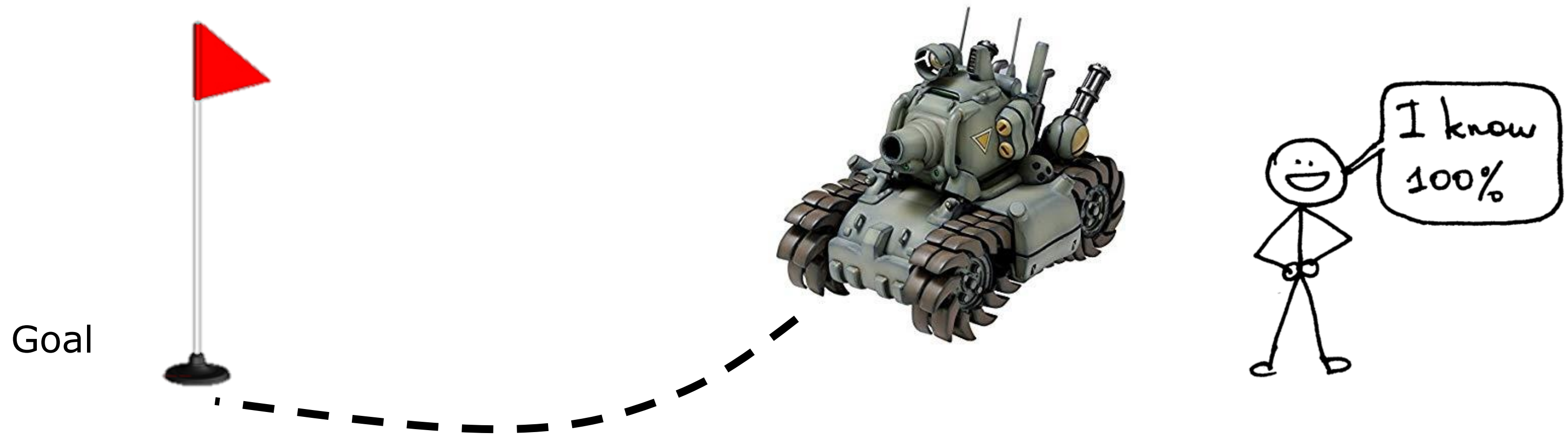
Mobile Robot Kinematics Lecture 3

Jeong-Yean Yang

2020/10/22

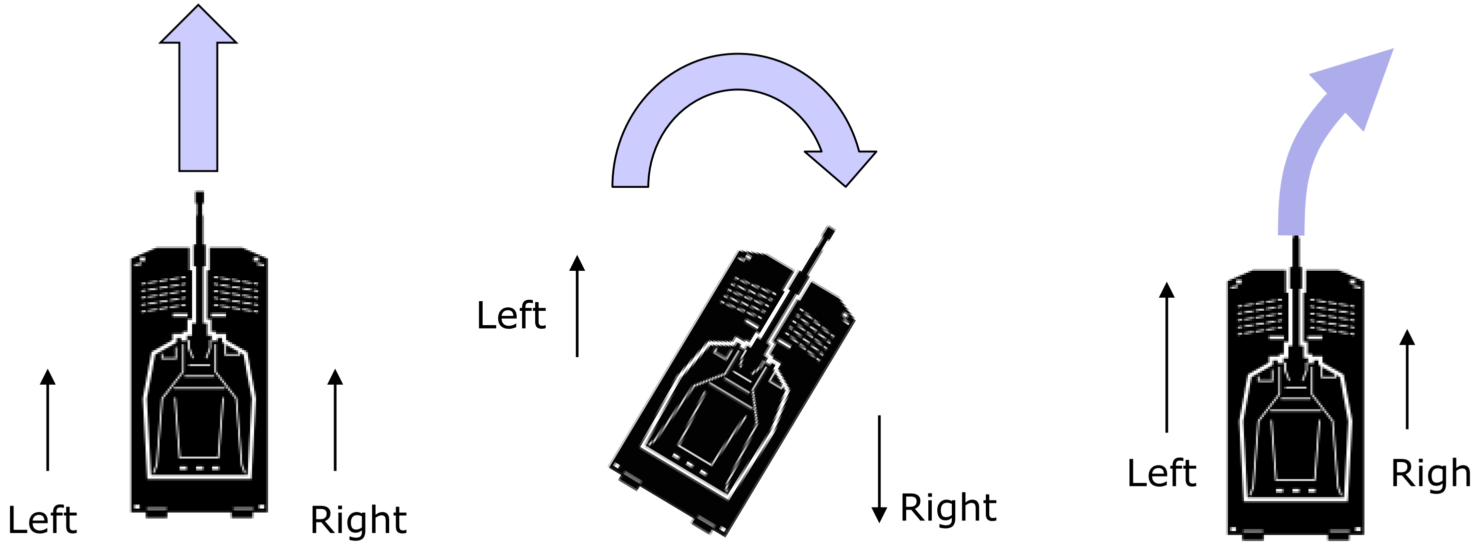
1 Mobile Robot Kinematics

How to Drive a Mobile Robot?



- Mobile Robot is very familiar with Everybody.
  - We have played Plamodel Tank or Game.
- But, in Robotics, we know PD or PID Control.
  - What will be the Desired Angle for approaching the goal?
  - It is tough…

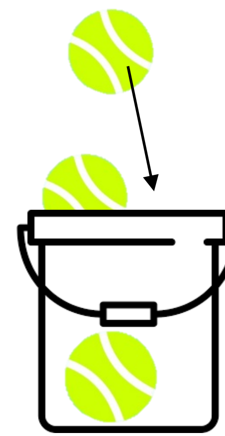
Control in Mobile Robots



- Control with Joystick is Easy → Visual Feedback
- Control with CPU for Moving → No Feedback
  - It means that we must know Complex Kinematics

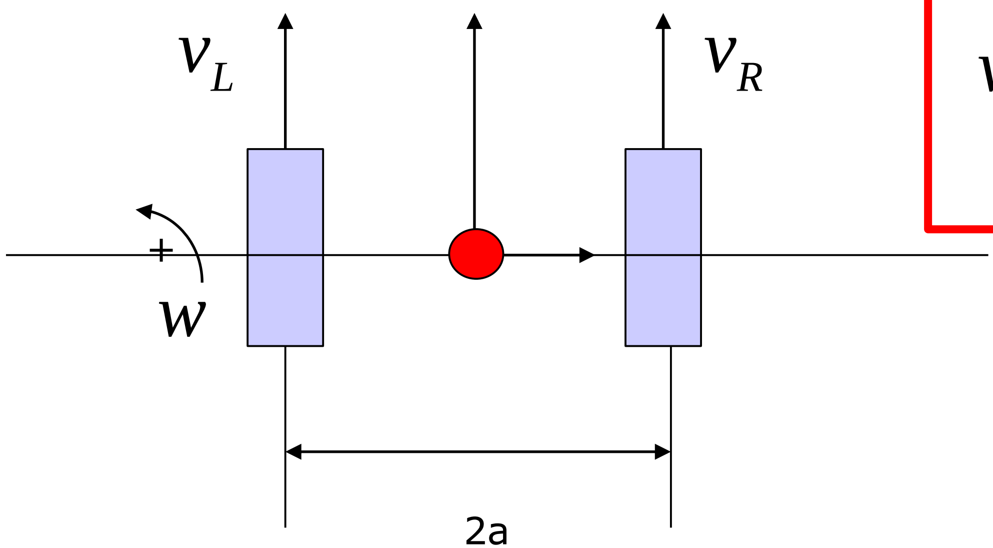
Problems of Tennibot

Tennibot

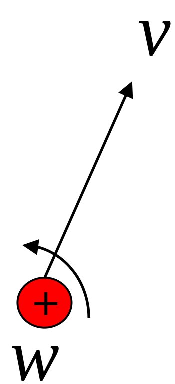


- Ball Detection → Ball Position → Robot Moves
  - How tennibot moves?
- Where is a Bucket after collecting balls?

Differential Drive Kinematics

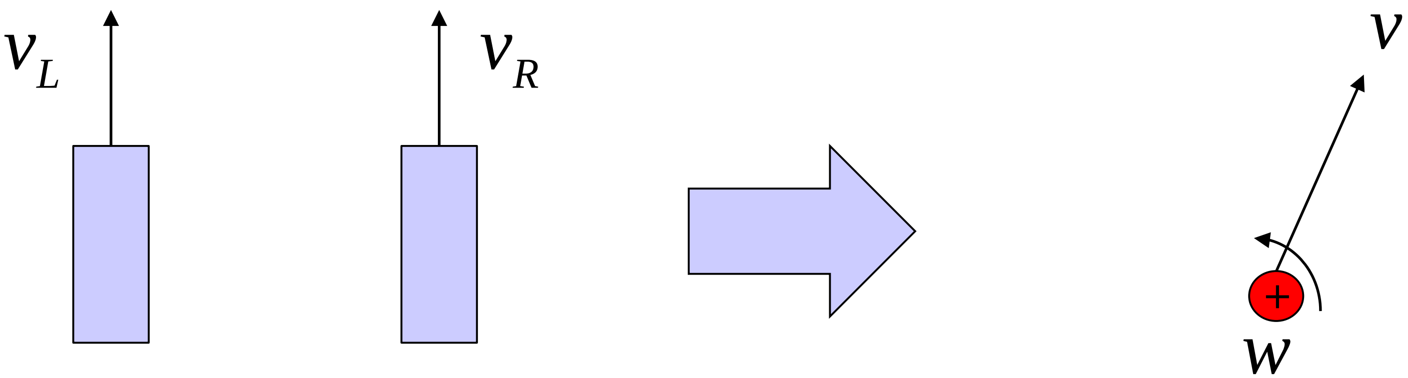


$$v = \frac{v_L + v_R}{2}$$

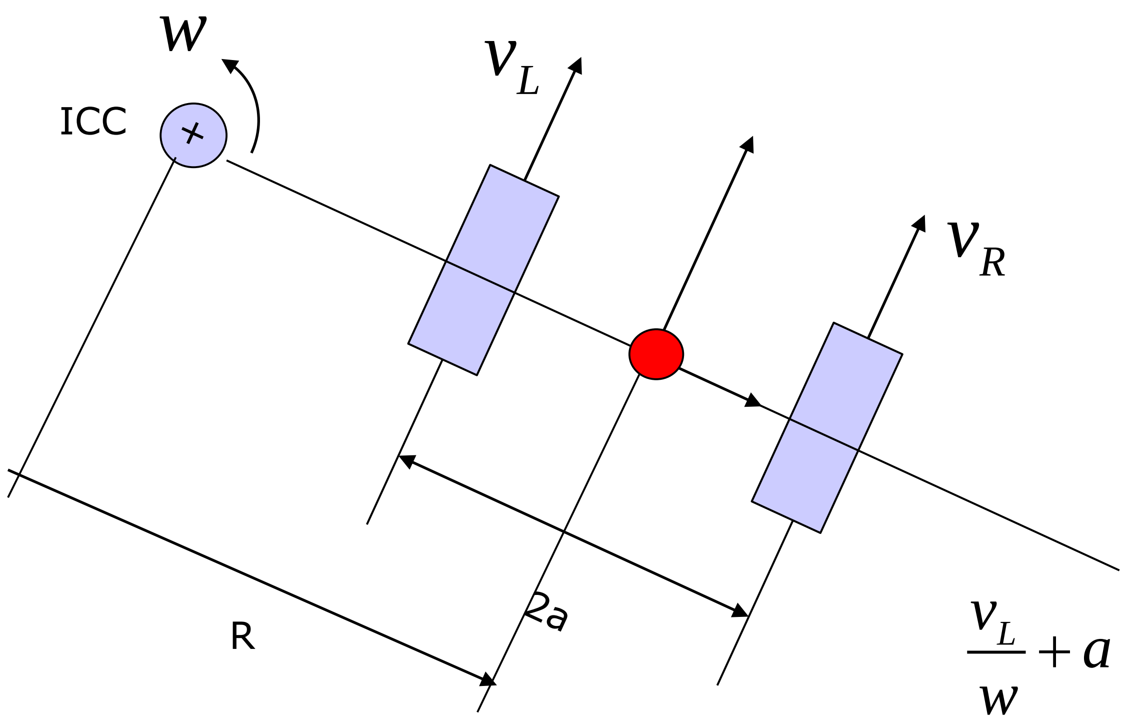


w: Angular velocity
vL: left velocity
vR: Right velocity

Question: Velocity of Red mark ● ?
when vL, vR is given.


