HMI
HUMAN MACHINE INTERACTION LABORATORY

ĐẠI HỌC CÔNG NGHỆ

DEAD RECKONING & SENSOR FUSION

Undergraduate course (Spring 2020)

Nguyen Do Van, PhD

HMI
HUMAN MACHINE INTERACTION LABORATORY

Part of the Lecture

- Introduction to Dead Reckoning
- Dead Reckoning Calculation
- Sensor Fusion
- Kalman Filter

HMI
HUMAN MACHINE INTERACTION LABORATORY

ĐẠI HỌC CÔNG NGHỆ

DEAD RECKONING INTRODUCTION

Dead Reckoning

- Estimates a position based on the change from a previous position
  - Estimate its position using only sensors that are available on the robot
- Does not require maps or outside references
  - Robot's location require maps or external references like GPS or signal strength
  - Perform even when the robot is in unfamiliar territory and is unable to communicate.
- Utilizes sensor data and the precise physical measurement of the robot:
  - Internal sensor to estimate distance of travel
  - Robot information: physical dimensions, size of its wheels, layout of the wheels

Dead Reckoning: Human

- No map or blind people: dead reckoning to navigate
- Estimate position and distance traveled:
  - Position: counting steps and multiplying by the size of each step
  - Distance traveled: how long they have been walking and how fast they typically walk
  - Difficulty with estimating direction with external supports.
- Path explained in dead reckoning system:
  - “Go five minutes down the road”
  - “Turn right then take ten steps”

Dead Reckoning: Robot

- Precise positioning: GPS and other external localization
  - GPS: accurate several meters
  - Car on road need more accurately position
- Additionally many robots operate in areas where there are limited environmental cues to aid in positioning
  - Underwater robots often have limited access to environmental markers or communication with other entities
    - They must be able to estimate their position using only the sensors available onboard.
  - Aerial robots are susceptible to gusts of wind that can change their position
    - A good dead reckoning system can help correct from these issues by constantly tracking the movement of the robot

Dead Reckoning: Robot

- Additionally many robots operate in areas where there are limited environmental cues to aid in positioning
  - Ground robots may be unable to connect to large sensor networks (like GPS) when they are in remote areas (caves, forests, Mars, etc).
  - In these cases dead reckoning is essential for the robot to continue moving while knowing where it is located
- Dead reckoning is a technique that is employed by robots in these scenarios.

When to Use Dead Reckoning

- When external references are noisy or imprecise
  - Conjunction with other techniques that incorporate environmental inputs
- When positioning systems (GPS) are unavailable or unreliable
  - Robot is in a cave or building then it may not be able to receive GPS or other signals to help find its location
  - In highly uniform environments (like underwater or in space), the robot may not be able to identify references so it cannot track its movement against another object

Sensor to Aid in Dead Reckoning

- Accelerometer: integrated with respect to time in order to establish robot velocity
- Gyroscope: detect and calculate changes in the robot's direction
- Compass: determine its direction (limited to two axis)
- Motor Encoders: track how far the wheels have rotated, which can be used to estimate the robot's displacement

Basic Dead Reckoning

- x_t is the position of the robot at the current time
- x_0 is the position of the robot at the previous step
- v is the velocity of the robot
- t is the duration between updates

$$x_t = x_0 + vt$$

- Problems:
  - Xo or V are not possible to know exactly.
  - System must predict from the previous position and velocity
  - The more accurately the robot knows its previous position and velocity, the more accurately it can estimate its current position.

