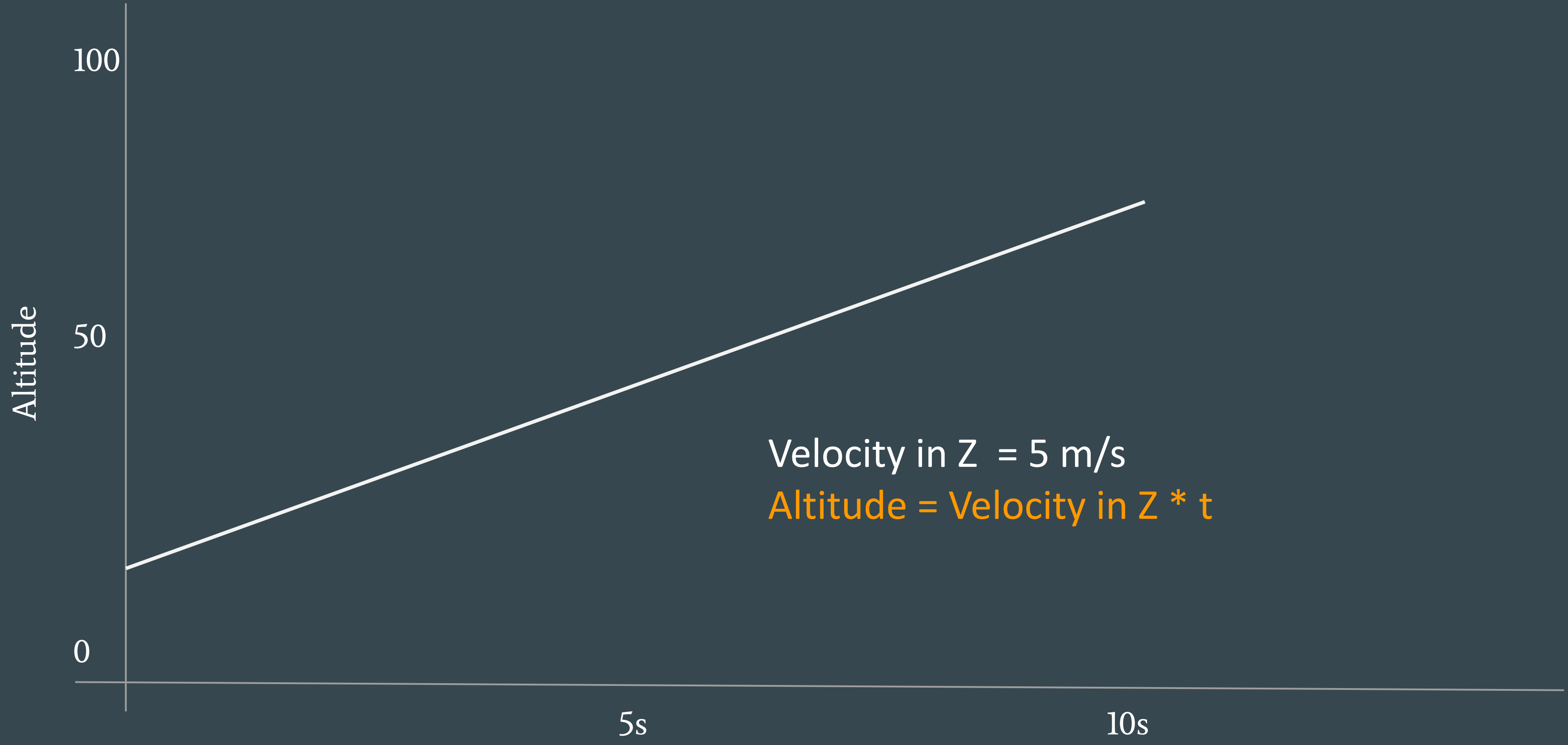
What is the drone doing?



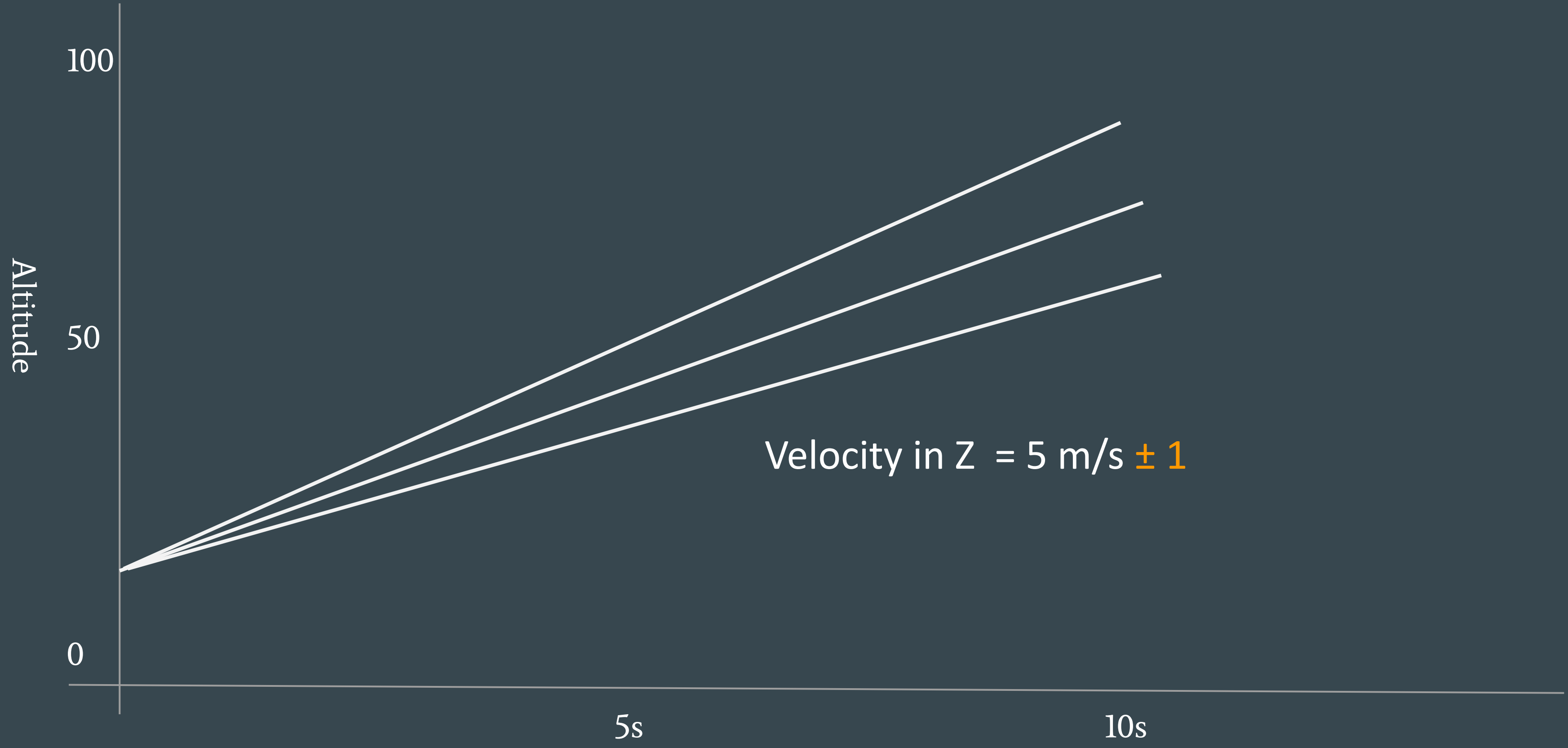
It is not going down
It does not seem to be hovering
It seems to be going up…