v_L
v_R
v
w

Differential Drive Kinematics



Remind that

$$v = w \times r$$

$$\frac{v_L}{w} + a = \frac{v_R}{w} - a = R$$

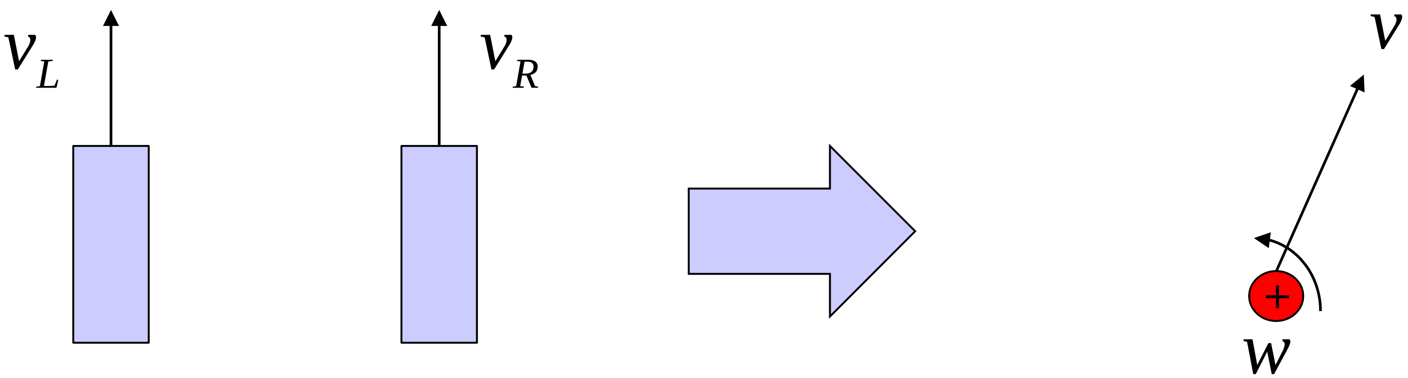
ICC: Instantaneous
Center of curvature

$$v_L = w(R - a)$$

$$v_R = w(R + a)$$

$$2a = \frac{v_R - v_L}{w}$$

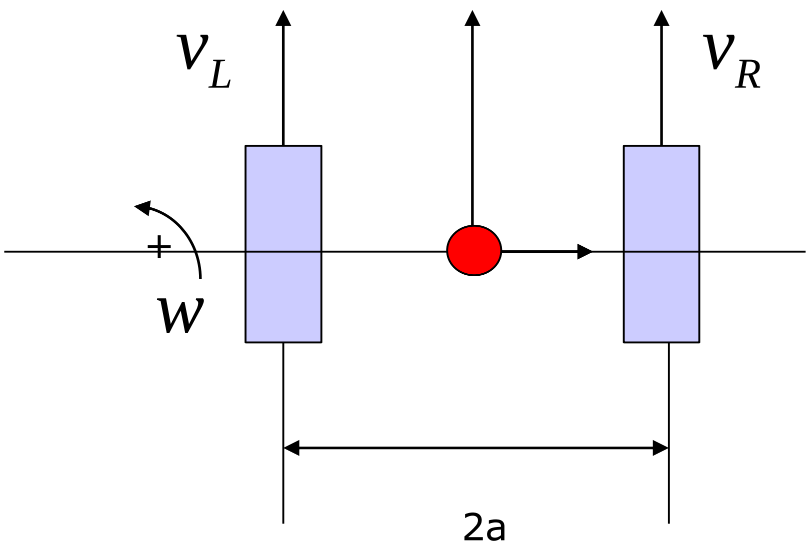
$$\therefore w = \frac{v_R - v_L}{2a}$$



$$w = \frac{v_R - v_L}{2a}$$

$$v = \frac{v_L + v_R}{2}$$

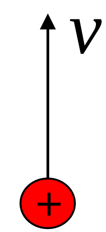
Differential Drive Kinematics



$$w = \frac{v_R - v_L}{2a}$$

$$v = \frac{v_L + v_R}{2}$$

$$case\ 1)\ w = \frac{v_R - v_L}{2a} = 0$$

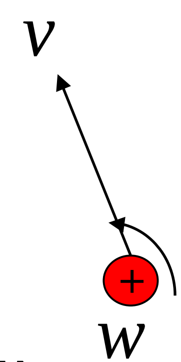


$$v = v_R = v_L$$

Linear motion

$$case\ 2)\ v_L = -v_R$$

$$w = \frac{v_R}{a}, v = \frac{v_R + v_L}{2} = 0$$



$$case\ 3)\ v_L = 0$$

$$w = \frac{v_R}{2a}, v = \frac{v_R + v_L}{2} = \frac{v_R}{2}$$

$$v_L = w(R-a) \qquad 2a = \frac{v_R - v_L}{w}$$

$$v_R = w(R+a) \qquad \therefore w = \frac{v_R - v_L}{2a}$$

$$v_L = w(R-a) = \frac{v_R - v_L}{2a}(R-a)$$

$$2av_L = (v_R - v_L)(R-a)$$

$$\therefore R = a + \frac{2av_L}{v_R - v_L}$$

$$= a\frac{v_R + v_L}{v_R - v_L}$$

$$v = \frac{v_L + v_R}{2}$$

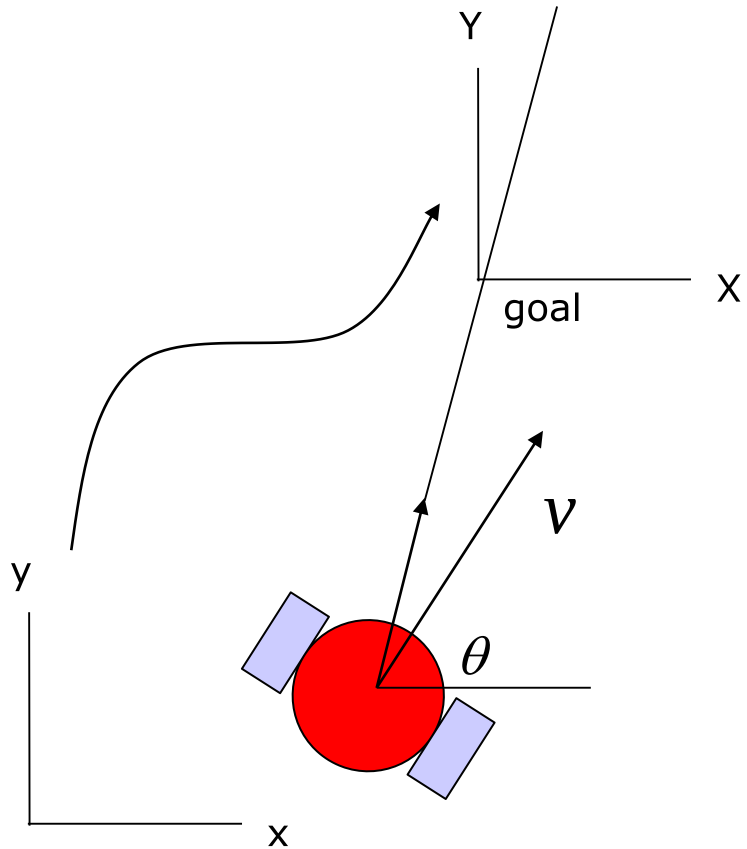
$$\left(= wR = \frac{v_R - v_L}{2a} a \frac{v_R + v_L}{v_R - v_L} = \frac{v_L + v_R}{2}\right)$$

Differential Drive Kinematics

$$\begin{pmatrix} v \\ w \end{pmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{-1}{2a} & \frac{1}{2a} \end{bmatrix} \begin{pmatrix} v_L \\ v_R \end{pmatrix} = r \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{-1}{2a} & \frac{1}{2a} \end{bmatrix} \begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

$$\begin{pmatrix} w_L \\ w_R \end{pmatrix} = \begin{bmatrix} \frac{1}{r} & \frac{-a}{r} \\ \frac{1}{r} & \frac{a}{r} \end{bmatrix} \begin{pmatrix} v \\ w \end{pmatrix}$$

Kinematic Position



$$v \begin{cases} v_x = v\cos\theta \\ v_y = v\sin\theta \end{cases}$$

$$X = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \Rightarrow X_g = \begin{pmatrix} x_g \\ y_g \\ \theta_g \end{pmatrix}$$

$$\dot{X} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v\cos\theta \\ v\sin\theta \\ w \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix}$$

Jacobian Matrix

$$\dot{X} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v\cos\theta \\ v\sin\theta \\ w \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{v_L + v_R}{2} \\ \frac{v_R - v_L}{2a} \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{-1}{2a} & \frac{1}{2a} \end{pmatrix} \begin{pmatrix} v_L \\ v_R \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -\frac{1}{a} & \frac{1}{a} \end{pmatrix} \begin{pmatrix} v_L \\ v_R \end{pmatrix}$$

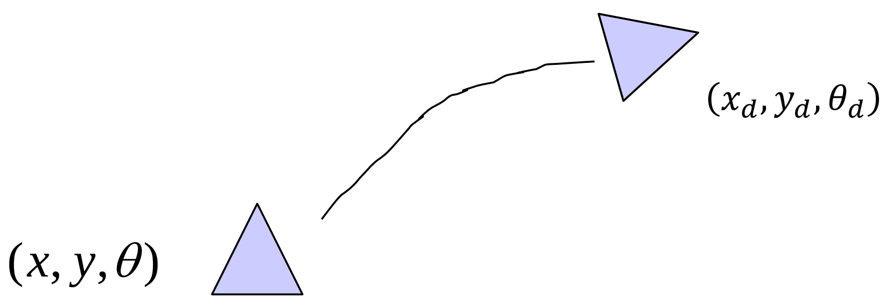
r: wheel radius

$$= \frac{1}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -\frac{1}{a} & \frac{1}{a} \end{pmatrix} \begin{pmatrix} rw_L \\ rw_R \end{pmatrix} = \frac{r}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ \frac{-1}{a} & \frac{1}{a} \end{pmatrix} \begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

Jacobian for Cartesian Control

$$\dot{X} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \frac{r}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -1/a & 1/a \end{pmatrix} \begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

r: wheel radius
wL : left wheel angular velocity
wR : right wheel angular velocity



- Our goal is from x,y,q → xd,yd,qd
- Inverse of Jacobian matrix is required.
- But, J = [3x2]… how to find inverse matrix?

Kinematic Position

- Remind that x, y are defined at general coordinate

$$\dot{X} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \frac{r}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -1/a & 1/a \end{pmatrix} \begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

$$\Rightarrow \quad dX = \frac{r}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -1/a & 1/a \end{pmatrix} \begin{pmatrix} d\theta_L \\ d\theta_R \end{pmatrix} = J \begin{pmatrix} d\theta_L \\ d\theta_R \end{pmatrix} = Jd\Theta$$

- But, in many cases localization is difficult.

  * Localization: knowing where it is.

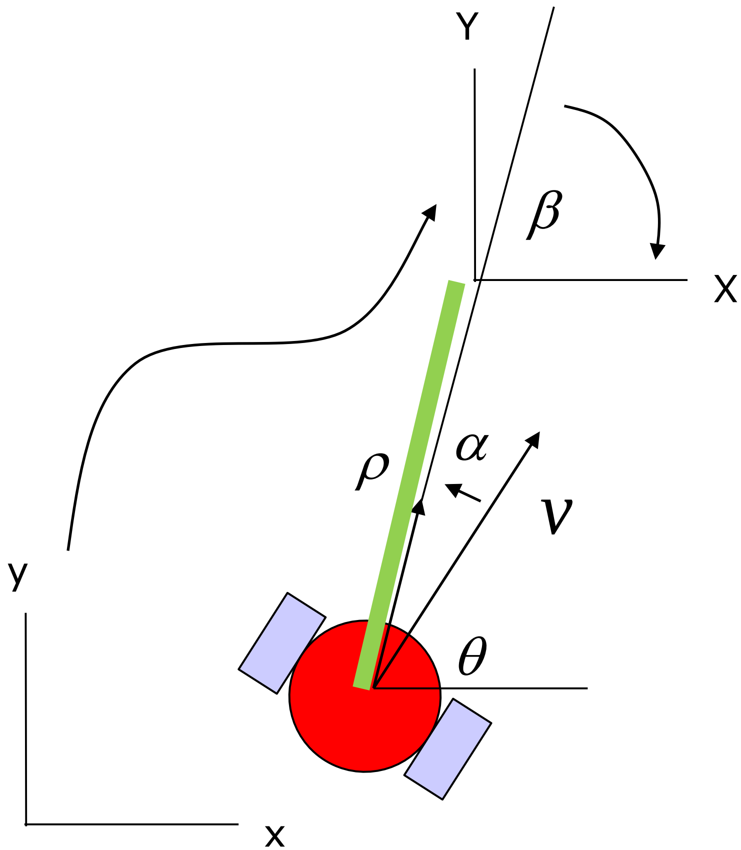
- New transform is required. → Polar coordinate

