اصول علم ربات – اسلاید دوازدهم

Fundamentals of Robotics – Slide 12

Control 2

دکتر مهدی جوانمردی

زمستان ۱۴۰۰ – بهار ۱۴۰۱

[slides adapted from Gianni Di Caro, @CMU with permission]

PID Control of coupled device-environment dynamical systems

Dynamical system with state $\dot{\mathbf{x}} = F(\mathbf{x},\mathbf{u})$

observations $\boldsymbol{y}$ $\mathbf{y} = G(\mathbf{x})$

control inputs $\boldsymbol{u}$ and $\mathbf{u} = H_i(\mathbf{y})$

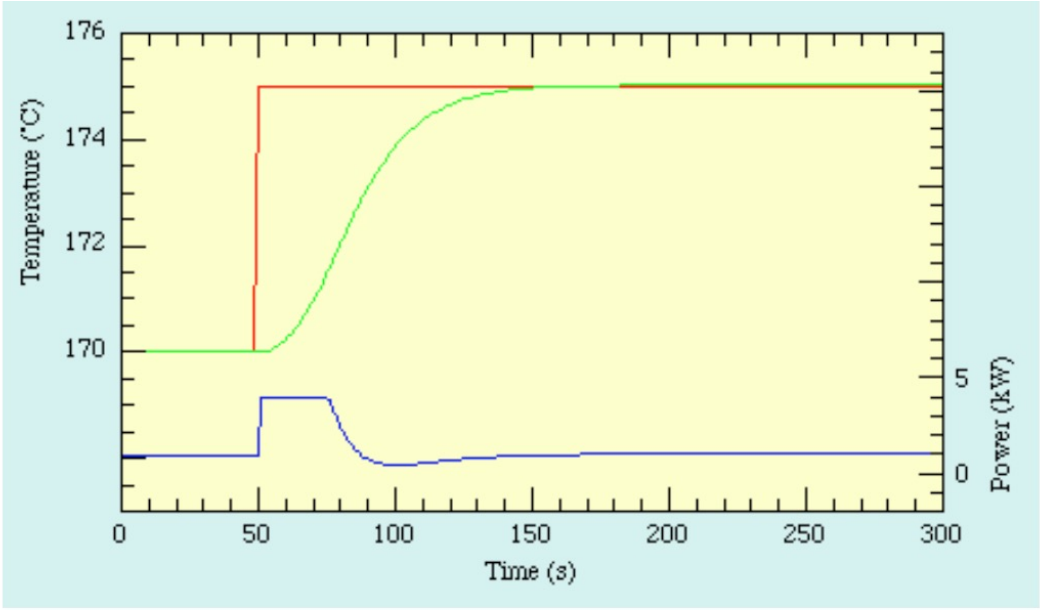
$$\dot{\mathbf{x}} = F(\mathbf{x},H_i(G(\mathbf{x})))$$

$$\dot{\mathbf{x}} = \Phi(\mathbf{x})$$

- A weighted combination of Proportional, Integral, and Derivative terms.

$$u(t) = -k_P\, e(t) - k_I \int_0^t e\, dt - k_D \dot{e}(t)$$

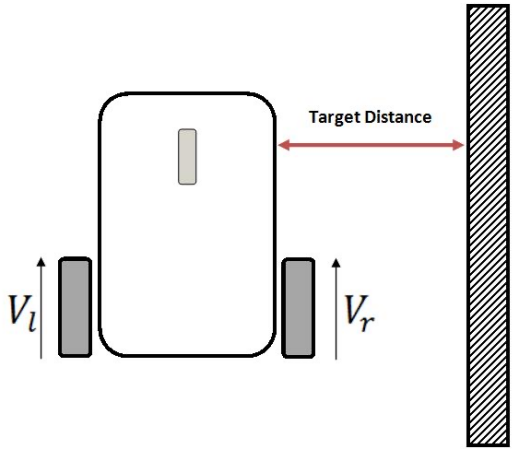
- The PID controller is the workhorse of the control industry. Tuning is non-trivial.