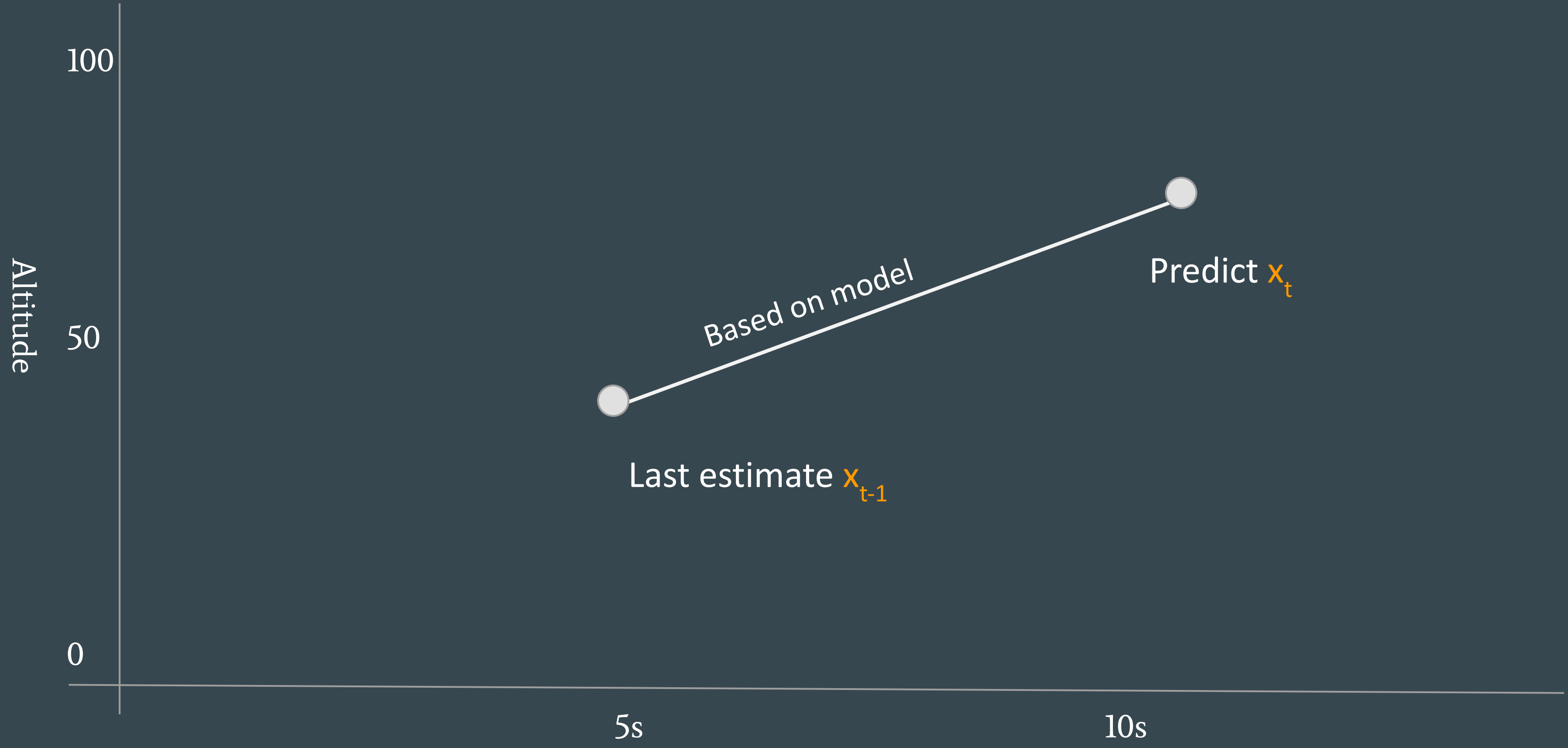
Going up by how much?