Polar Coordinate

$$\rho = \sqrt{(X-x)^2 + (Y-y)^2} = \sqrt{\Delta x^2 + \Delta y^2}$$

$$\alpha = a\tan 2(\Delta y, \Delta x) - \theta$$

$$\beta = -a\tan 2(\Delta y, \Delta x) = -\theta - \alpha$$



$$\dot{\rho} = v\cos\alpha$$

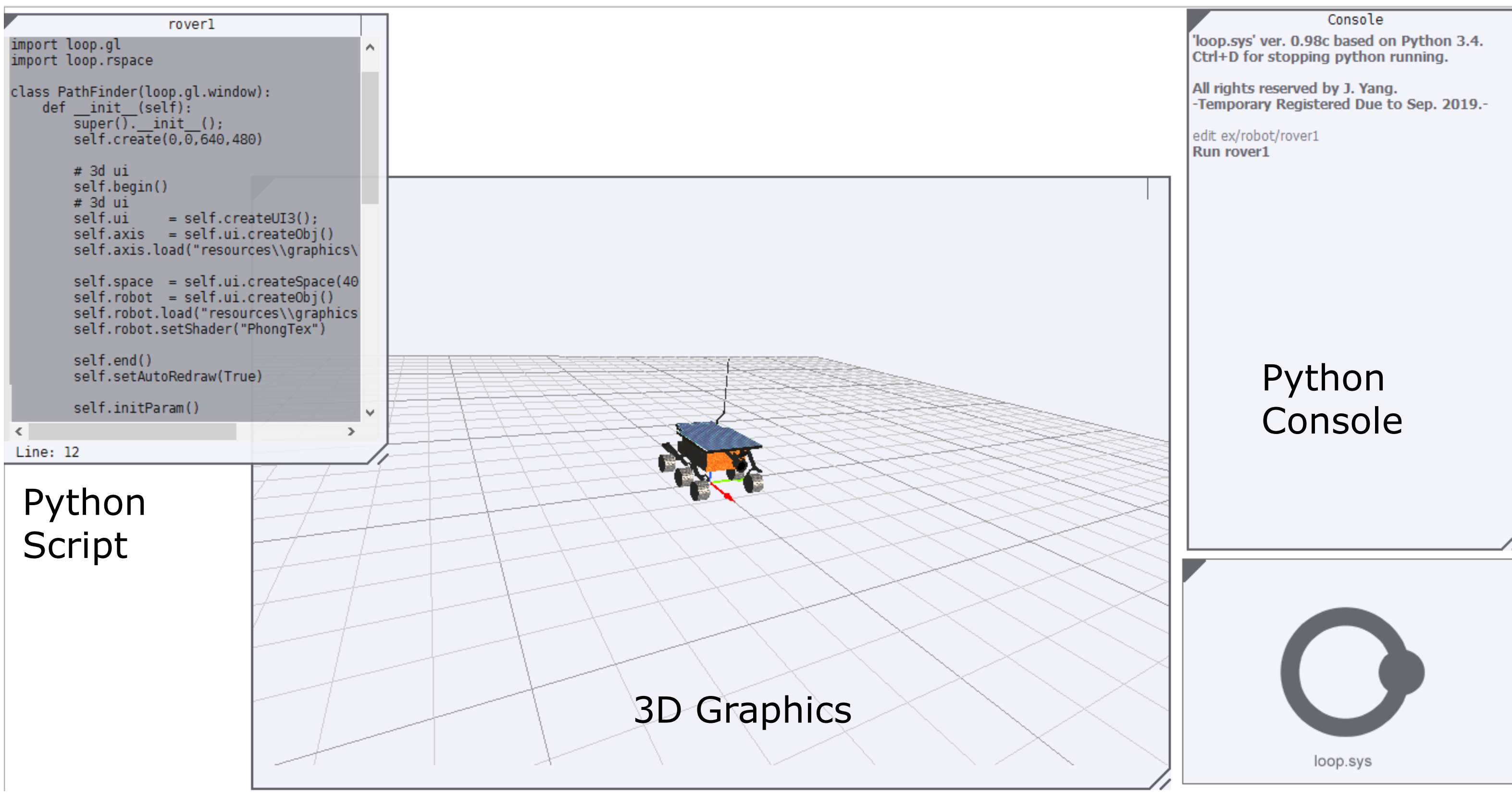
$$\dot{\alpha} = -\frac{v}{\rho}\sin\alpha + w$$

$$\dot{\beta} = -w - \dot{\alpha} = \frac{v}{\rho}\sin\alpha$$

$$\begin{pmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{pmatrix} = \begin{pmatrix} \cos\alpha & 0 \\ -\dfrac{\sin\alpha}{\rho} & 1 \\ \dfrac{\sin\alpha}{\rho} & 0 \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix}$$

Simulation Environment

- 3D Mobile robot in graphics
- Python Script for control objects



Ex) rover1.py
Graphical Movement

```
class PathFinder(loop.gl.window):
    def __init__(self):
        super().__init__();
        self.create(0,0,640,480)
```

```
# graphics
def move(self,x,y,q):
    self.x  = x;
    self.y  = y;
    self.q  = q;
    h  = loop.rspace.H()
    h  = h.Trans(x,y,0)*h.RotZ(q);
    self.robot.T(h);
```

```
# Instantiates virtual Mars Rover, Sojourner.
mr= PathFinder()

def init():
    mr.move(0,0,0)

def move(x,y,q):
    mr.move(x,y,q)
```

- “edit ex/robot/rover1”
  - Press F5 for Run code
- rover1. move(x,y,q)
- rover1.move(0,0,90)
  90 degree rotation
- rover1.move(5,0,-90)
  move to (5,0) and -90 degree rotation

$$H(x, y, \theta) = H_T H_R$$

New coordinate ← direction

Ex) rover2.py (with Jacobian Matrix) Forward Kinematics(fk) and J matrix

```
# Lecture3 pp.15
def J(self,wl,wr):
    q   = RAD(self.q);
    c   = math.cos(q)
    s   = math.sin(q)
    dx  = self.r/2*( c*wl + c*wr)
    dy  = self.r/2*( s*wl + s*wr)
    dq  = self.r/2*(-wl/self.a + wr/self.a)
    dq  = DEG(dq)
    return [dx,dy,dq]

def fk(self,wl,wr):
    [dx,dy,dq]  = self.J(wl,wr)
    self.x+=dx;
    self.y+=dy;
    self.q+=dq;
    self.q=DPI(self.q)
    self.move(self.x,self.y,self.q)
```

- Jacobian J at pp.15

$$\dot{X} = \begin{pmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{pmatrix}^T$$

$$= \frac{r}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -1/a & 1/a \end{pmatrix} \begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

$$\frac{dX}{dt} = J\frac{d\Theta}{dt}$$

- [x,y,q]=FK(w_L,w_R)
- X'= X+ dx=X+J*dq

Dept. of Intelligent Robot Eng. MU

What is DPI function? (Very Important)

- Mobile Robot Kinematics is calculated by Jacobian.
- But it is a Periodic Function

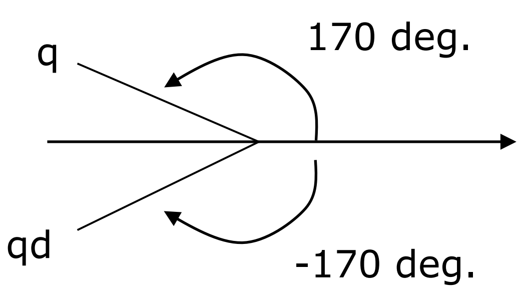
$$\frac{dX}{dt} = J\frac{d\Theta}{dt}$$

$$dX = Jd\Theta$$

$$X' = X + dX = X + Jd\Theta$$

$$\rightarrow \theta' = \theta + J_{3rd\ row} d\Theta$$

$$\rightarrow -\pi < \theta < \pi$$



$$e = q_d - q$$
$$= -170 - 170$$
$$or$$
$$= 20^\circ$$

Ex)

If q is 170 degree,
….
q'=q+30

→q'= q+30 = 200 degree
→ No problems for graphics or SLAM

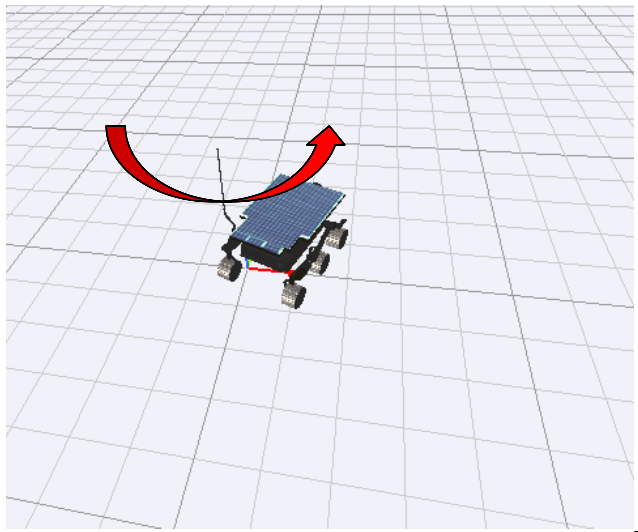
* Problem occurs for Control

Error of q = qd-q = (-170)-200
= -370

Think T = Kp * (Error of q)

Ex) Forward Kinematics with rover2.py WL=1 and WR=2

```
rover2.init()
rover2.mr.fk(1,2)
rover2.mr.fk(1,2)
rover2.mr.fk(1,2)
rover2.mr.fk(1,2)
rover2.mr.fk(1,2)
rover2.mr.fk(1,2)
```



$$\dot{X} = \begin{pmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{pmatrix}^T$$

$$= \frac{r}{2}\begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -1/a & 1/a \end{pmatrix}\begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

$$= J(\theta)\begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

$$X \leftarrow X + dX = X + Jd\Theta$$

x y q

```
rover2.init()
rover2.mr.fk(1,2)
rover2.mr.x
0.15000000000000002
rover2.mr.y
0.0
rover2.mr.q
11.459155902616464
```

$J(0)$

```
rover2.init()
rover2.mr.fk(1,2)
rover2.mr.x
0.15000000000000002
rover2.mr.y
0.0
rover2.mr.q
11.459155902616464

rover2.mr.fk(1,2)
```

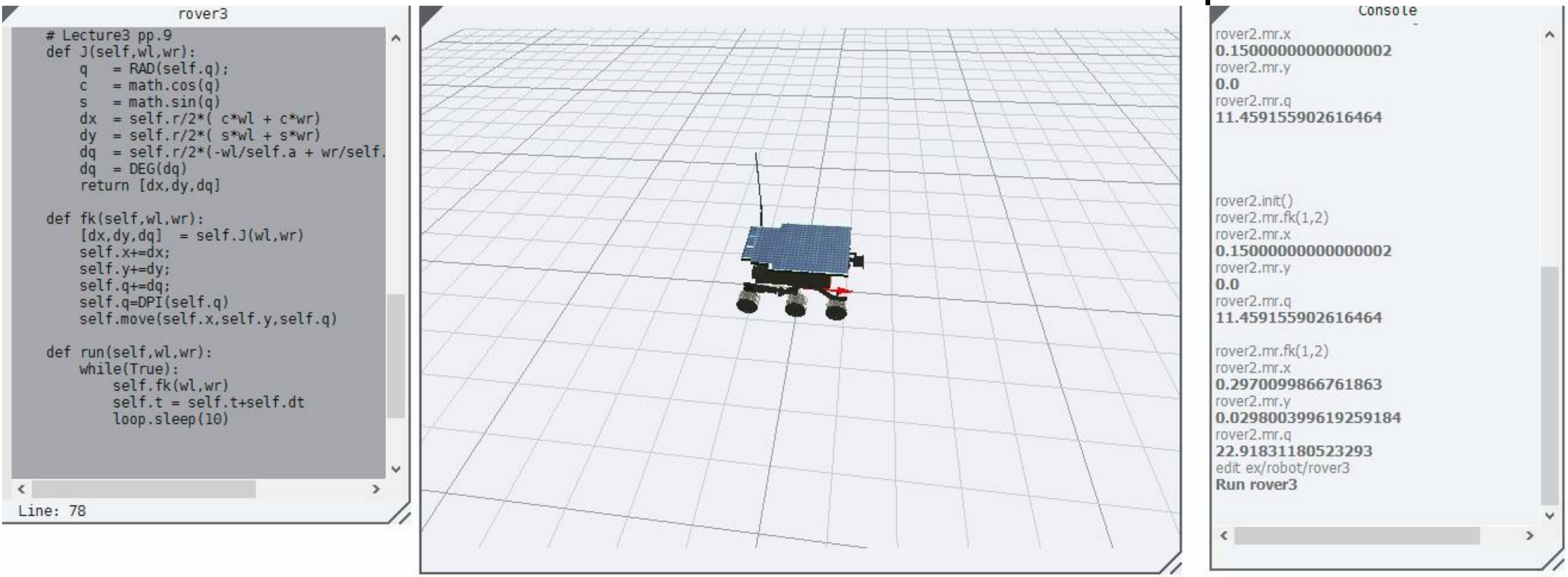
$J(11.45915)$

```
rover2.mr.x
0.2970099866761863
rover2.mr.y
0.029800399619259184
rover2.mr.q
22.9183118052329З
```

Ex) rover3.py for Control Loop

```
def run(self,wl,wr):
    while(True):
        self.fk(wl,wr)
        self.t = self.t+self.dt
        loop.sleep(10)
```

- run() starts infinite loop.
- **Type "Ctrl+D"** on console for stopping control loop



Inverse Kinematics

- Forward Kinematics of Mobile Robot

$$\dot{X} = \begin{pmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{pmatrix}^T$$

$$\dot{\Theta} = (w_L, w_R)^T$$

$$dX = Jd\Theta$$

$$(\Delta x, \Delta y, \Delta\theta) = J(\Delta\theta_L, \Delta\theta_R)$$

$$X' = X + \Delta X = FK(\Delta\theta_L, \Delta\theta_R)$$

Input: 2
Output: 3

- Is it Invertible? It is NOT a Rectangular Matrix.