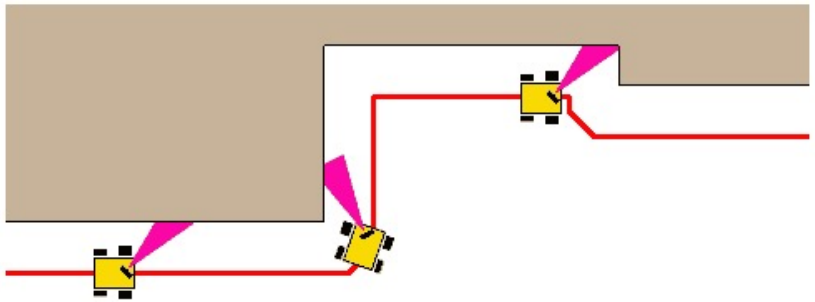
– But, good behavior depends on good tuning!

Example: wall following

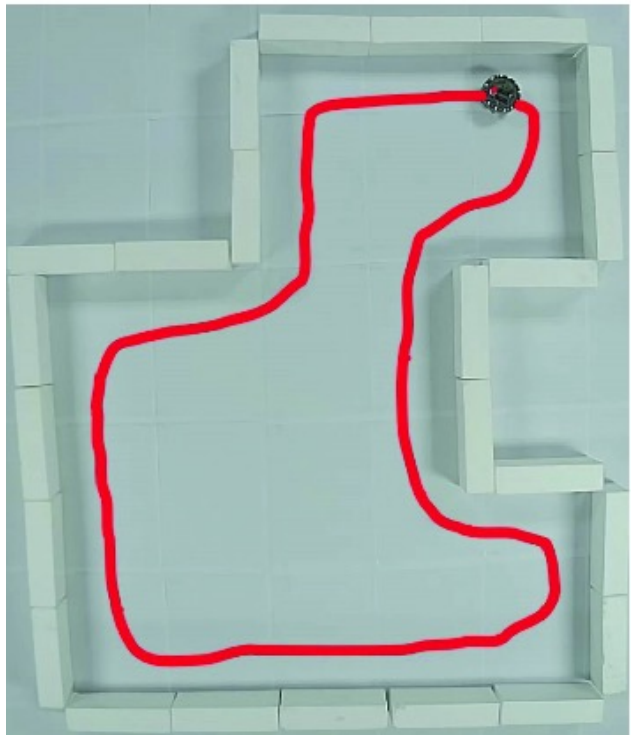
Follow a wall:
Keep at a defined distance D from the wall based on the inputs from some sensor (laser, sonar, IR, camera, …)
Maintenance goal