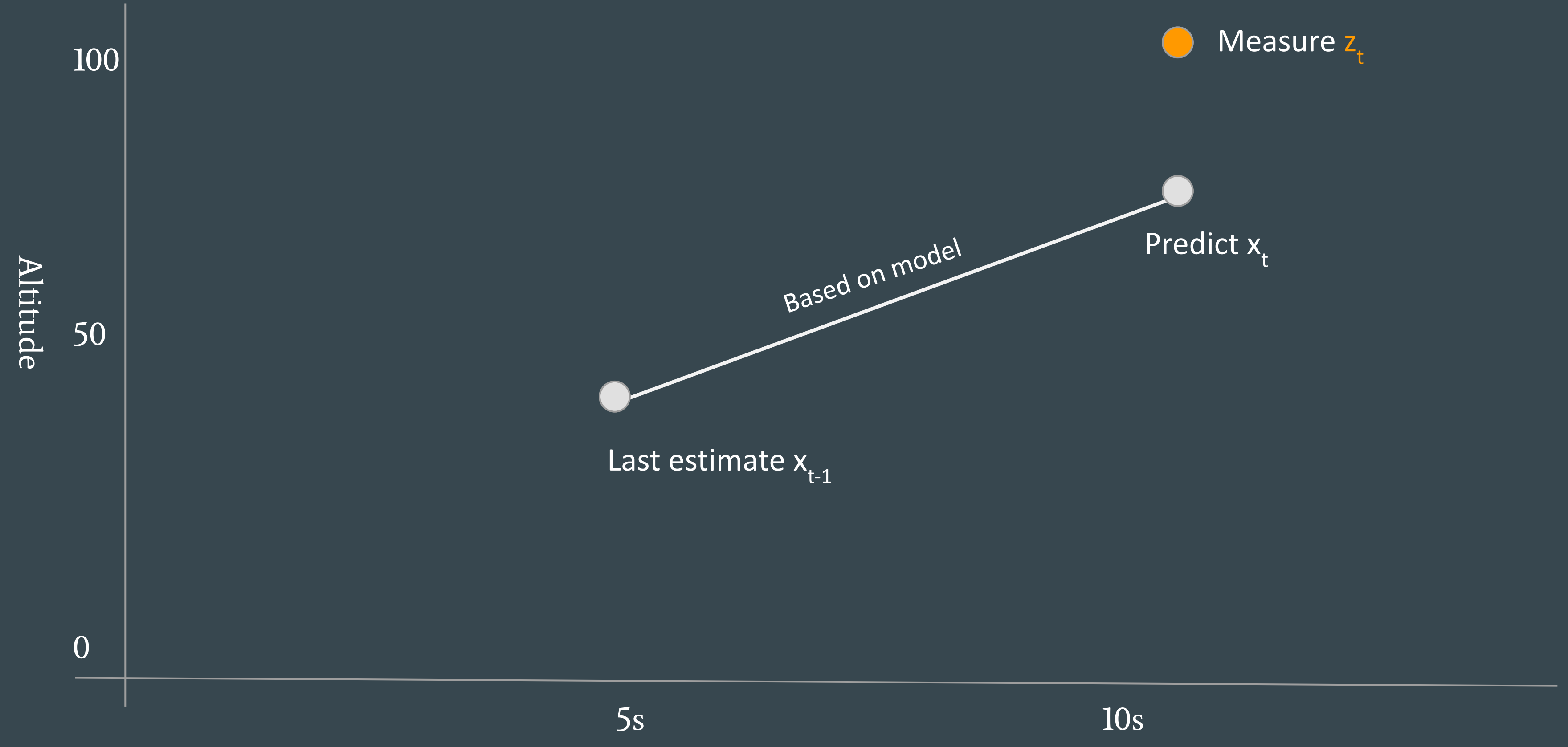


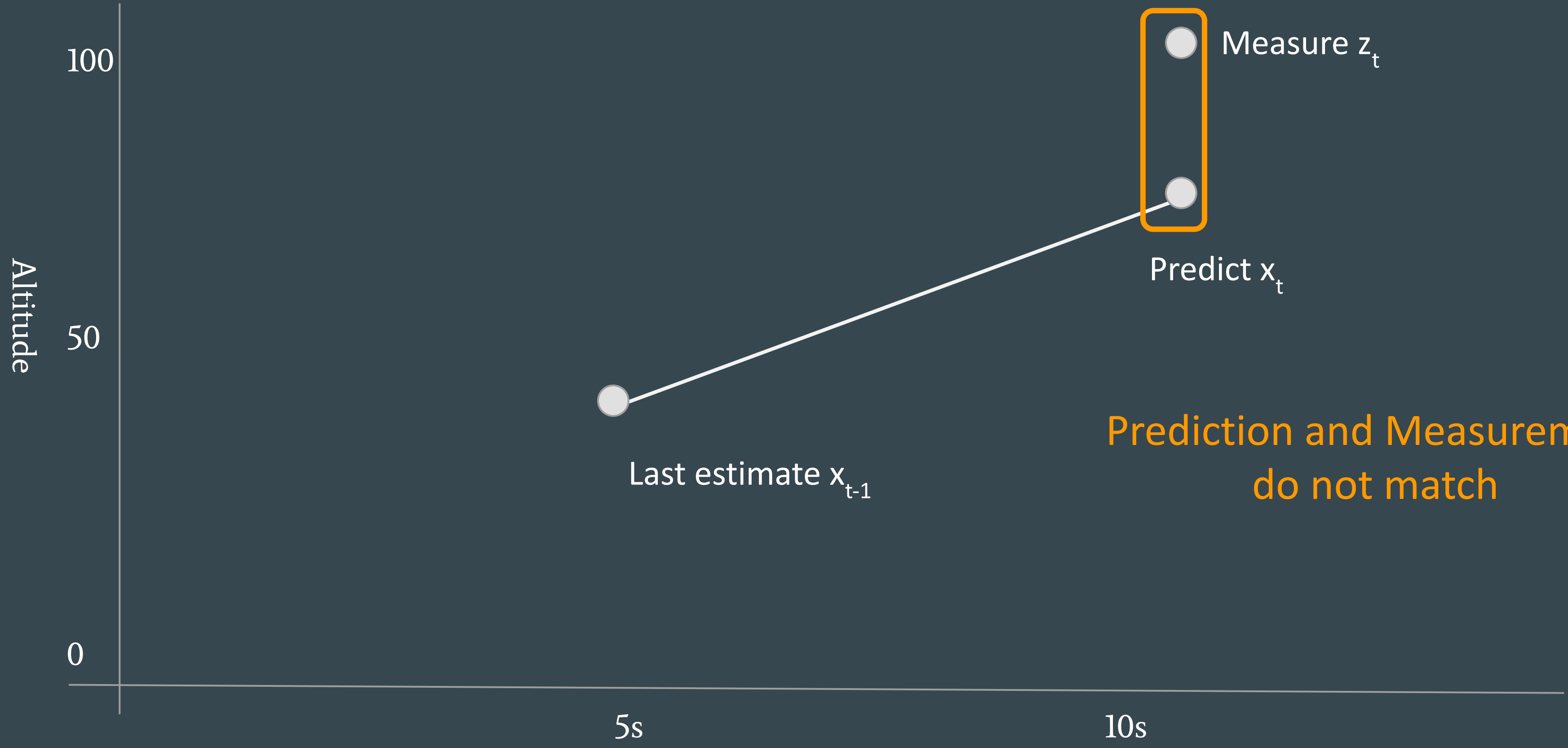
If I had another sensor with a model…



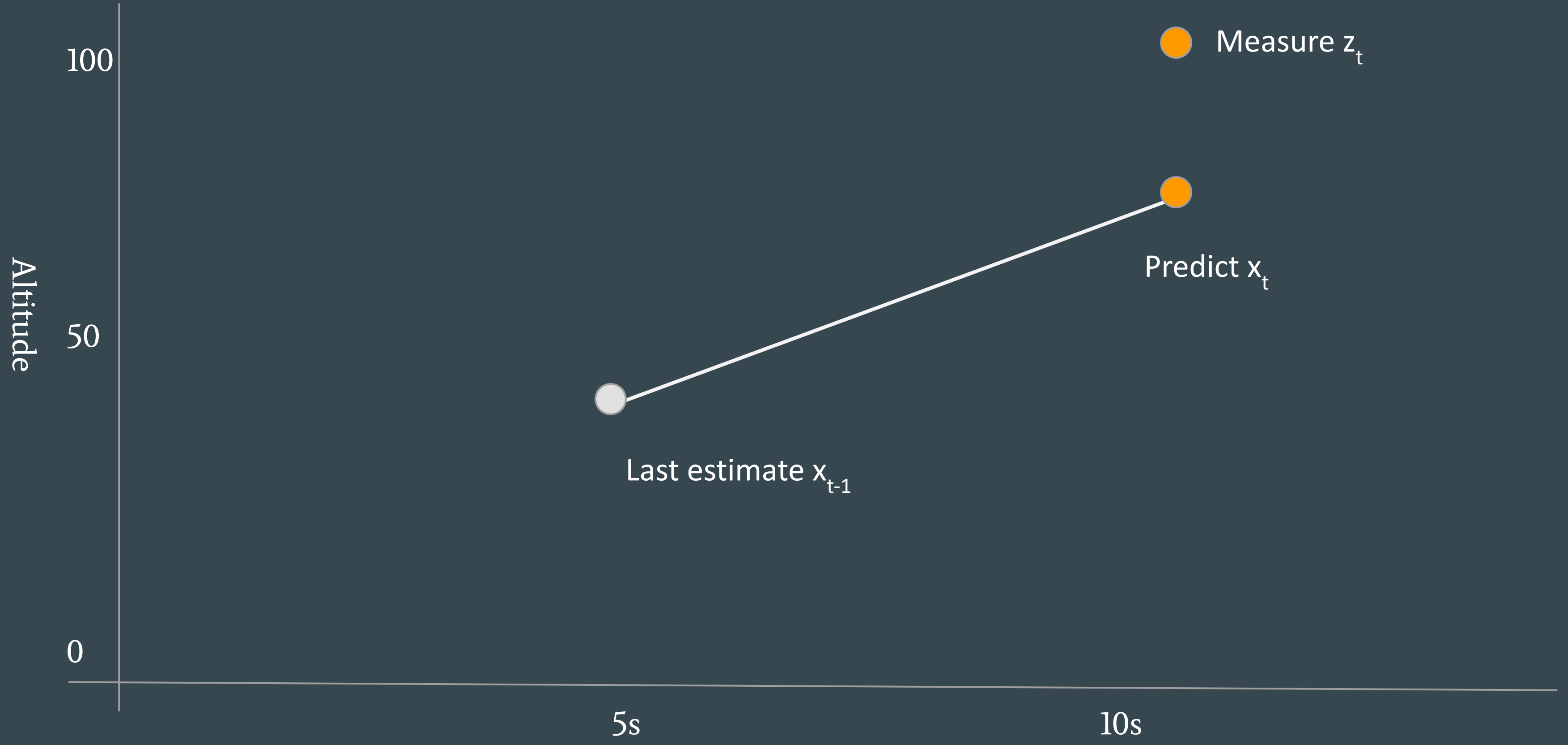
If I had another sensor with a model… but is not perfect either