- How we find a Inverse Kinematics?

$$\Theta' = \Theta + \Delta\Theta = IK(\Delta x, \Delta y, \Delta\theta)$$

  – We cannot derive IK directly by using inverse matrix.

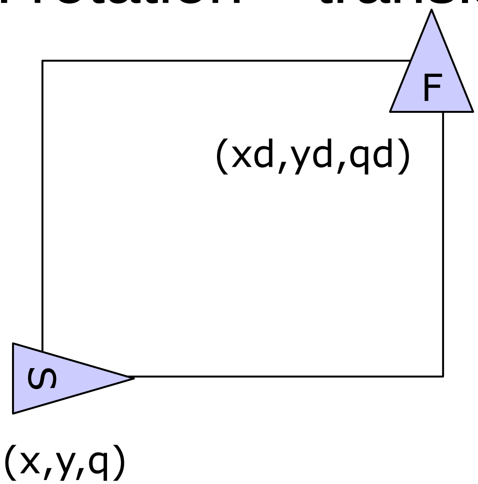
Two Inverse Kinematics Approaches

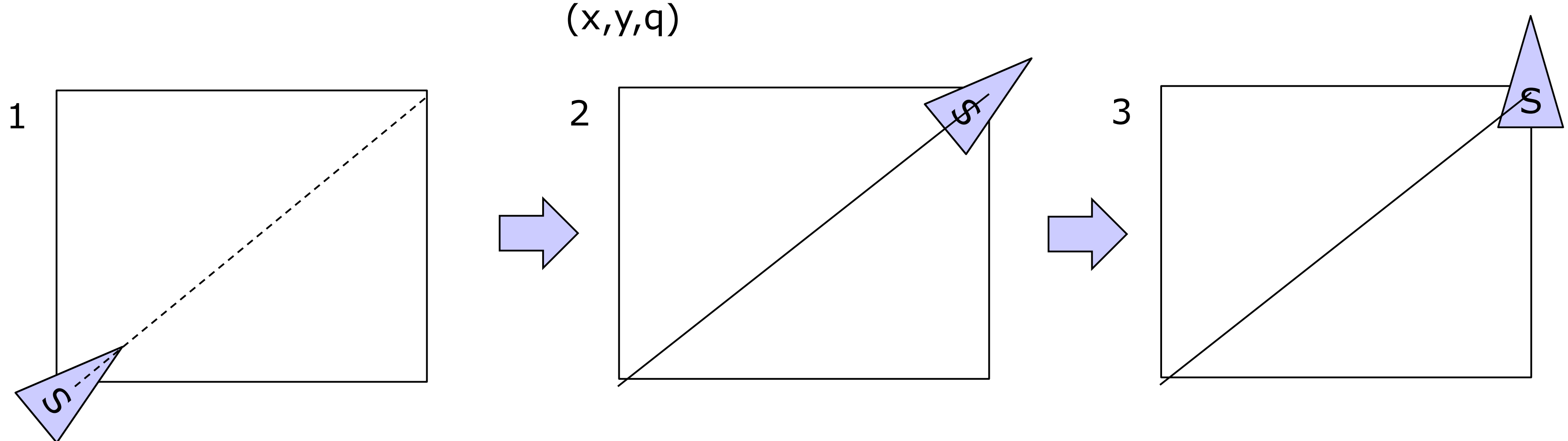
- 1. Simple Method
  - Rotation – Translation – Rotation

- 2. Inverse Kinematics with optimality
  - Lack of DOF.

Ex) rover4.py

- Simple Strategy: rotation – translation - rotation





Rotation first	Translation	Rotation for qd
$\theta_1 = \tan^{-1}\dfrac{y_d - y}{x_d - x} - \theta_s$	$X_d = (x_d, y_d)\ with\ w_0$	$\theta_d \leftarrow \theta$

I. Simple Inverse Kinematics Strategy
Case 1) Only Translation

- Assume that WL=WR=Wo

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \frac{r}{2}\begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -1/a & 1/a \end{pmatrix}\begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

⇩

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \frac{r}{2}\begin{pmatrix} 2\cos\theta \\ 2\sin\theta \\ 0 \end{pmatrix} w_o = \begin{pmatrix} \mathrm{r}\cos\theta \\ \mathrm{r}\sin\theta \\ 0 \end{pmatrix} w_o$$

⇨

$$w_L = w_0 = \dot{x}/(r\cos\theta)$$

$$w_R = w_0 = \dot{y}/(r\sin\theta)$$

$$\frac{\dot{x}}{\dot{y}} = \frac{\cos\theta}{\sin\theta}$$

I. Simple Inverse Kinematics Strategy Case 2) Only Rotation

- Assume that WL= (-)WR → WR= wo and WL=(-)wo

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \frac{r}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -1/a & 1/a \end{pmatrix} \begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

⇩

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \frac{r}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -1/a & 1/a \end{pmatrix} \begin{pmatrix} w_o \\ -w_o \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -rw_o/a \end{pmatrix}$$

⇨

$$w_L = w_0 = -\dot{\theta} a / r$$

$$w_R = -w_0 = \dot{\theta} a / r$$

I. Simple Strategy:
Rotation-Translation-Rotation

- 1. Translation by WL=WR=Wo
  - A robot cannot reach the desired position.
  - It is over constrained..
  - Wo cannot satisfy two different values, dx and dy.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \frac{r}{2}\begin{pmatrix} 2\cos\theta \\ 2\sin\theta \\ 0 \end{pmatrix} w_o = \begin{pmatrix} \mathrm{r}\cos\theta \\ \mathrm{r}\sin\theta \\ 0 \end{pmatrix} w_o$$

- 2. Rotation by WL= (-) WR=Wo
  - A robot does not Move but only rotate at an initial position.
  - It is also Over constrained so that a robot cannot move to the desired position.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \frac{r}{2}\begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -1/a & 1/a \end{pmatrix}\begin{pmatrix} w_o \\ -w_o \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -rw_o/a \end{pmatrix}$$

II. New Strategy for Inverse Kinematics

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \frac{r}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ -1/a & 1/a \end{pmatrix} \begin{pmatrix} w_L \\ w_R \end{pmatrix} \Rightarrow \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \frac{r}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \end{pmatrix} \begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

Det (c*s-c*s)=0

Not invertible.

$$\Downarrow$$

$$\begin{pmatrix} \dot{x} \\ \dot{\theta} \end{pmatrix} = \frac{r}{2} \begin{pmatrix} \cos\theta & \cos\theta \\ -1/a & 1/a \end{pmatrix} \begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

$$= B \begin{pmatrix} w_L \\ w_R \end{pmatrix}$$

$$\Rightarrow \quad B^{-1} = \begin{pmatrix} \dfrac{1}{r\cos\theta} & \dfrac{-a}{r} \\ \dfrac{1}{r\cos\theta} & \dfrac{a}{r} \end{pmatrix}$$

$$\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{pmatrix} = \frac{r}{2}\begin{pmatrix}\cos\theta & \cos\theta\\ \sin\theta & \sin\theta\\ -1/a & 1/a\end{pmatrix}\begin{pmatrix}w_L\\ w_R\end{pmatrix}$$

$$\Rightarrow \begin{pmatrix}\dot{x}\\ \dot{\theta}\end{pmatrix} = \frac{r}{2}\begin{pmatrix}\cos\theta & \cos\theta\\ -1/a & 1/a\end{pmatrix}\begin{pmatrix}w_L\\ w_R\end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix}w_L\\ w_R\end{pmatrix} = B^{-1}\begin{pmatrix}\dot{x}\\ \dot{\theta}\end{pmatrix}$$

$$B^{-1} = \begin{pmatrix}\dfrac{1}{r\cos\theta} & \dfrac{-a}{r}\\ \dfrac{1}{r\cos\theta} & \dfrac{a}{r}\end{pmatrix}$$

$$\dot{y} = \frac{r}{2}\sin\theta(w_L + w_R)$$

$$\Downarrow$$

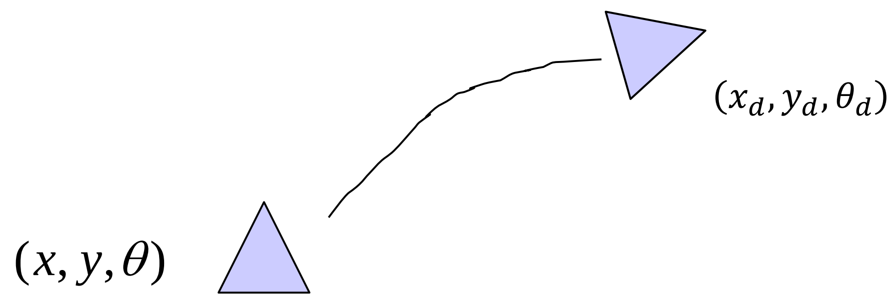
$$w_L = \frac{2\dot{y}}{r\sin\theta} - \left(\frac{\dot{x}}{r\cos\theta} + \frac{a}{r}\dot{\theta}\right)$$

$$w_R = \frac{2\dot{y}}{r\sin\theta} - \left(\frac{\dot{x}}{r\cos\theta} - \frac{a}{r}\dot{\theta}\right)$$

$$\Rightarrow \therefore \begin{pmatrix}w_L\\ w_R\end{pmatrix} = \begin{pmatrix}\dfrac{-1}{r\cos\theta} & \dfrac{2}{r\sin\theta} & \dfrac{-a}{r}\\ \dfrac{-1}{r\cos\theta} & \dfrac{2}{r\sin\theta} & \dfrac{a}{r}\end{pmatrix}\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{pmatrix}$$

Limitation of
II. New Strategy for Inverse Kinematics

$$\begin{pmatrix} w_L \\ w_R \end{pmatrix} = \begin{pmatrix} \frac{-1}{r\cos\theta} & \frac{2}{r\sin\theta} & \frac{-a}{r} \\ \frac{-1}{r\cos\theta} & \frac{2}{r\sin\theta} & \frac{a}{r} \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix}$$



- Why inverse Jacobian matrix **cannot** reach the desired goal exactly?
  - 1. The solution is the optimal one NOT the exact one.
  - 2. Remind Inverse Jacobian Method for manipulator. It is NOT Inverse Kinematics

When a Robot Stops?

$$\begin{pmatrix} w_L \\ w_R \end{pmatrix} = \begin{pmatrix} \frac{-1}{r\cos\theta} & \frac{2}{r\sin\theta} & \frac{-a}{r} \\ \frac{-1}{r\cos\theta} & \frac{2}{r\sin\theta} & \frac{a}{r} \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} \Rightarrow \text{if } \theta \to \theta_d, \text{ then } \dot{\theta} \to 0$$

$$\Rightarrow \begin{pmatrix} w_L \\ w_R \end{pmatrix} = \begin{pmatrix} \frac{-1}{r\cos\theta} & \frac{2}{r\sin\theta} \\ \frac{-1}{r\cos\theta} & \frac{2}{r\sin\theta} \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} \Rightarrow w_L = w_R = \frac{-\dot{x}}{r\cos\theta} + \frac{2\dot{y}}{r\sin\theta}$$

$$= 0$$

$$\Rightarrow \tan\theta = \frac{2\dot{y}}{\dot{x}} = \frac{2Kpe_y}{Kpe_x} = \frac{2e_y}{e_x} \Rightarrow$$

- W/O orientation error, there is position error.
- W/O position error, there is orientation error.
- Because Jacobian is NOT square matrix, you CANNOT satisfy all at once.