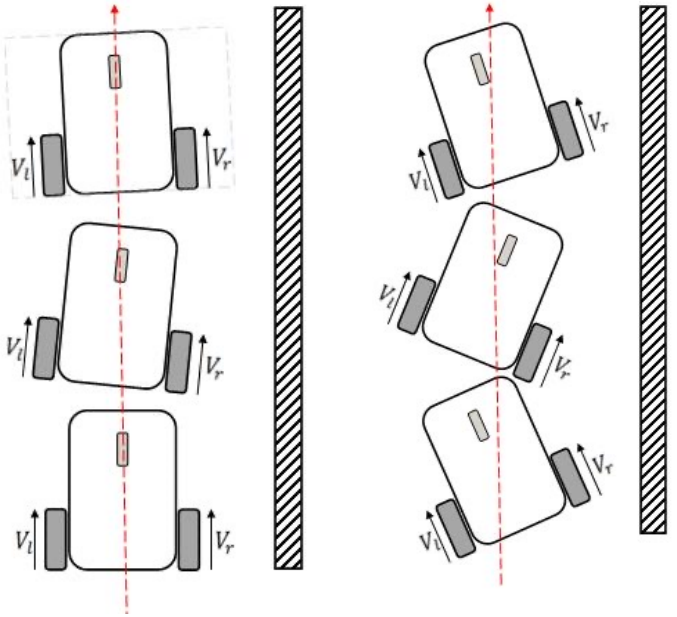


Ideal



Real





Controllers with different gains

Proportional controller (P) for wall following

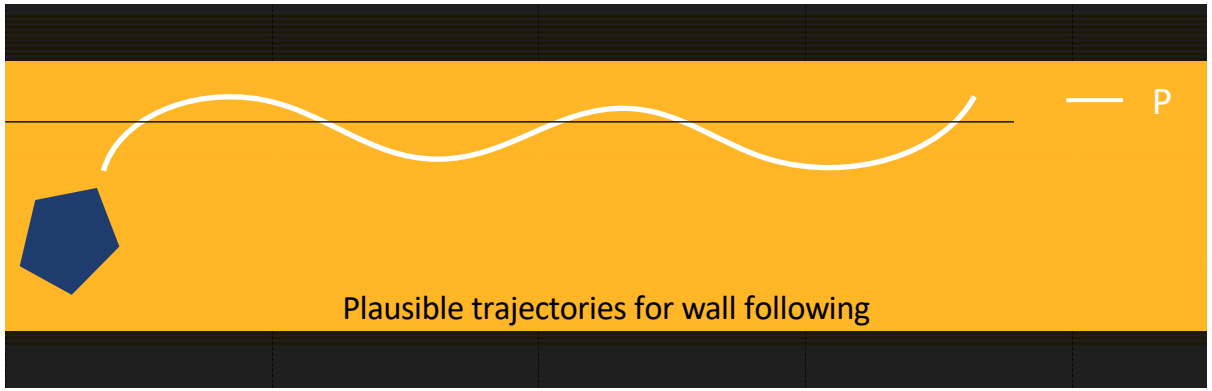
Proportional controller (P):

The controller responds in **proportion** to the **measured error** e using both magnitude and direction of the error

Output: $u = K_P e$

d = *DistanceToWall*(sensors)
$e = D - d$
if $e < 0$ # too far
    turn by $K_\theta e$ degrees toward wall
    move with a speed $v = K_v e$
else if $e > 0$ # too close
    turn by $K_\theta e$ degrees away from wall
    move with a speed $v = K_v e$
else # doing fine
    keep moving forward at current speed

- **Linear** response to error
- K_P is the **Proportional Gain**
- Gain has a major impact on oscillations and convergence
- **Damping** is the process of systematically decreasing oscillations: Gains must be adjusted to find the right balance between convergence rate and oscillations



Plausible trajectories for wall following

Derivative controller (D) for wall following

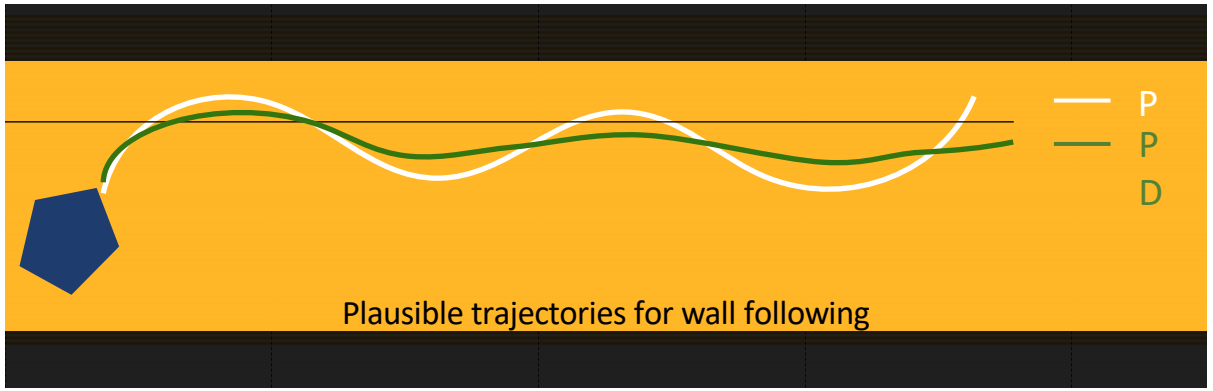
When the system is close to the desired state it needs to be controlled differently that when it is far from it!

Derivative controller (D):

The controller responds in **proportion** to the **derivative of error** e using both magnitude and direction of the error

Output: $u = K_D \frac{de}{dt}$

- **Linear** response in the *rate of change of the error*
- The behavior of a PD controller is smoother and less reactive compared to a P controller
- The controller corrects for the momentum (velocity) of the system error at it approaches the desired state
- As the robot gets closer to the wall, the controller progressively slows down the turning angle (but the same should not be applied tout-court to the linear velocity, otherwise it would get to 0!)