100
50
0
Altitude
Based on model
Predict x_t
Last estimate x_{t-1}
5s
10s

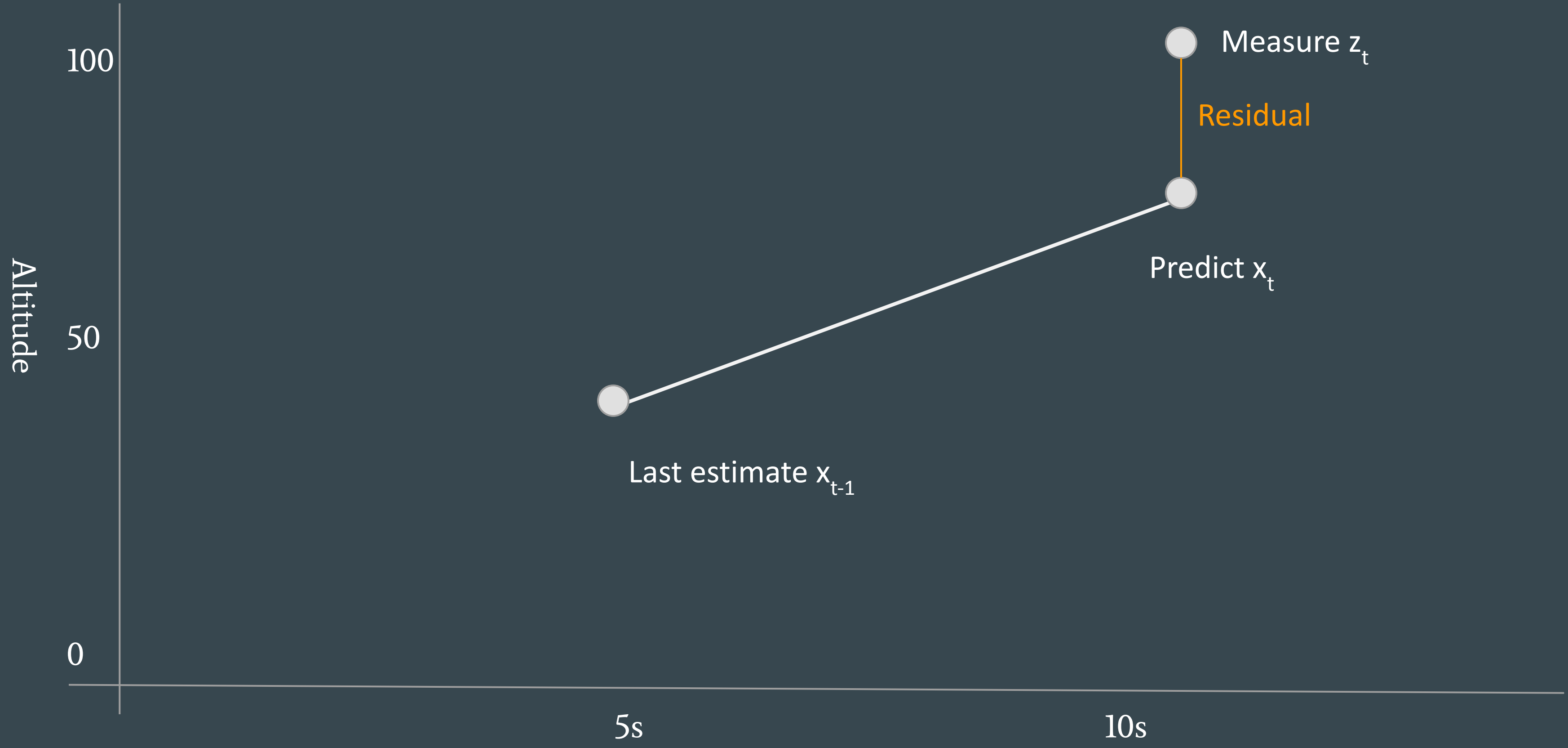

Measure z_t
100
Altitude
50
0
Based on model
Predict x_t
Last estimate x_{t-1}
5s
10s