Inverse Jacobian is used for What?

- Think control
  - 1. Need path.
  - 2. Control input for only short distance is estimated by Inverse Jacobian method.

- Now, what is required for a mobile robot?
  - we should think about PATH for mobile robot navigation.
    Ex) for a valet parking, path is required.

  - Also, a mobile robot should KNOW WHERE IT IS?
    How to detect the current position (x,y,q)?

Ex) rover4.py (con't)

Rotation controller

```
# Only Rotation
def rcontrol(self,Kw,qd):
    self.start()
    while(self.on):
        e   = qd-self.q
        e   = DPI(e)
        if (ABS(e)<=self.esq):
            self.stop()
            print("OK")
            return True
        dq  = Kw*e
        wl  = -dq
        wr  = -wl
        self.fk(wl,wr)
        loop.sleep(10)
    return False
```

Translation Controller

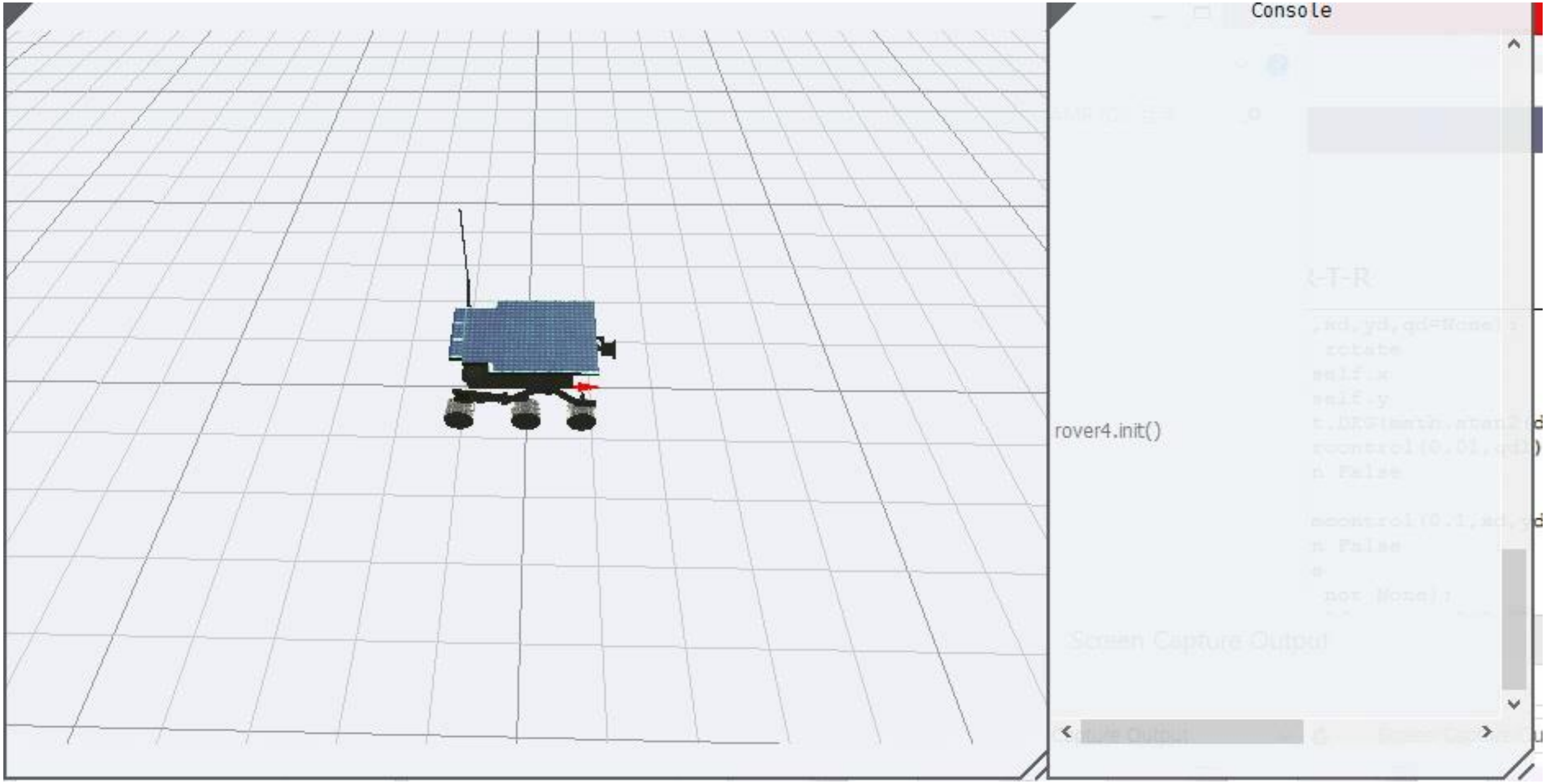
```
# only Translation
def mcontrol(self,K,xd,yd):
    self.start()
    while(True):
        ex  = xd-self.x
        ey  = yd-self.y
        e   = sqrt(ex*ex+ey*ey)
        if (e<=self.esx):
            self.stop()
            print("OK")
            return True

        # v <- v+ dv
        dv  = SAT(K*e,self.maxv)
        wl  = dv/self.r
        wr  = dv/self.r

        self.fk(wl,wr)
        loop.sleep(10)
```

Goto with R-T-R

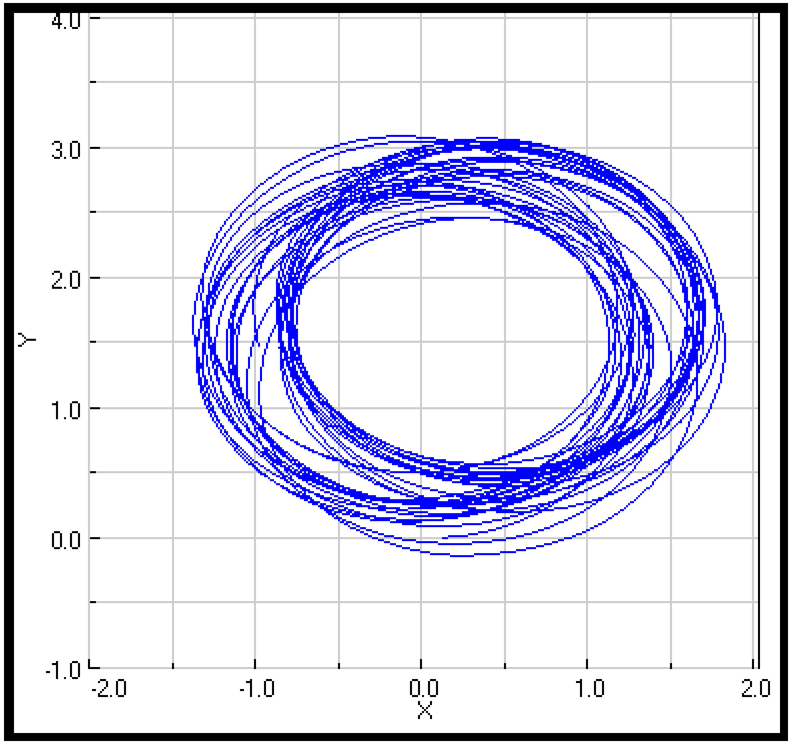
```
def goto(self,xd,yd,qd=None):
    # step 1: rotate
    dy  = xd-self.x
    dx  = yd-self.y
    qd1 = DEG(atan2(dx,dy))
    if (self.rcontrol(0.01,qd1)==False):
        return False
    # 2.move
    if (self.mcontrol(0.1,xd,yd)==False):
        return False
    # 3.Rotate
    if (qd is not None):
        if (self.rcontrol(0.01,qd)==False):
            return False
    return True
```

Import kin4
kin4.so.goto(x,y,q)

Example) Rover4.py



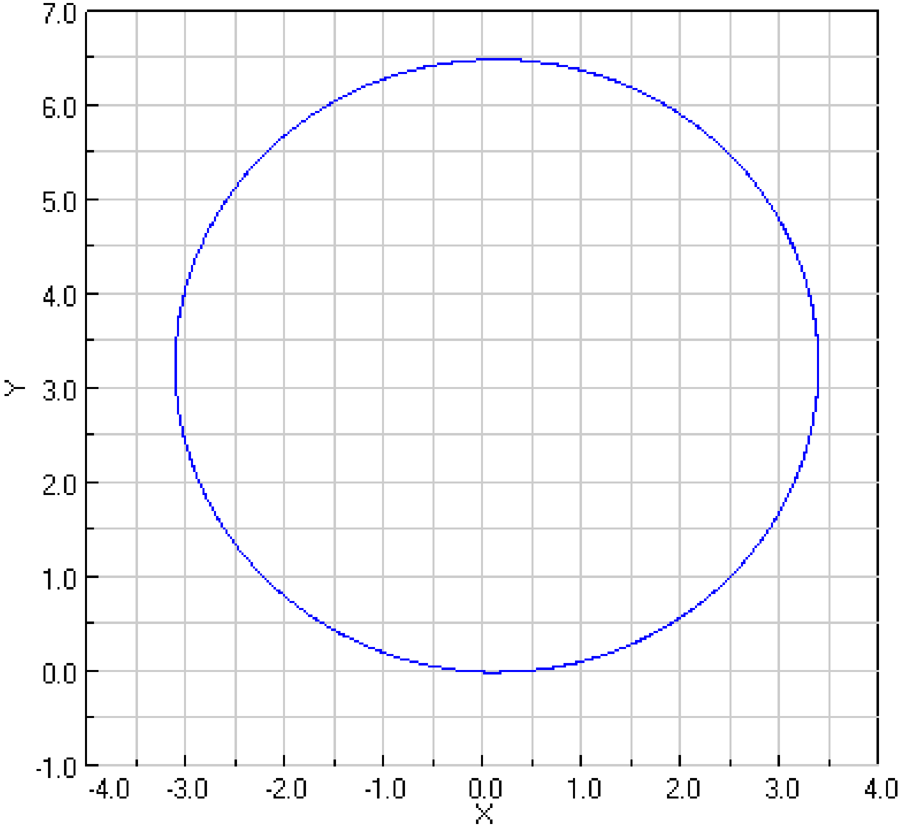
If There are **Slips** on Wheels, Forward Kinematics works well?



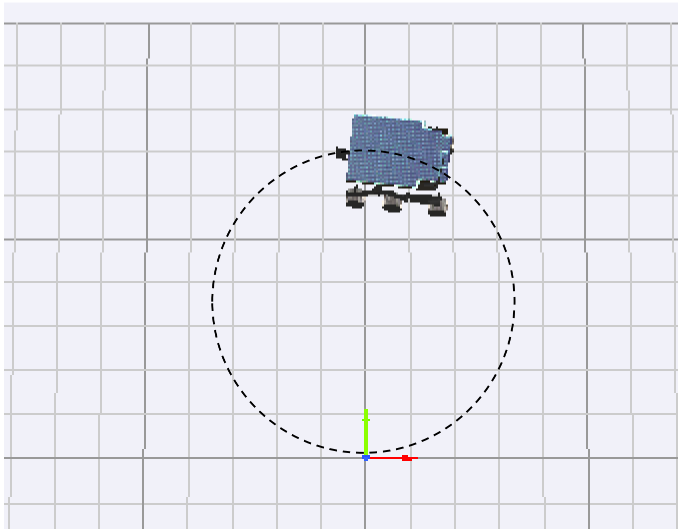
- If wheel has an error, then WL or WR has an error.