Plausible trajectories for wall following

Integral (I) controller

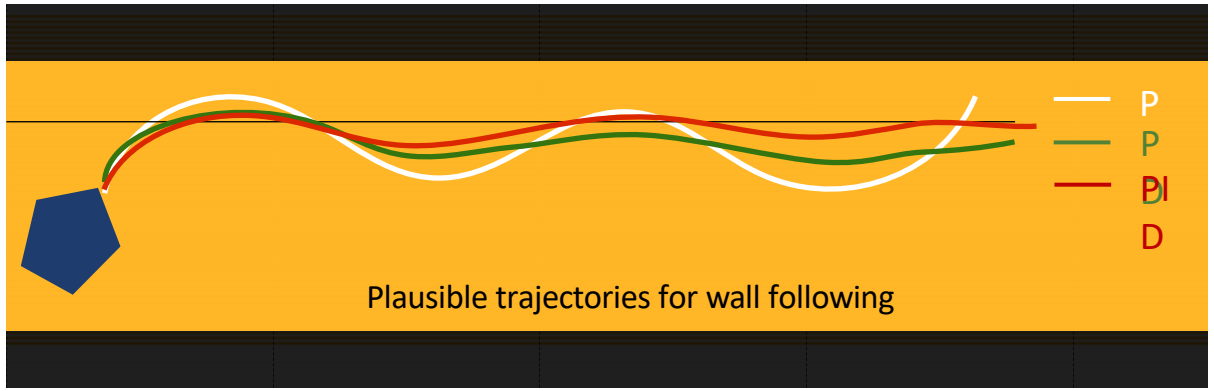
Integral controller (D):

The controller keeps track of the errors over a time window and integrates (adds) them, a **proportional** corrective control is issued

Output: $u = K_I \int_{t_0}^{t} e(t)dt$

$\int \varepsilon(t)dt$

- Linear response in the *cumulative sum of errors*
- A PI controller provides a smooth(er) response since history is considered
- It can be useful/**necessary** to keep having finite velocity when a zero error is reached, which is needed in maintenance goal states: even if the current error is zero, the sum of the errors so far or in some defined time window might still be different from zero, guaranteeing a finite velocity



PID controller

Proportional + Derivative + Integral controller (PID):

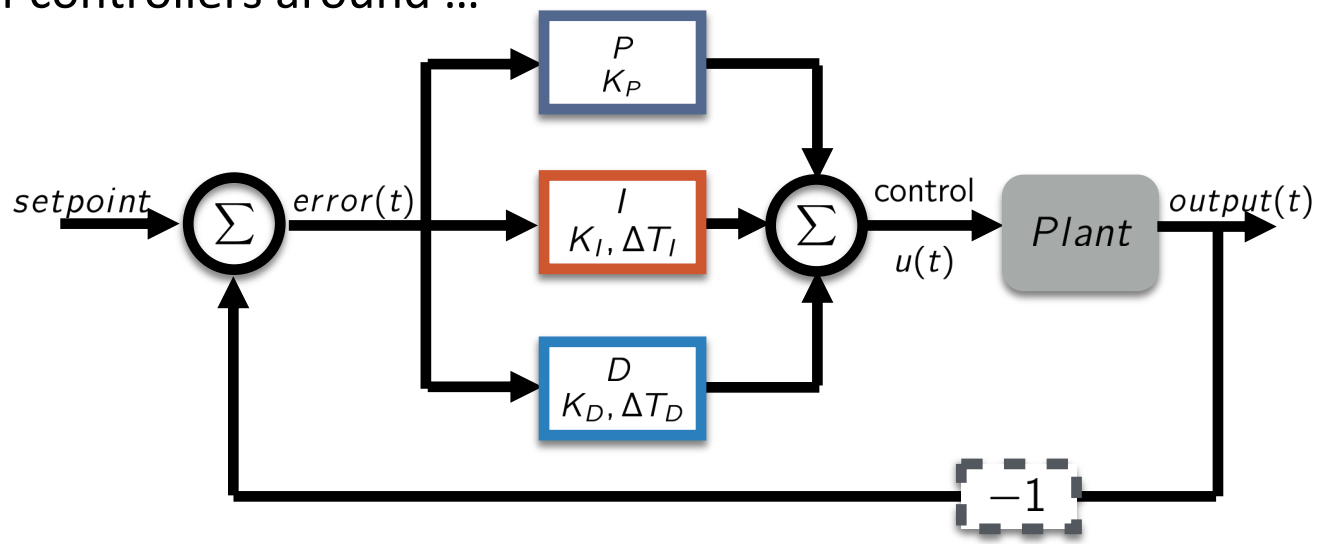
The controller linearly combined the outputs from P, I, and D control terms, each one with its own gain