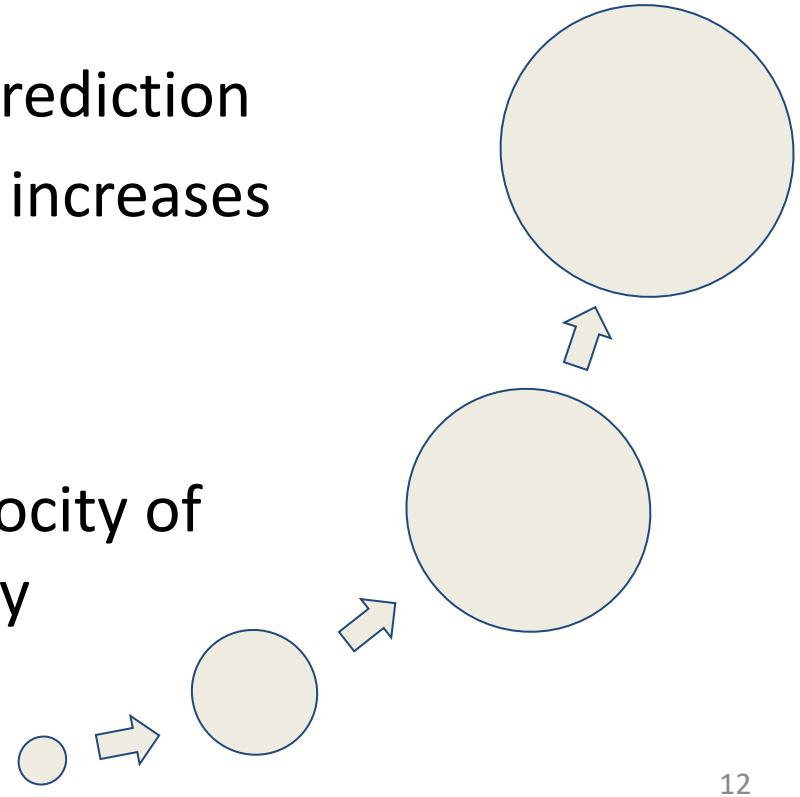
Probabilistic Dead Reckoning

- Previous position and velocity are not known precisely, they are often expressed as random variables
- Probability can be used to define the uncertainty of these measurements.

$$X_t = X_0 + Vt$$

Cumulative Errors

- Cumulative Errors:
  - Center of each circle is the prediction position at each time step
  - Size of the circle depicts the uncertainty of the prediction
  - As the robot continues to move, the uncertainty increases
- We may reduce errors
  - Ways of mitigating this increasing uncertainty
  - Fine-tuning the sensors used to estimate the velocity of the robot can significantly reduce the uncertainty



HMI
HUMAN MACHINE INTERACTION LABORATORY

ĐẠI HỌC CÔNG NGHỆ

DEAD RECKONING CALCULATION

Basic Kinematics

- Basic physics equations are used to update the position of the robot

$$x_t = x_0 + v(t - t_0) + \frac{1}{2}a(t - t_0)^2$$

- Sensors are used to compensate for environmental noise
  - However, dead-reckoning still requires the engineer to specify how the robot can move and how sensors indicate these changes in location
  - The x's, v's and a's can be replaced with vectors to specify the kinematics for a two or three dimensional location.

Timestep

- One of the important aspects of building a dead-reckoning system is to define how often the location is updated.
  - Smaller timesteps can yield more precise estimates
  - Smaller timesteps also produce noisier readings
  - Timesteps that are too large may miss important fluctuations
- Instead of choosing an arbitrary time duration:
  - Iterate over several timesteps and measure the accuracy of the robot's positioning.
  - Choose the time duration that was the most accurate.
- Measurement rate of your sensors, since limitation by how frequently you receive data

Using Accelerometers

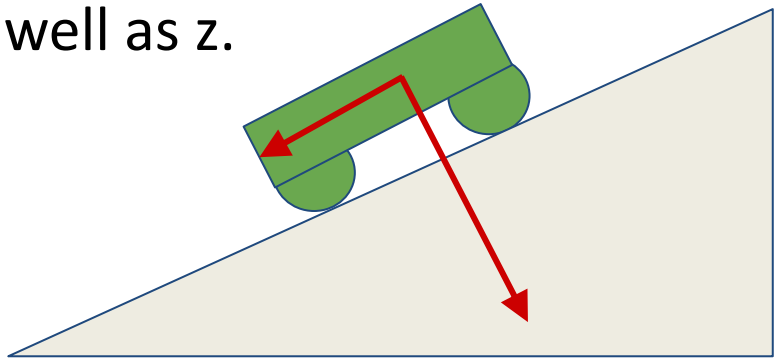
- Accelerometer readings can be used to estimate velocity
- Velocity estimates are then used to estimate displacement
- The state of the robot is both the current velocity and the current position

$$v_t = v_0 + a(t - t_0)$$

- Check and adjust:
  - Units of accelerometers
  - Location of accelerometer if it is not in the center of the robot

Accelerometer Issues

- Velocity estimates from acceleration integration can drift over time
  - More rapid updates to the velocity can help mitigate the drift of velocity values
  - Combining accelerometers with other sensors (like encoders) can provide a more accurate estimate of velocity and position
- Care must be taken when use accelerometers on slopes to account for gravity
  - When a robot is on a slope, the acceleration due to gravity can be measured across both compass axis (x and y) as well as z.