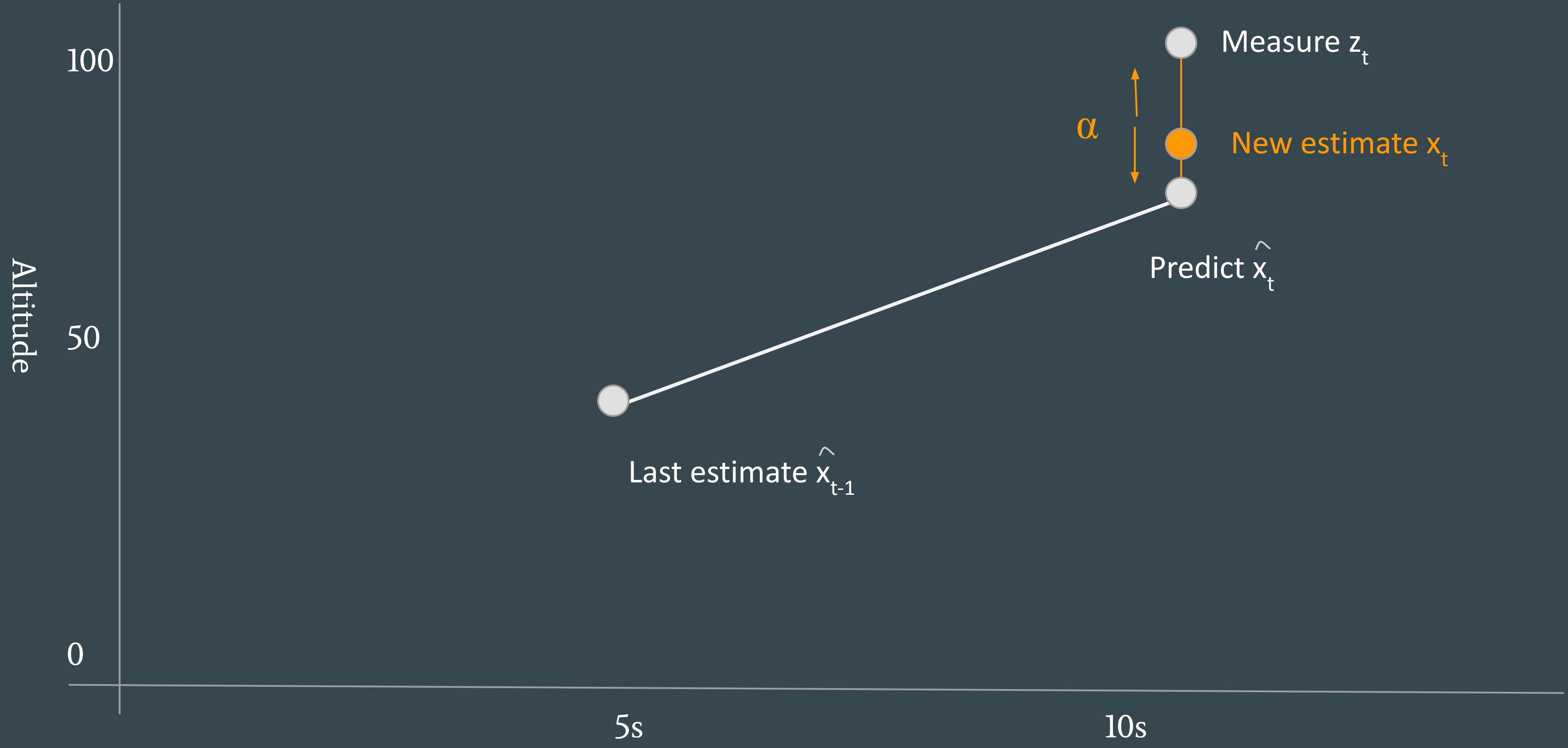
Prediction and Measurement do not match


Measure z_t
100
Altitude
50
Predict x_t
Last estimate x_{t-1}
0
5s
10s

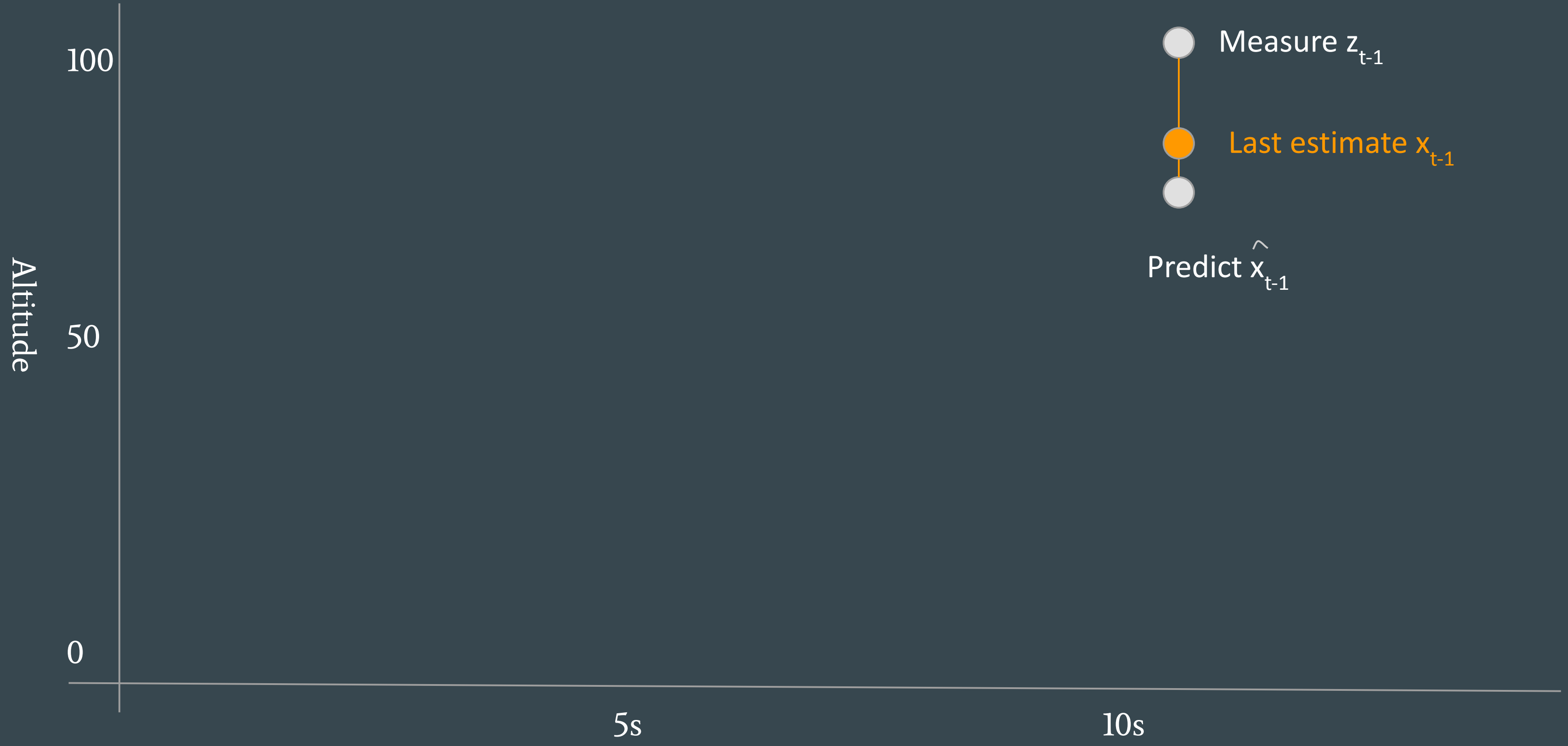
Blend



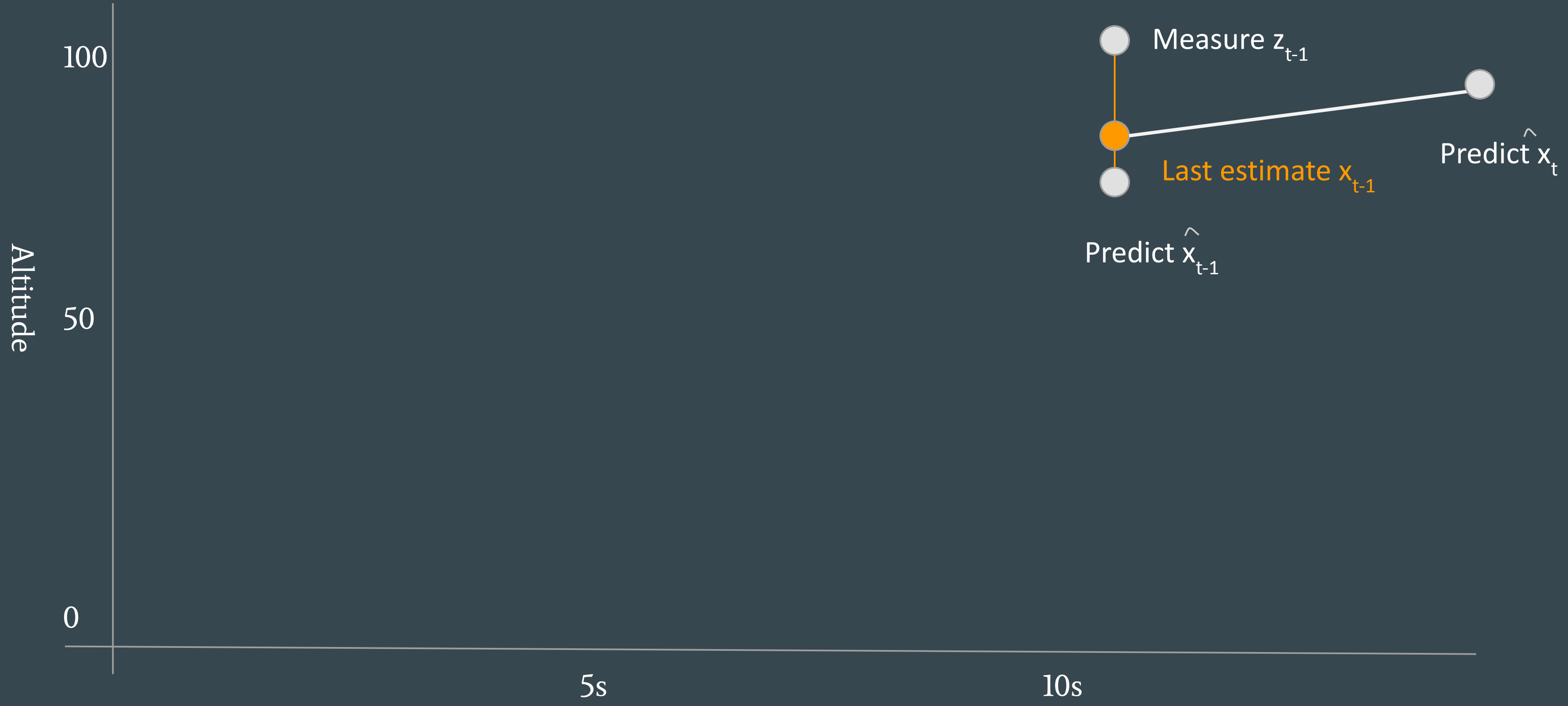
Blend prediction and measurement -- update



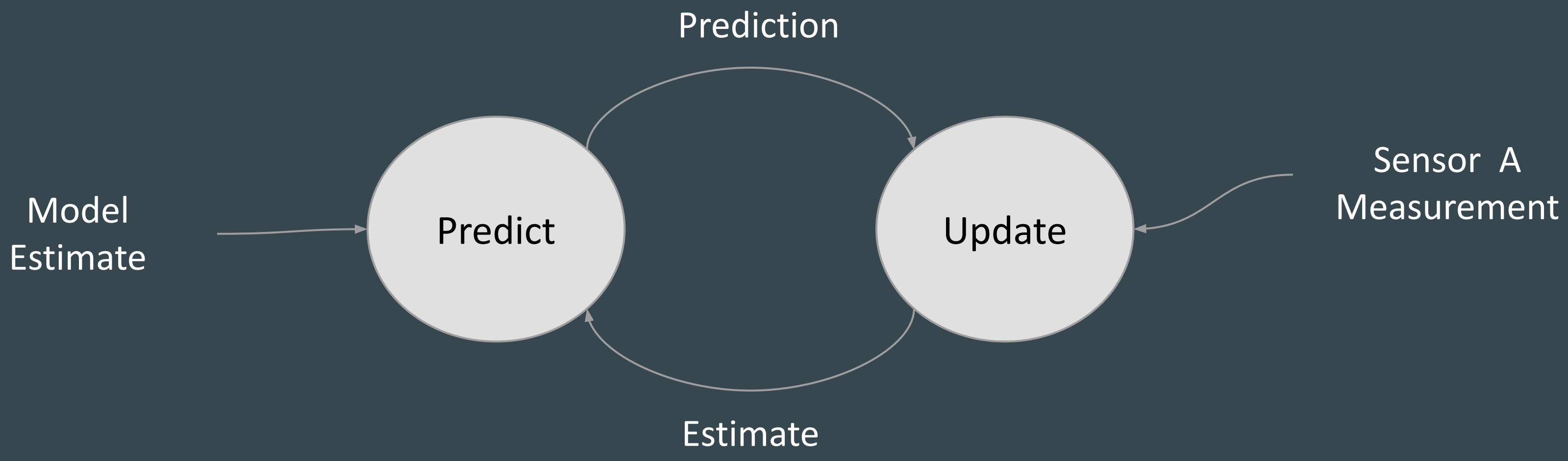
Estimate = α (measurement, prediction)