Output $_{PID}$: $u(t) = K_P e(t) + K_D e(t) + K_I \int_{t_0}^{t} e(t)dt$

- Model-free: we do not model the robot-environment, only use the feedback error
- The three gains have to be determined and tuned jointly … not simple to do
- The most employed type of controllers around …

$$u(t) = K_p e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt}$$

$$u(t) = K_p \left(e(t) + \frac{1}{\Delta T_I} \int_0^t e(t)dt + \Delta T_D \frac{de(t)}{dt} \right)$$



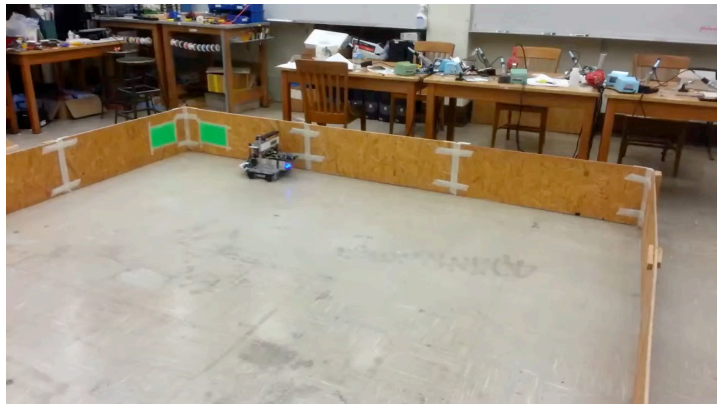
Wall following is a maintenance task: track a reference signal

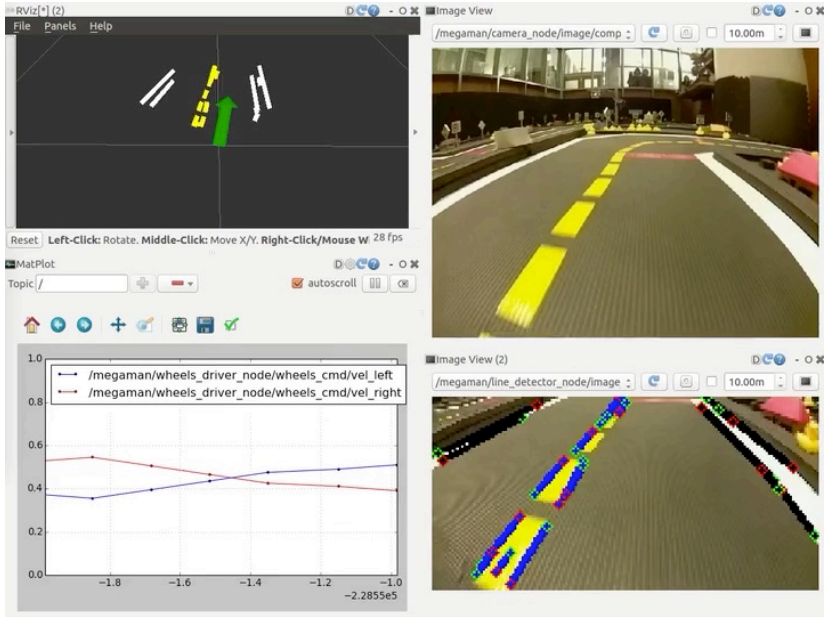
General concept: **tracking a *reference signal* keeping the robot at a specified distance / angle:**

- Wall / fence on one side, two sides, front
- Line on the road
- Person moving
- Another robot moving
- Group of people / robots moving
- Path expressed in Cartesian coordinates
- Trajectory as (Cartesian coordinates, pose angles, time)

Applications:

- Transportation
- Patrolling
- Surveillance
- Logistics
- Manufacturing processes
- Sports / Entertainment