HMI
HUMAN MACHINE INTERACTION LABORATORY

Using Gyroscopes

- Used in many dead-reckoning systems because they provide mostly accurate orientation information in many environments.
  - Gyroscopes are essential for stabilizing and orienting drones and flying robots where since there are no wheel encoders
- Gyroscopes and accelerometers can be used in conjunction
  - Refine velocity and orientation estimates that can compensate for sloped ground
- Limitation of gyroscopes:
  - Do not detect linear motion (only angular rotations)
  - Must be paired with other sensors
  - Gyroscopes are also known to drift over time, so their estimates can be less accurate as the robot continues to move.

HMI
HUMAN MACHINE INTERACTION LABORATORY

Using Compasses

- Like a gyroscope, a compass measures the robot's orientation
- Gyroscopes typically measures orientation in 3d, while a compass measures orientation in 2d.
- Since a compass provides absolute orientation, it's reliability does not degrade over time.
- Like a gyroscope a compass must be used in conjunction to another sensor that measure linear velocity

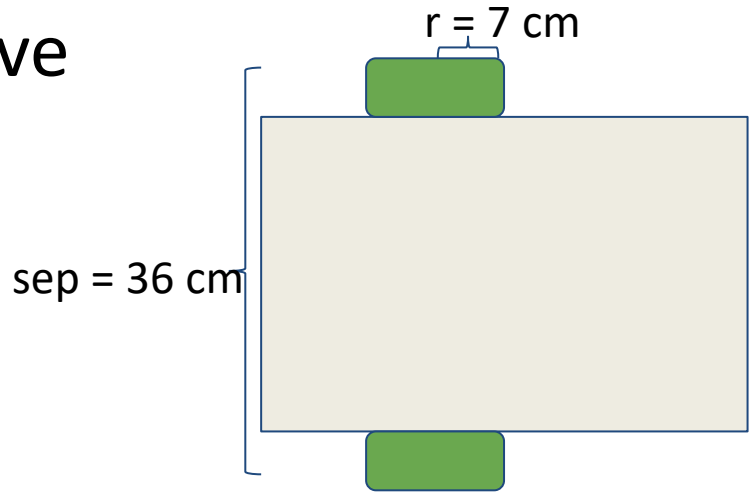
HMI
HUMAN MACHINE INTERACTION LABORATORY

Using Encoders

- Common choice for wheeled robots that use dead-reckoning
  - Reliable and inexpensive
  - Easiest way to estimate the robot's velocity
- Care must be taken
  - Detect free-spinning wheels or wheel slippage, which can cause encoder data to not accurately represent velocity
  - Size and position of the wheels must be known in order to use encoders for estimating velocity
- Differential drive robots (robots with two independent drive wheels) can easily use encoders
- Bicycle-type robots (front wheels can rotate) require more advanced physics calculations and must incorporate the direction of the front wheel

Differential Drive

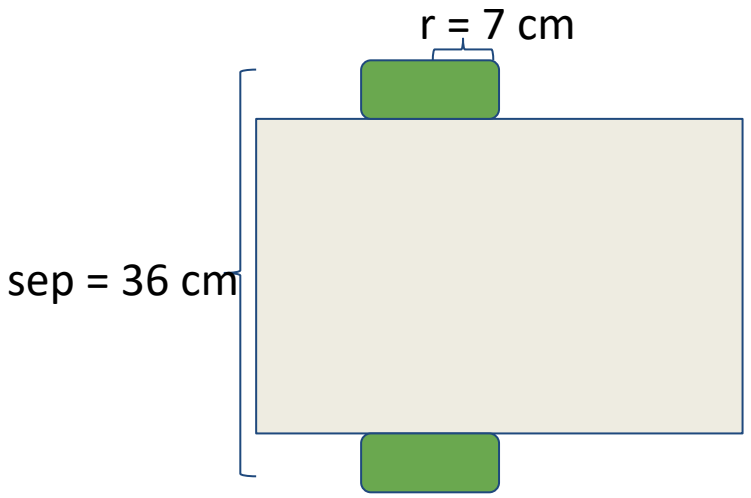
- The two motorized wheels in a differential drive robot are independent and are fixed on the sides of the robot
- The wheels can only spin, their orientation cannot change
- Many small mobile robots are differential drive because the kinematics equations are simple and there is no need for an additional directional motor
- In order to balance, most differential drive robots have a caster wheel that is non-motorized and spins freely