Estimate = α (measurement, prediction)



Estimate = α (measurement, prediction)



Estimate = α (measurement, prediction)

Algorithm

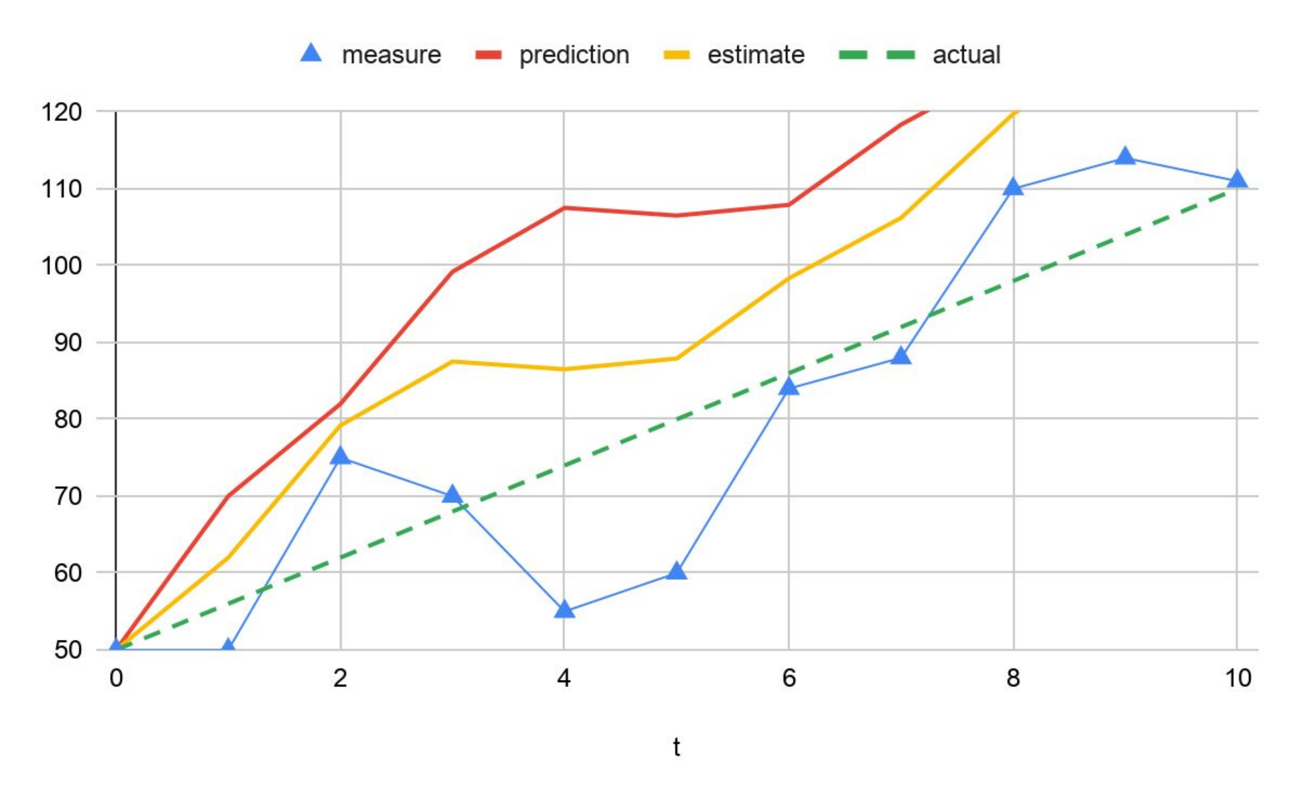
```
time_step = 1.0  # sec
scale_factor = 4.0/10

def predict_alt(estimated_alt, gain_rate):

    for z in sys.stdin:
        # predict
        predicted_alt = estimated_alt + gain_rate * time_step

        # update
        residual = z - predicted_alt
        estimated_alt = predicted_alt + scale_factor * residual

    return

# assume velocity of 5 m/s and initial altitude of 50m
predict_alt(50, 5)
```

Correct Model

Velocity in Z = 5 m/s

Altitude = Velocity in Z * t

First try



#Assume wrong model
gain_rate: **20** m/s (instead of 5)
initial altitude of 50m

time_step = 1.0
scale_factor = 4.0/10

prediction = estimated_alt + gain_rate * time_step
residual = z - predicted_alt
estimated_alt = predicted_alt + scale_factor * residual

@t1

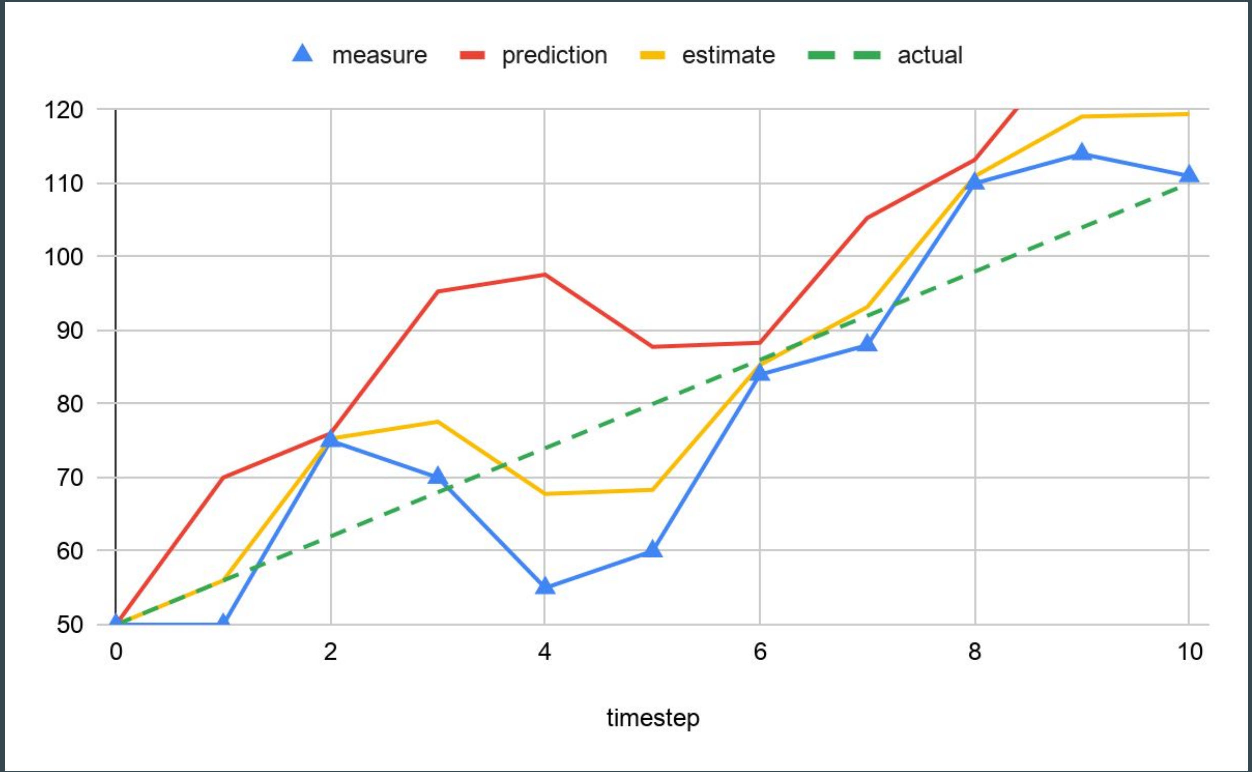
- Estimate between measure and prediction
- Bad model hurts