Skid marks

Slip Example: rover3slip1 with No Slip

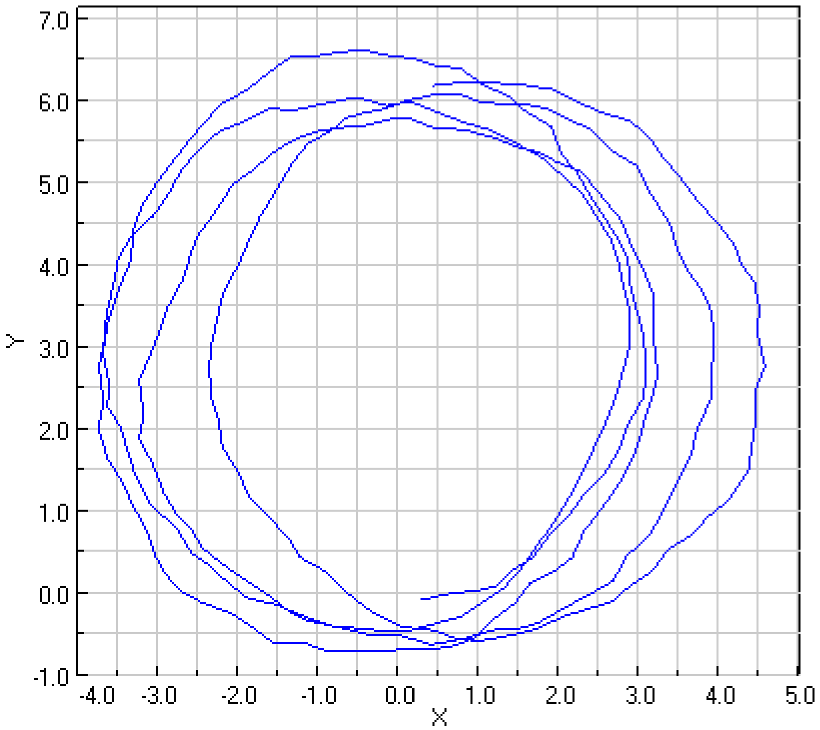


- rover3slip1.mr.run(3,3.5)



Slip Example: rover3slip2

```
def run(self,wl,wr):
    figure(1)
    clear(1)

    while(True):
        self.fk(wl,wr)

        # 0<= rand() <=1
        # -1<= nx,ny <=1

        nx = rand()*2-1
        ny = rand()*2-1
        self.x = self.x+nx*0.1
        self.y = self.y+ny*0.1

        plot(self.x,self.y)
        self.t = self.t+self.dt
        loop.sleep(10)
```

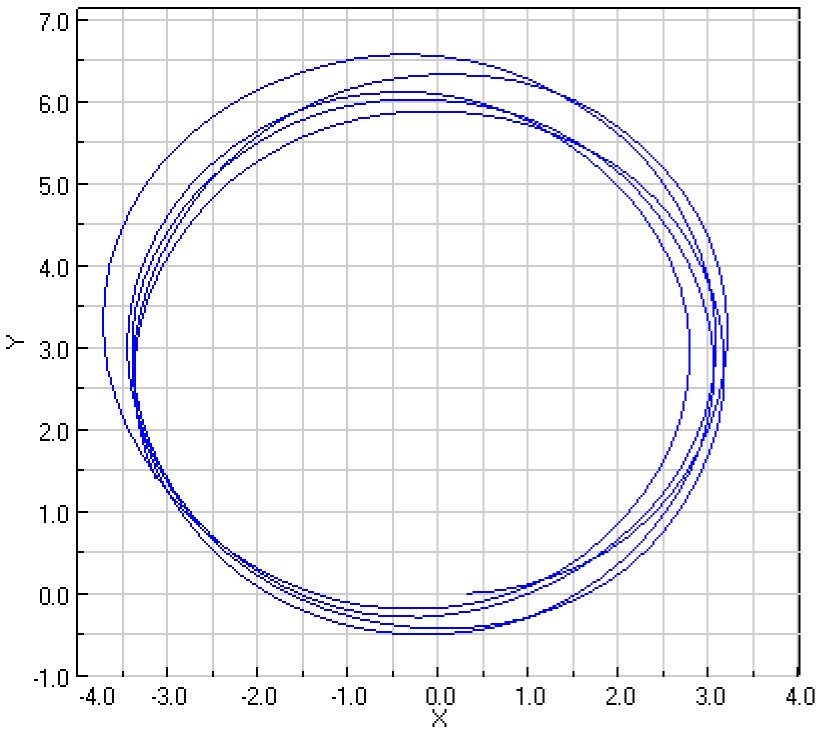


- [x,y,q] is noisy ➔ Localization Error

$$X_{noise} = X + 0.1N(0,1) \qquad N(0,1) = \text{Gaussian Noise}$$

Slip Example: rover3slip3

```
def run(self,wl,wr):
    figure(1)
    clear(1)

    while(True):
        # 0<= rand() <=1
        # -1<= nx,ny <=1

        nl = wl+0.1*(rand()*2-1)
        nr = wr+0.1*(rand()*2-1)

        self.fk(nl,nr)

        plot(self.x,self.y)
        self.t = self.t+self.dt
        loop.sleep(10)
```



- [WL, WR] is noisy ➔ Wheel slips. Wheel does not roll.

$$\Delta\theta \rightarrow w\Delta t \qquad w_{noise} = w + \alpha N(1,0)$$

HW. ex/robot/cart1 and cart2

def move(self,x,y,q): → ql, qr =? → fk(wl,wr)

```
def move(self,x,y,q):
    self.x      = x;
    self.y      = y;
    self.q      = q;
    h  = loop.rspace.H()
    h  = h.Trans(x,y,0)*h.RotZ(q);
    z     = self.r
    a     = self.a
    self.o.T( h*h.Trans(0,0,z))
    self.WL.T(h*h.Trans(0,a,z)*h.RotY(0))
    self.WR.T(h*h.Trans(0,-a,z)*h.RotY(0))
```

ql,qr?? (arrows point to the 0 in each RotY(0))

HW1. How to calculate ql and qr?

- wl and wr in fk() are the angular velocity.
- From wl and wr, we can calculate ql and qr.

HW2. Reduce Position Error

- Mobile Robots have more position error.
  - It works with **Velocity control**, it does not with Position control

- Do mr.goto(6.28,0,0) and go back by mr.goto(0,0,0)
- x,y, q, ql, qr have a lot of error.
- How to reduce those errors?

Localization

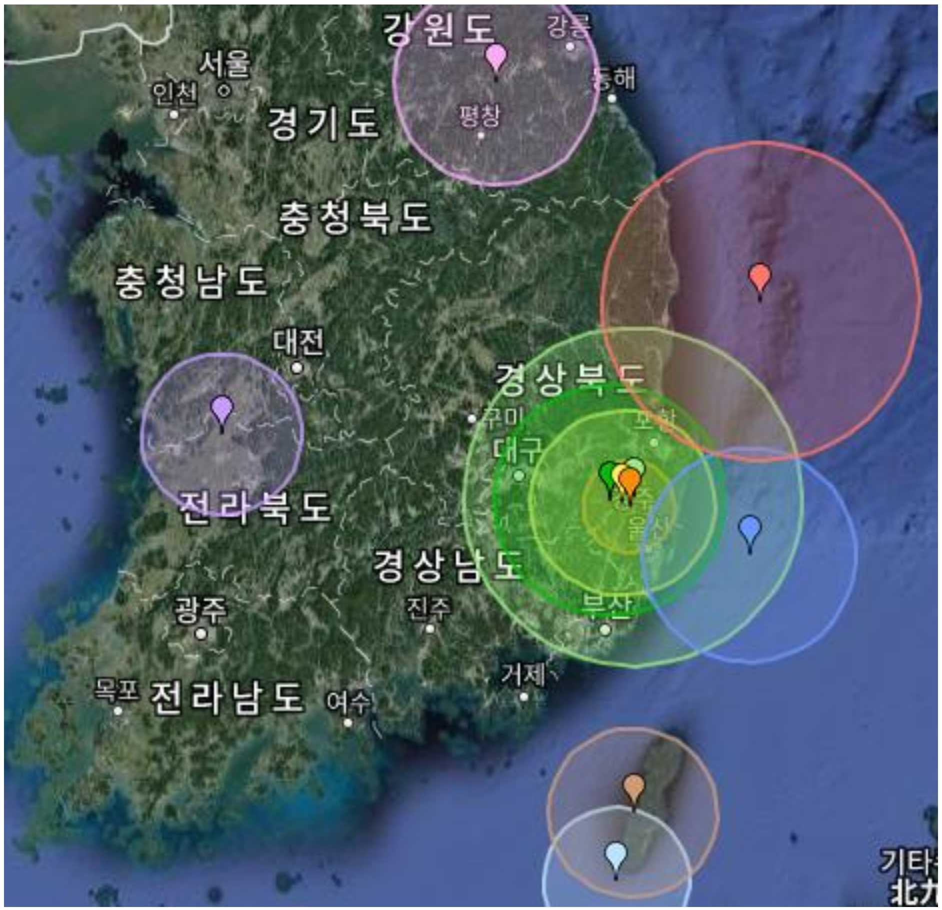
2

If an error on wheels, how to find the exact position?
Where is a mobile robot?

Localization

- A robot knows where it is.
- GPS : global positioning system is also localization.
- GPS for a car navigation system? FALSE
  - Satellite GPS is the right word.

Earthquakes

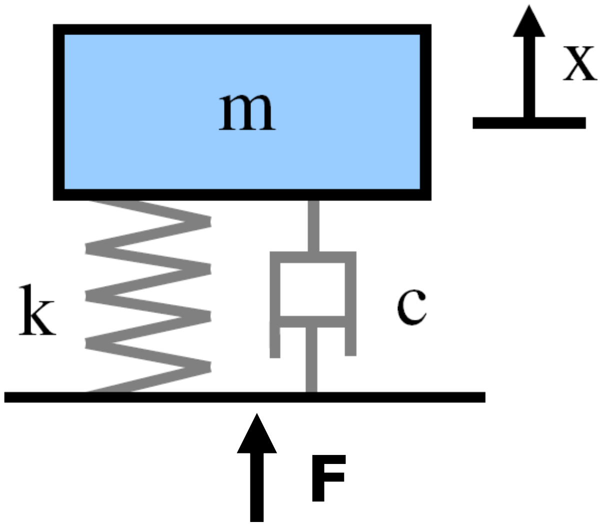


Korea has about 200 sensors.
Japan has about 4000 sensors.

How to localize the earthquake position?

Remind Control Engineering

- Think 1 DOF 2nd order system.
  - Mass-Spring-Damper



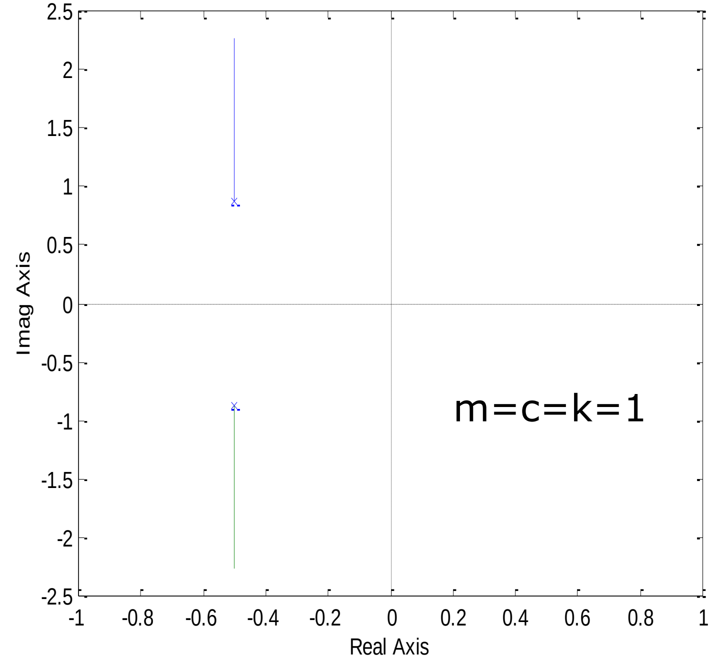
$$\sum F = ma \Rightarrow F - kx - c\dot{x} = ma = m\ddot{x} \Rightarrow m\ddot{x} + c\dot{x} + kx = F$$

Laplace Transform

$$m\ddot{x} + c\dot{x} + kx = F$$

$$(ms^2 + cs + k)X(s) = F(s)$$

$$\frac{X(s)}{F(s)} = \frac{1}{ms^2 + cs + k} = \frac{1}{(s + p_1)(s + p_2)}$$



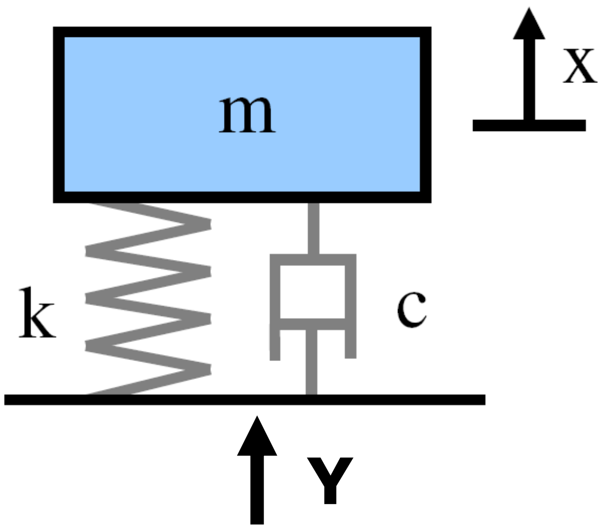
rlocus(tf(1, [1 1 1]))

- Two poles lie on left half plane.
- It is stable.
- Remind that

  Input is Force, F(s) and Output is Displacement, X(s).