Differential Drive Equations

- update the location of the robot as well as its orientation
  - r is the radius of wheels
  - u_l is the angular velocity of the left wheel
  - u_r is the angular velocity of the right wheel
  - L is the distance of the wheels

$$\dot{x} = \frac{r}{2}(u_l + u_r)\cos\theta$$
$$\dot{y} = \frac{r}{2}(u_l + u_r)\sin\theta$$
$$\dot{\theta} = \frac{r}{L}(u_r - u_l).$$



HMI
HUMAN MACHINE INTERACTION LABORATORY

ĐẠI HỌC CÔNG NGHỆ

SENSOR FUSION AND KALMAN FILTER

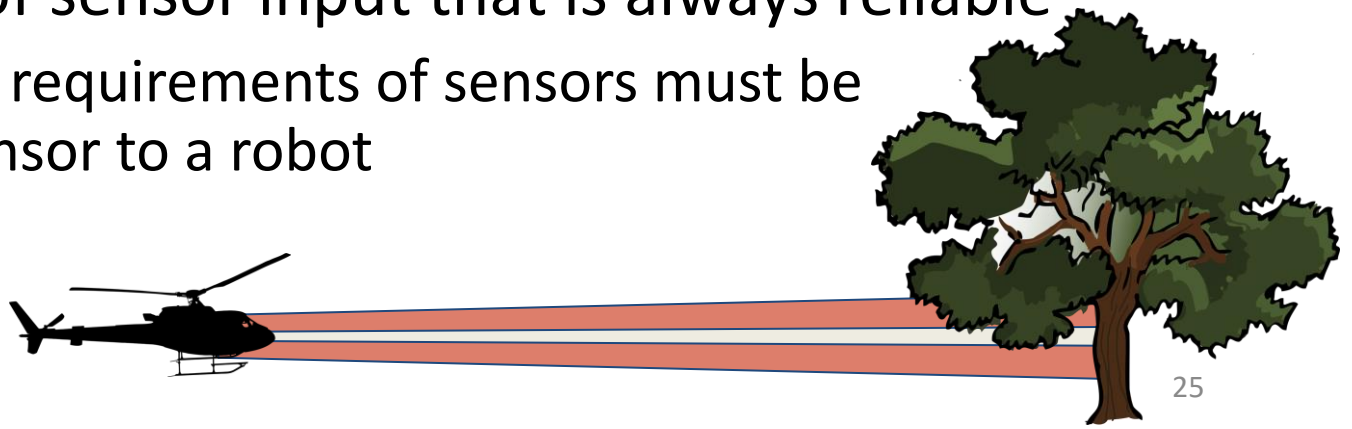
HMI
HUMAN MACHINE INTERACTION LABORATORY

Sensor Fusion

- Each type of sensor offers different information about the environment
- Combining data from multiple sensors improves the ability of a robot to understand the world
  - Compass and gyroscope for dead-reckoning: Combine with Accelerometer to determine linear velocity
- Sometimes sensors disagree, so engineers must design ways to reconcile different inputs.
  - Some sensors may have high uncertainty
  - Detecting faults in sensors is made easier when multiple sensors are used because errant measurements will be more obvious.

Example: Collision Avoidance

- Essential to the operation of most autonomous mobile robots
  - More information surroundings the more likely robots can avoid collisions
- Designing a robot to include sensors that are complimentary
  - Some sensors are useful during certain situations while other sensors are a useful during other conditions
  - At night or in foggy conditions, Lidar and radar may be more effective
  - During the daytime, cameras may provide more accurate information
- Rarely will there be a single type of sensor input that is always reliable
  - However, the cost, size, and power requirements of sensors must be considered before adding every sensor to a robot



Explicit Sensor Fusion

- One solution to sensor fusion is writing rules based on multiple sensors

Example: Obstacle Avoidance

```
if (sonarValue < 10 && cameraDetectsObject) {
    avoidObstacle();
}
```