Second try

Correct Model

Velocity in Z = 5 m/s

Altitude = Velocity in Z * t



#Assume wrong model
gain_rate: **20** m/s (instead of 5)
initial altitude of 50m

time_step = 1.0
scale_factor = **7.0**/10 (biased weight)

Prediction = estimated_alt + gain_rate * time_step
residual = z - predicted_alt
estimated_alt = predicted_alt + scale_factor * residual

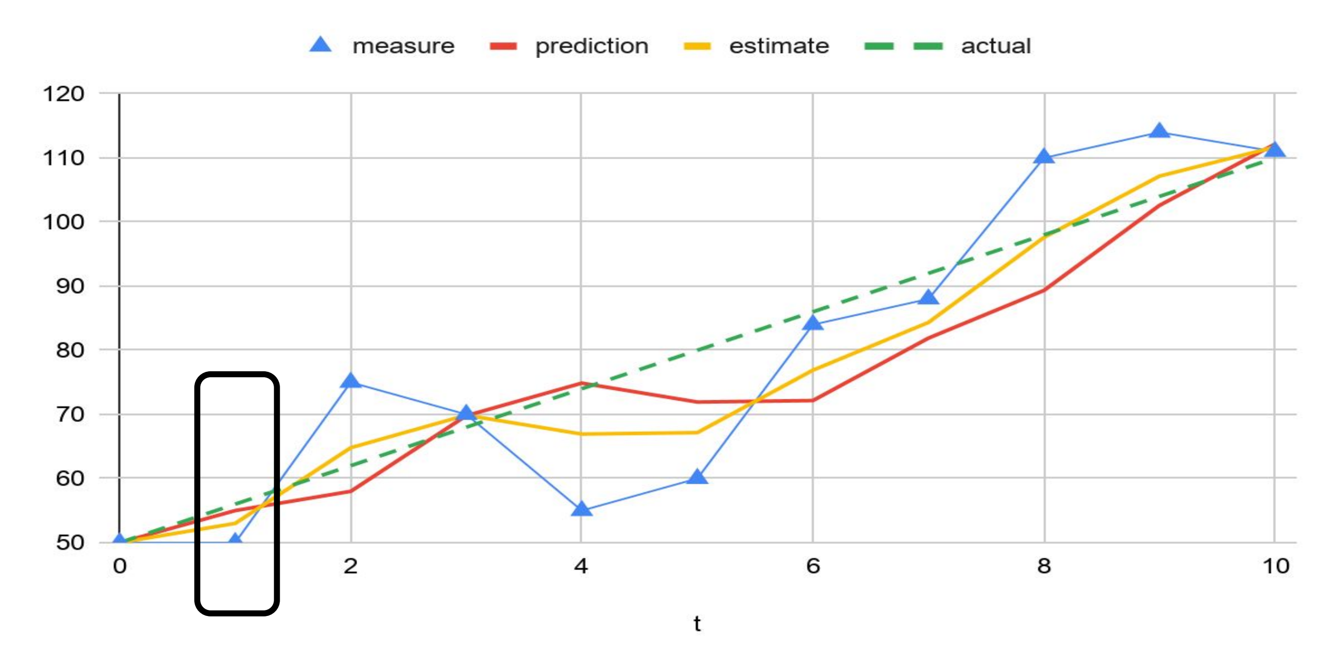
- Better: adjust blending process to favor sensor
- But we are reducing value of added model
- If measurements got worse, or sensor got broken, then?

Third try

Correct Model

Velocity in Z = 5 m/s

Altitude = Velocity in Z * t



gain_rate: 5 m/s - fixed model
initial altitude of 50m

time_step = 1.0
scale_factor = 4.0/10

Prediction = estimated_alt + gain_rate * time_step
residual = z - predicted_alt
estimated_alt = predicted_alt + scale_factor * residual

Get a better model

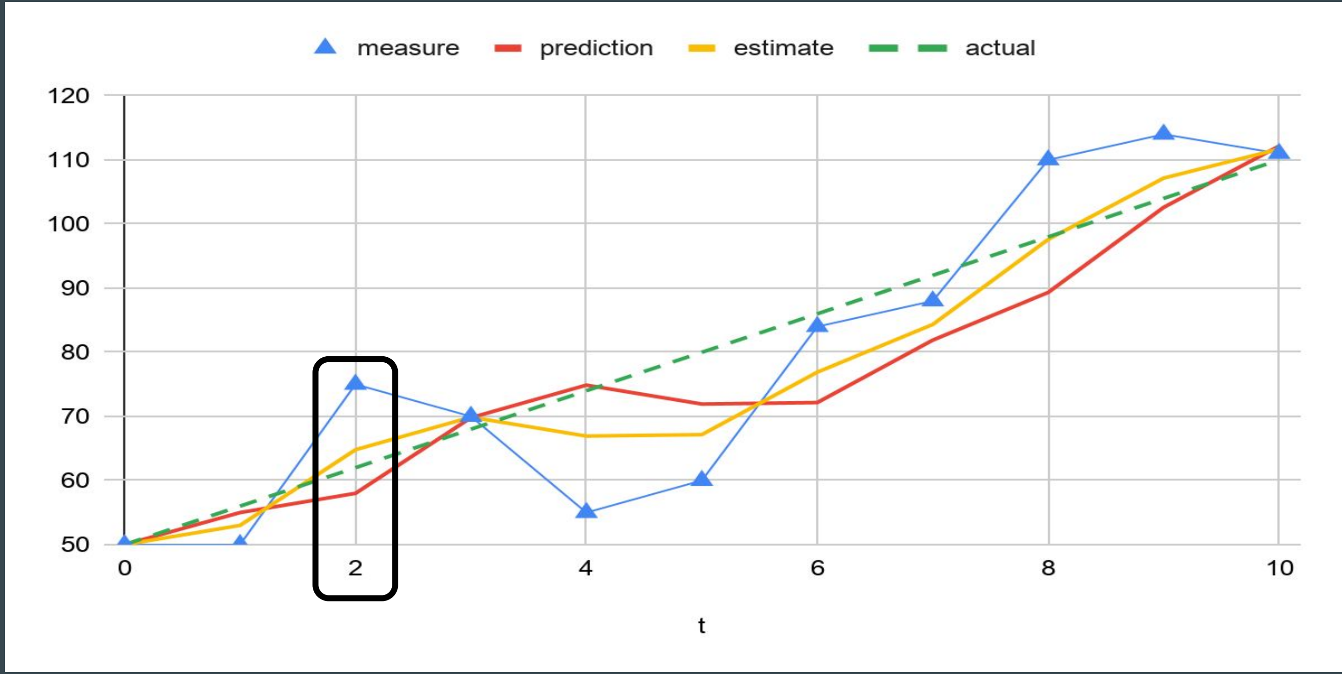
@t1

- good prediction
- estimate is closer than measurement to actual value

Great resource on this topic: Kalman and Bayesian Filters by R. Labbe

Third try

Correct Model
Velocity in Z = 5 m/s
Altitude = Velocity in Z * t



gain_rate: 5 m/s - fixed model
initial altitude of 50m

time_step = 1.0
scale_factor = 4.0/10

Prediction = estimated_alt + gain_rate * time_step
residual = z - predicted_alt
estimated_alt = predicted_alt + scale_factor * residual

Get a better model

@t2

- high measurement
- good prediction
- estimate is closer than both to actual value