For Earthquake detection, a New Input becomes displacement.

- Earthquake generate ground vibration
  - Ground displacement is the INPUT.



$$\sum F = ma$$

$$-k(x-y)-c(\dot{x}-\dot{y}) = ma = m\ddot{x}$$

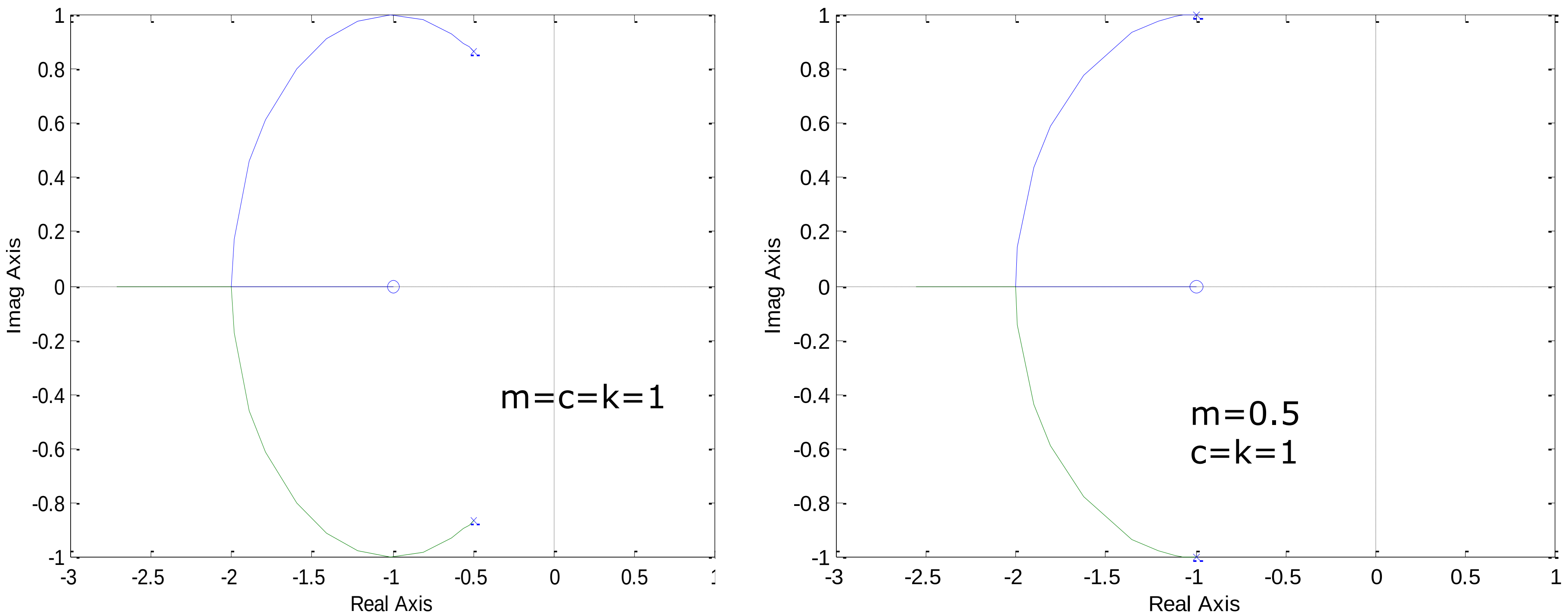
$$\therefore m\ddot{x}+c\dot{x}+kx = c\dot{y}+ky$$

$$(ms^2+cs+k)X = (cs+k)Y$$

$$\therefore \frac{X}{Y} = \frac{cs+k}{ms^2+cs+k}$$

What is the different with X/F system? → Zero.

System Behaviors of Seismic sensor



rlocus(tf([1,1], [1 1 1]))

- Zero shifts root locus to the left half plane.

Seismic sensor detects Relative motion between ground input and mass movement

Relative motion: $z = x - y$

$$-k(x-y) - c(\dot{x} - \dot{y}) = ma = m\ddot{x}$$

$$-kz - c\dot{z} = m\ddot{z} + m\ddot{y}$$

$$-m\ddot{y} = m\ddot{z} + c\dot{z} + kz$$

For intutive understanding, neglect damping

$$-m\ddot{y} = m\ddot{z} + 0 + kz$$

$$\therefore \frac{Z}{Y} = \frac{-ms^2}{ms^2 + k}$$

Instead of Root locus in s plane, we verify frequency domain by s= wj. → Remind Bode plot

$$\frac{Z}{Y} = \frac{-ms^2}{ms^2 + k} = \frac{-s^2}{s^2 + k/m} = \frac{-s^2}{s^2 + w_n^2}$$

$$\frac{Z}{s^2 Y} = \frac{Z}{\ddot{Y}} = \frac{-m}{ms^2 + k} = \frac{-1}{s^2 + k/m} = \frac{-1}{s^2 + w_n^2}$$

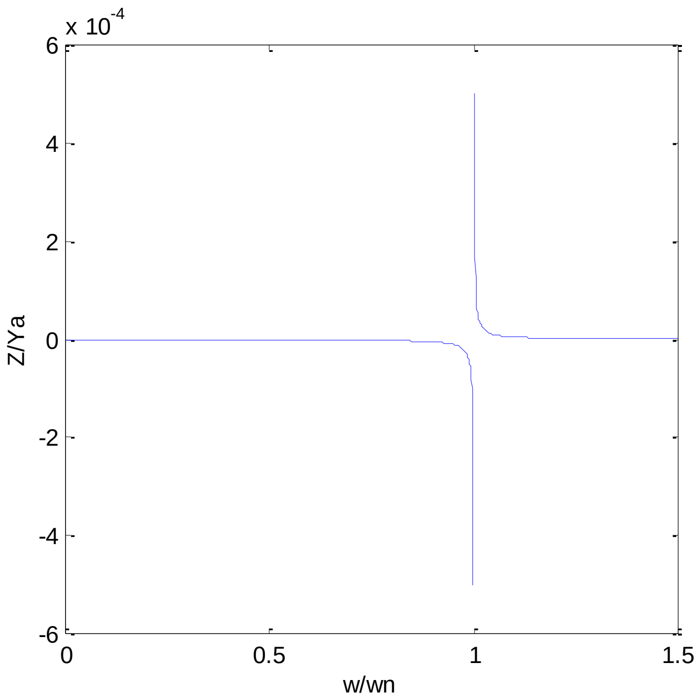
$$w_n^2 = \frac{k}{m}$$

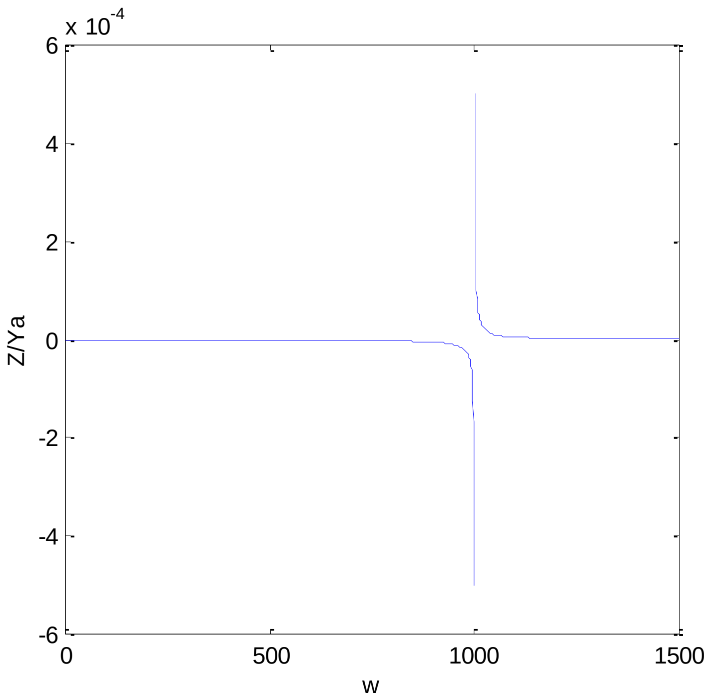
Natural frequency

$if \;\; s = wj$

$$\frac{Z}{\ddot{Y}} = \frac{-1}{-w^2 + w_n^2} = \frac{1}{w_n^2 \left(\frac{w^2}{w_n^2} - 1 \right)}$$

Relative Motion/ Ground Acceleration





$$\frac{Z}{\ddot{Y}} = \frac{1}{w_n{}^2\left(\frac{w^2}{w_n{}^2} - 1\right)}$$

Testseismic.m

w = 0~ 1500
Wn = 1000

$if \;\; \mathrm{w} << \mathrm{w}_n$

$$\frac{Z}{\ddot{Y}} = \frac{1}{w_n{}^2\left(\frac{w^2}{w_n{}^2} - 1\right)} \approx \frac{1}{w_n{}^2(0-1)} = \frac{-1}{w_n{}^2}$$

$$\therefore Z = k\ddot{Y}$$

Seismic Sensor

- **We cannot measure ground vibration directly.**

  →the displacement of a mass can be measured.

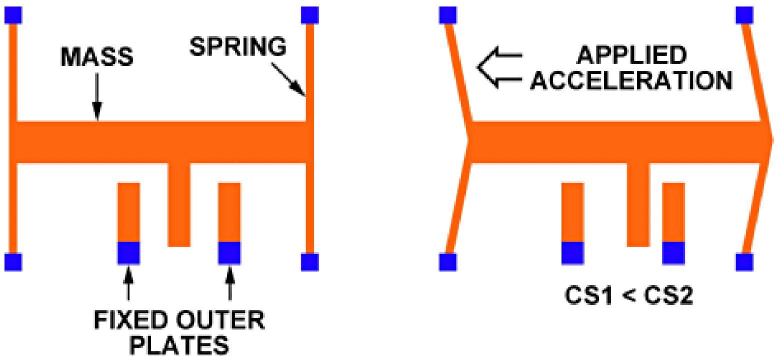
- Seismic sensor attempts to increase Wn to avoid resonance area.

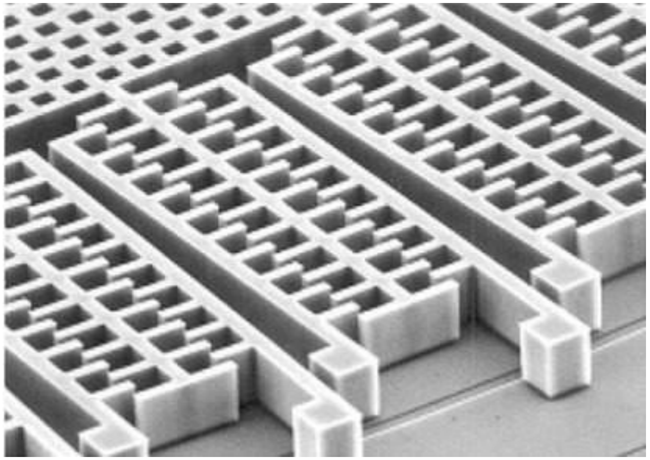
- If $w<<wn$, the relative motion z is proportional to ground acceleration.

- Question: Can you directly measure acceleration of moving object?

  → NO. IMU sensor has the same background..