Example: Localization

```
x_pos = ((x_pos + vel_x * time_diff) + gps_x) / 2
y_pos = ((y_pos + vel_y * time_diff) + gps_y) / 2
```

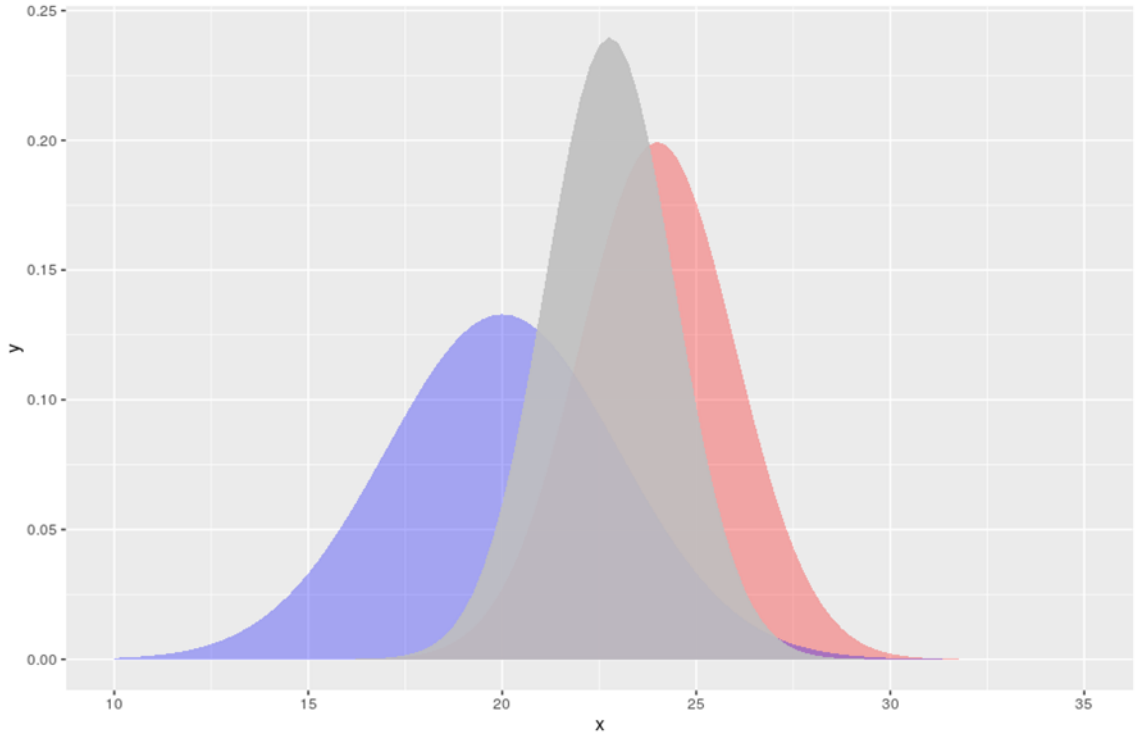
- Explicit rules for combining sensors can be useful for simple sensor data or specific cases where the expected behavior is known.
- However, code will become unmaintainable and complex if too many rules and conditionals are used to fuse sensor data.