Great resource on this topic: Kalman and Bayesian Filters by R. Labbe

Third try

Correct Model

Velocity in Z = 5 m/s

Altitude = Velocity in Z * t

velocity of 5 m/s
initial altitude of 50m

time_step = 1.0
scale_factor = 4.0/10

Prediction = estimated_alt + gain_rate * time_step

@t2

- high measurement
- good prediction
- estimate is closer than both to actual value

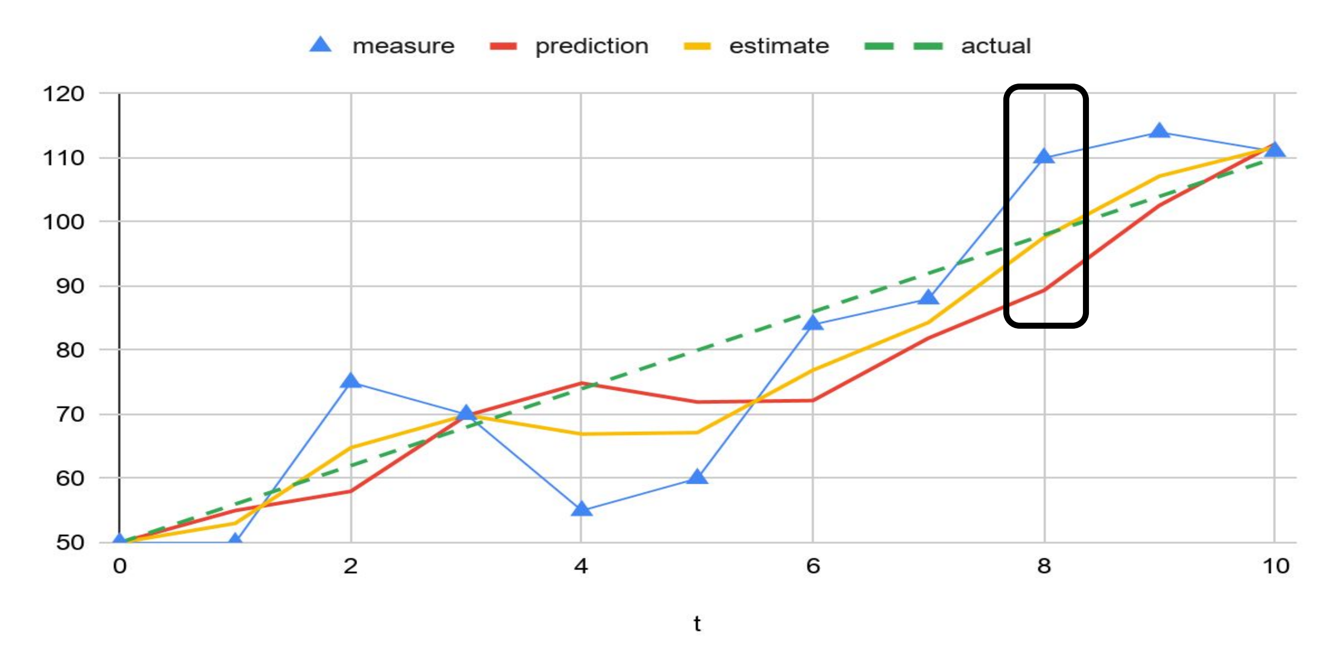
Great resource on this topic: Kalman and Bayesian Filters by R. Labbe

Third try

Correct Model

Velocity in Z = 5 m/s

Altitude = Velocity in Z * t



velocity of 5 m/s
initial altitude of 50m

time_step = 1.0
scale_factor = 4.0/10

Prediction = estimated_alt + gain_rate * time_step

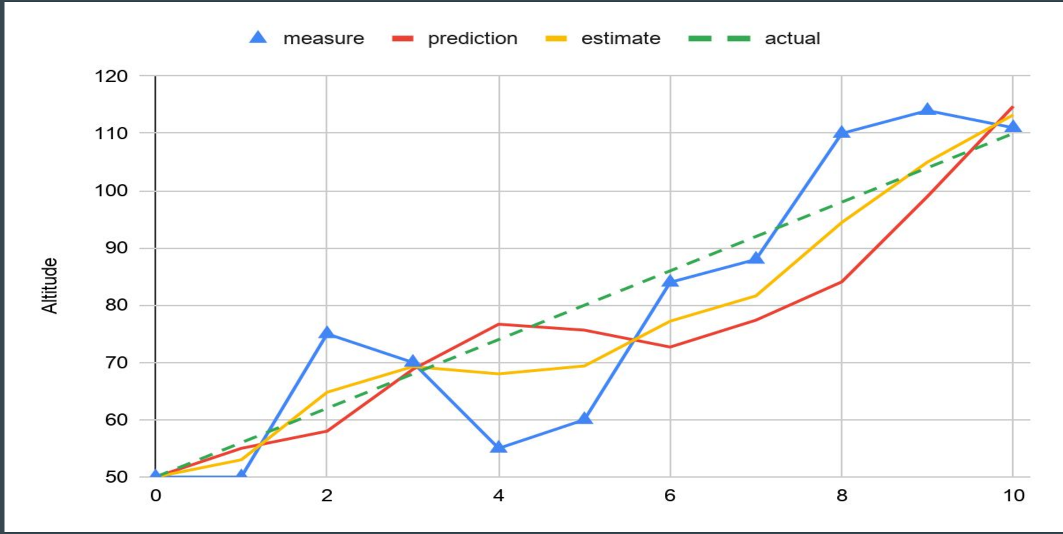
@t8

- High measurement
- Low prediction
- Estimate is closer to actual value

OR Revise Algorithm for adjusting model

```
time_step = 1.0
scale_factor= 4./10
gain_scale = 1./3

def predict_alt(estimated_alt, gain_rate):

    for z in sys.stdin:
        # predict
        predicted_alt = estimated_alt + gain_rate * time_step

        # update
        residual = z - predicted_alt
        estimated_alt  =  predicted_alt  +  scale_factor * residual

        # dynamically adjust gain rate according to residual changes
        gain_rate = gain_rate + gain_scale  * (residual/time_step)

    return

# assume velocity of 5 m/s and initial altitude of 50m
predict_alt(50, 5)
```

Final try



velocity of ~~5 m/s~~ Dynamic based on residual size
initial altitude of 50m

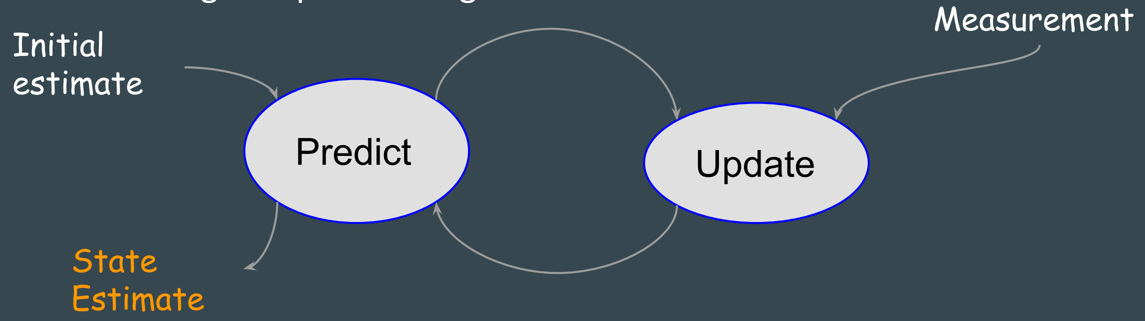
time_step = 1.0
scale_factor = 4.0/10 # H
gain_scale = 0.3 # G: How strongly to adjust predictions

Performance close as with tuned model

Much nicer… still a few magic blending numbers for G and H

Predictive Filters

- Predict next value & rate of change based on
  - Current estimate
  - Predict of how it will change
- Choose new scaling estimate between prediction and measurement
  - scales measurements
  - scales for changes in prediction gains



- Basis for Kalman filter

Takeaways

- Sensors capture data about robot and world state
- We cannot rely on sensors for perfect data
- To manage noise
  - Calibration
  - Fusion
  - Filtering