Accelerometer with MEMS





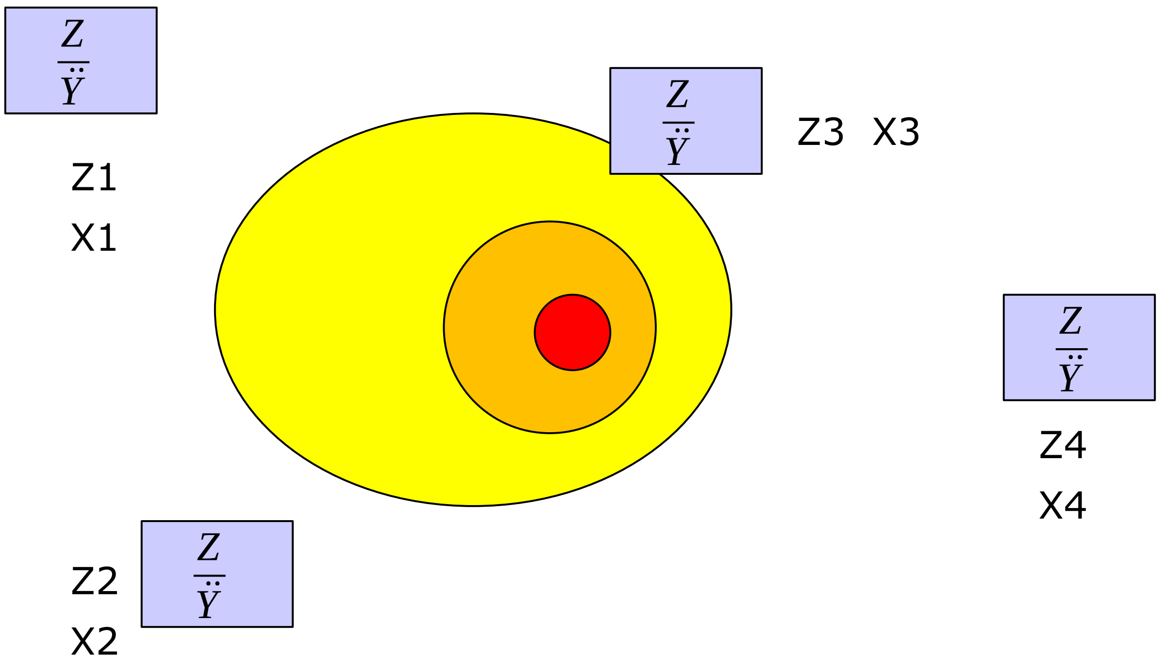
- Under the condition, w<<wn , measured capacitance detects relative motion, which is proportional to the acceleration.

$$\textit{if} \;\; w \ll w_n$$

$$Z = \frac{-1}{w_n^{\;2}} \ddot{Y}$$

The world can be analyzed in most cases.

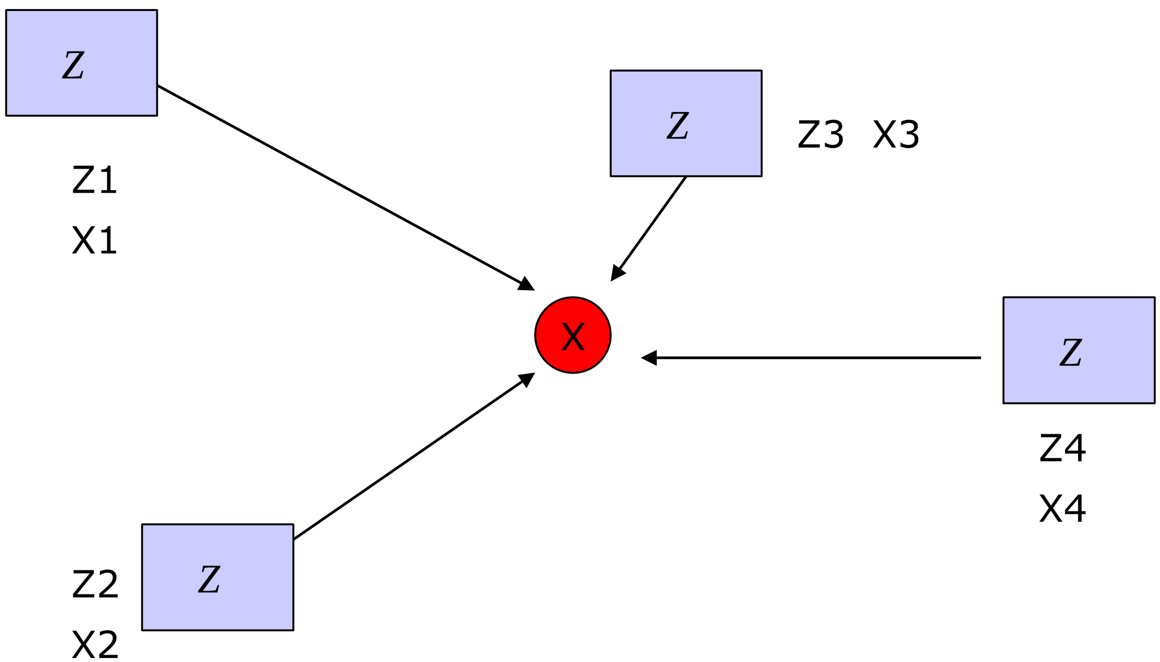
Why Earthquake for Localization



- At given time, all Zi are under consideration.
- Can we estimate the earthquake position?
- $|X_{earthquake}-X_i| = a*Z_i$

Dept. of Intelligent Robot Eng. MU

In the same manner, Beacon-based Localization.



- Solve equations,

  Zi = |Xi-X| → X

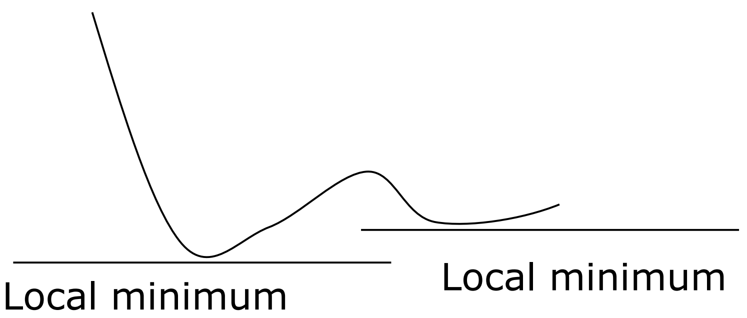
How to Solve equations?
Cost Optimization

- Current position: (x,y) unknowns.
- Beacon position
  - (0,0) , (10,0) , (0,10)
- Three distance equations.
  - |x-0|+|y-0|= d1
  - |x-10|+|y-0|=d2
  - |x-0|+|y-10|=d3
- Absolute value is inconvenient for differentiation
- So, two norm is used.

$$\| x-0 \|^2 + \| y-0 \|^2 = d_1^{\,2}$$
$$\| x-10 \|^2 + \| y-0 \|^2 = d_2^{\,2}$$
$$\| x-0 \|^2 + \| y-10 \|^2 = d_3^{\,2}$$

⇒ Find x,y

Optimization
Convex hull should be exist



- Convex hull means that it is one of the local minima.
- Think that $Y=x^2$ has the minimum value at $x=0$

- In 2D problem. $A = x^2 + y^2$ has the minimum at $x=0$ and $y=0$

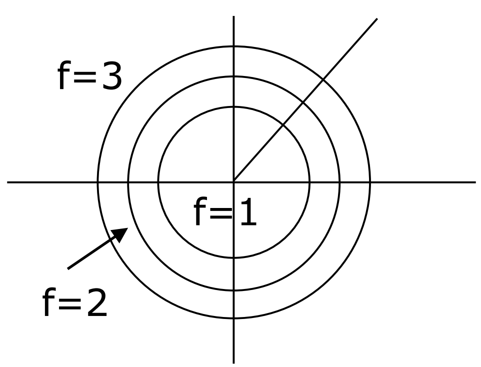
Gradient Descent Method.

- Gradient of function
  - Is defined

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

  - Keep in mind that gradient is a VECTOR.
- When f(x,y) = x^2+y^2

$$\nabla f(x,y) = (2x)\hat{i} + (2y)\hat{j}$$

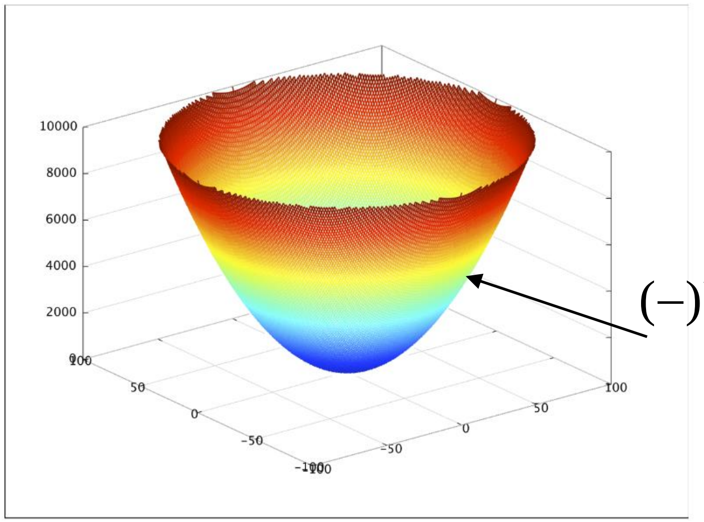


$$\nabla f(0,0) = 0\hat{i} + 0\hat{j}$$
$$\nabla f(1,0) = 2\hat{i} + 0\hat{j}$$
$$\nabla f(1,1) = 2\hat{i} + 2\hat{j}$$
$$\nabla f(-1,0) = -2\hat{i} + 0\hat{j}$$
$$\nabla f(0,-1) = 0\hat{i} - 2\hat{j}$$

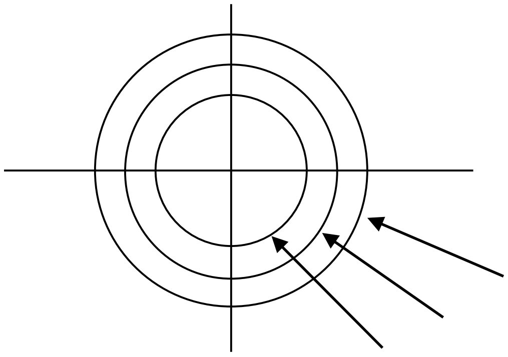
Gradient is a normal Vector!

Gradient Descent follows $(-)\nabla f(x, y)$

f(x,y) = x^2+y^2



$(-)\nabla f(x, y)$



- Gradient Descent stops at minimum.

$X \leftarrow guess$

Repeat

$X \leftarrow X - \alpha \nabla f(X)$

if $X - X_{old} < \varepsilon$ then stops.

Gradient Descent Example

$$f = x^2 + y^2$$

$$\nabla f = 2xi + 2yj$$

We know that x=0 and y=0 is the minimum.

Initial guess, x=3, y=3
Alpha = 0.01

x← x-alpha*2x
y← y-alpha*2y

X← 3 – 0.01*2*3 = 2.94
X← 2.94 -0.01*2*2.94 = 2.88
X← 2.88 -0.01*2*2.88 = 2.85
….
X← 0

- Testgd.m

```
% initial guess
x=3;
y=3;
alpha = 0.01;

for i=1:1000
    f = x^2 + y^2;
    gradx = 2*x;
    grady = 2*y;

    x = x-alpha*gradx;
    y = y-alpha*grady;

end
```

Beacon-based Localization
Iterative Optimal Problem.

- Cost function, f

X : *unknown position*

B_i : the ith beacon position

d_i : estimated distance from X to $B_i = \| X - B_i \|^2$

$$f = \sum_i^N \left(\| X - B_i \|^2 - {d_i}^2 \right)^2$$

$$\nabla f = \sum_i^N 4 \left(\| X - B_i \|^2 - {d_i}^2 \right) \left((x - B_{x,i}) \hat{i} + (y - B_{y,i}) \hat{j} \right)$$

$$X \leftarrow X - 4\alpha \sum_i^N \left(\| X - B_i \|^2 - {d_i}^2 \right) (X - B_i)^T \hat{u}$$

Example of Beacon-based Localization

$$X \leftarrow X - 4\alpha \sum_{i}^{N} \Big(\overbrace{\| X - B_i \|^2 - \underbrace{d_i^{\,2}}_{1}}^{3} \Big) \underbrace{(X - B_i)^T}_{2} \; \hat{u}$$

- Testb.m
- Beacon position = B
- B = [bx1, by1
- …..
- bx4, by4]
- 1.d = (xtrue –B)^2
- 2.xb = X-B
- 3.df = xb^2 –d
- Gradient = df *xb

```
% beacon position
B=[ 0  0
   11  0
   10   11
   0  10];

N = size(B,1);

% guess.
X=[5,5];
Xtrue=[4,8]; % we don't know it.

%sense.
d = repmat(Xtrue,N,1)-B;
d = sum( d.*d,2);
alpha = 0.001;

for i=0:100

   X

   % calc gradient.
   xb= repmat(X,N,1)-B;
   df= sum(xb.*xb,2)-d;

   g = df'*xb;
   X = X -alpha*g;
end
```

Gradient Descent Method, GDM

- You should learn GDM.
- Neural network is one of the example based on GDM.
- Over 80% of engineering method uses GDM.

- However, before use GDM, you should think about the convex hull problem.
- If there is no convex hull, GDM will be diverged.

However, calculation d has Noise.

- Uncertainty
  - Beacon position has errors.
  - Distance sensing has errors.

- Testb2: 4 beacons with errors.
  - Add Gaussian noise. B = B+ N(0,s) d = d+N(0,s)
  - Result : position estimation error = (0.29, -0.115) |e|=0.31
- Testb3: 5 beacons with errors.
  - position estimation error = (0.0086, -0.2003) |e|=0.20
- The more beacons, the better accuracy.
  - Remind that it is optimal problem → cost function f cannot be zero.