Probabilistic Sensor Fusion

Sensor A (red)
u = 23
v = 4

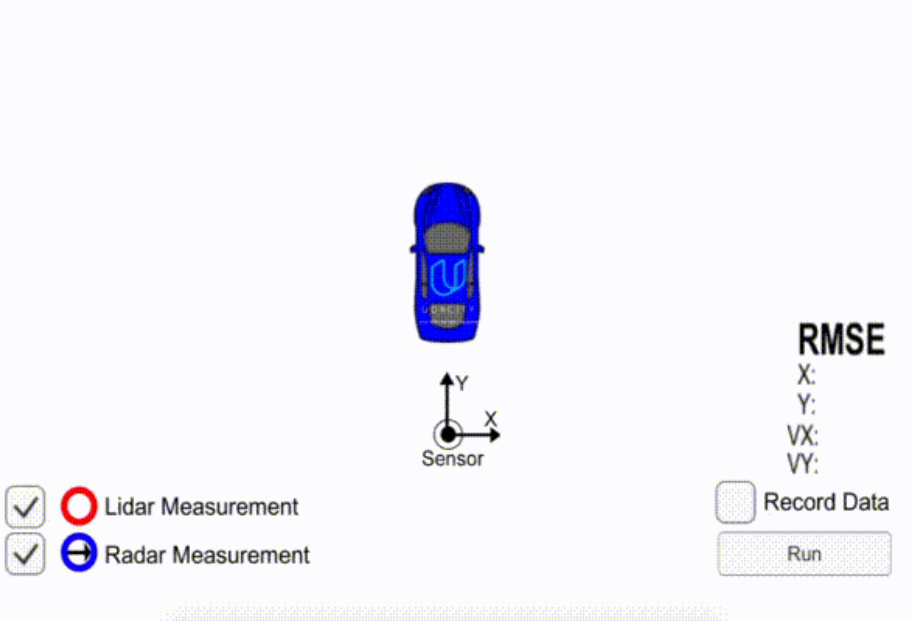
Sensor B (blue)
u = 20
v = 9

Combined Sensor (gray)
u = 22.77
v = 2.77



Kalman Filters

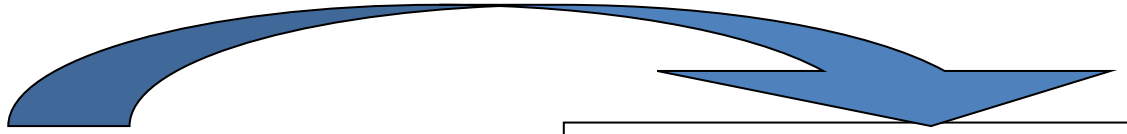
- Kalman filter is an algorithm that can be applied to automate sensor fusion.
- Kalman filters produce a probabilistic estimate of a state based on the past state, predicted changes, and sensor updates
- The state predicted by a Kalman filter can be anything that the robot needs to know (position, velocity, proximity of obstacles, etc.)



Steps of a Kalman Filter

- **Predict**: the first step is to estimate change in the state based on the previous state and inputs:
  - Dependently of the sensor data
  - Based on previous state estimation and expected transition of the state
- **Update**: the second step is refine that state estimate with the measured data
  - Incorporate sensor data to change previously estimated state to more closely with sensor data
- Result of the Kalman filter will be a weighted combination of the state prediction from the expected state transition and the state estimated by the measurement data

Steps of a Kalman Filter



Prediction (Time Update)

(1) Project the state ahead

$$x^-_k = Ax_{k-1} + Bu_k$$

(2) Project the error covariance ahead

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

Correction (Measurement Update)

(1) Compute the Kalman Gain

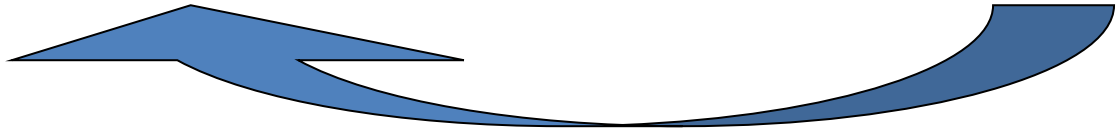
$$K = P^-_k H^T (HP^-_k H^T + R)^{-1}$$

(2) Update estimate with measurement z_k

$$x_k = x^-_k + K(z_k - H\,x^-_k)$$

(3) Update Error Covariance

$$P_k = (I - KH)P^-_k$$



Predicting: Estimate the state

- The state (x^-_k) is estimated by applying a state transition matrix (A) to the previous state (x_{k-1})
- Another transition matrix (B) is applied to the input control vector (u)

$$x^-_k = Ax_{k-1} + Bu_k$$

Predicting: Estimate the Covariance

- The covariance (P^-_k) is estimated by multiplying the state transition matrix (A) times the previous covariance (P_{k-1}) times the transpose of the transition matrix
- Q, the process noise covariance, contributes to the overall uncertainty. When Q is large, the Kalman Filter tracks large changes in the data more closely than for smaller Q
- Conceptually covariance is a measure of the uncertainty of the estimate

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

Kalman Gain

- Kalman Gain is calculated at each iteration of the Kalman filter based on the measurement estimation matrix (H), the state covariance, and the covariance of the measurement vector.
- A high Kalman gain will increase the impact of the measurements because the measurements have high certainty
- A low Kalman gain will mostly ignore the measurement data

$$K = P^{-}_{k}H^{T}(HP^{-}_{k}H^{T} + R)^{-1}$$

- R, the measurement noise covariance, determines how much information from the measurement is used.
  - If R is high, the Kalman Filter considers the measurements as not very accurate.
  - For smaller R it will follow the measurements more closely.

Updating: Refine the State

- The measurement estimation matrix (H) is multiplied by the estimated state (x_k) to produce an estimate of the measurement.
- The difference between the expected measurement and the actual measurement vector (z_k) is multiplied Kalman Gain (K) to refine the state estimate.
- Essentially this updates the state based on the measurement data collected.

$$x_k = x^-_k + K(z_k - H\,x^-_k)$$

Updating: Refine the Covariance

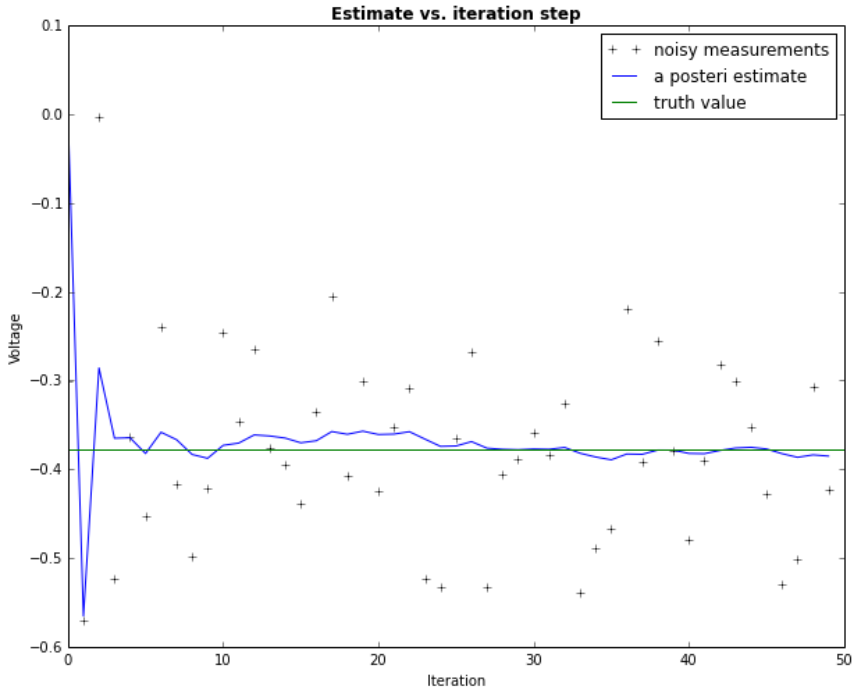
- Finally the covariance matrix (P_k) is updated by subtracting the multiplication if the Kalman Gain (K), the measurement estimation matrix (H) and the original estimated covariance (P_k).

$$P_k = (I - KH)P^-_k$$

- The update of the covariance increase the uncertainty of the components of the state that were most changed by the measurement data.
- If the Kalman gain is zero, then the state was not changed by the measurements so the covariance does not change

Simple Example

```
# intial parameters
n_iter = 50
sz = (n_iter,) # size of array
x = -0.37727 # truth value (typo in example at top of p. 13 calls this z)
z = np.random.normal(x,0.1,size=sz) # observations (normal about x, sigma=0.1)

Q = 1e-5 # process variance

# allocate space for arrays
xhat=np.zeros(sz)      # a posteri estimate of x
P=np.zeros(sz)         # a posteri error estimate
xhatminus=np.zeros(sz) # a priori estimate of x
Pminus=np.zeros(sz)    # a priori error estimate
K=np.zeros(sz)         # gain or blending factor

R = 0.1**2 # estimate of measurement variance, change to see effect

# intial guesses
xhat[0] = 0.0
P[0] = 1.0

for k in range(1,n_iter):
    # time update
    xhatminus[k] = xhat[k-1]
    Pminus[k] = P[k-1]+Q

    # measurement update
    K[k] = Pminus[k]/( Pminus[k]+R )
    xhat[k] = xhatminus[k]+K[k]*(z[k]-xhatminus[k])
    P[k] = (1-K[k])*Pminus[k]
```



http://scipy-cookbook.readthedocs.io/items/KalmanFiltering.html

HMI
HUMAN MACHINE INTERACTION LABORATORY

ĐẠI HỌC CÔNG NGHỆ

QUESTION?