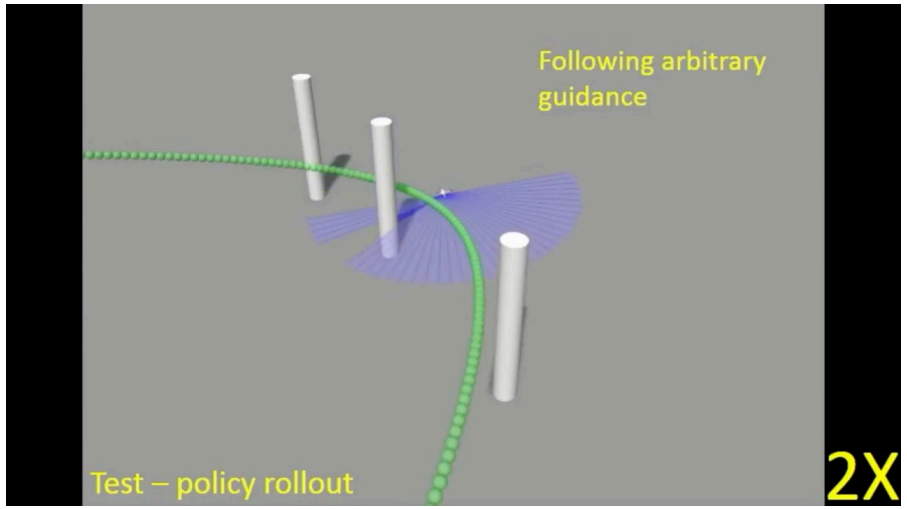


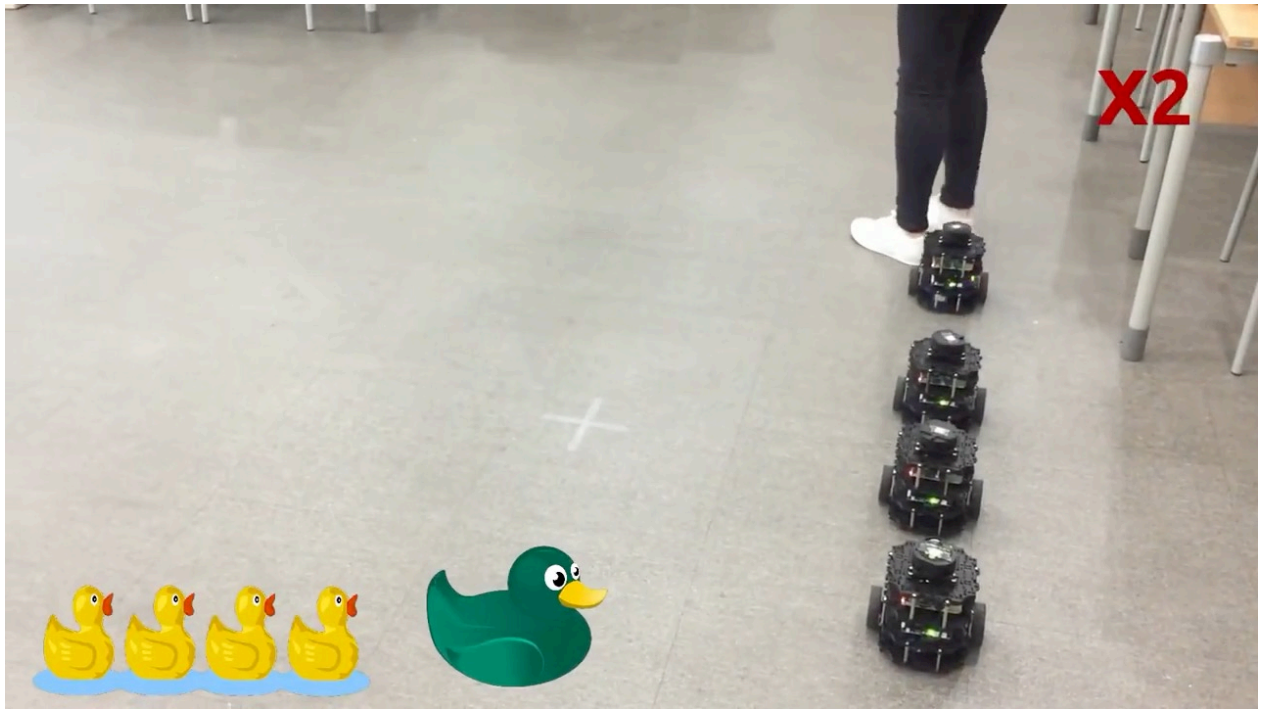


Wall following is a maintenance task: track a reference signal

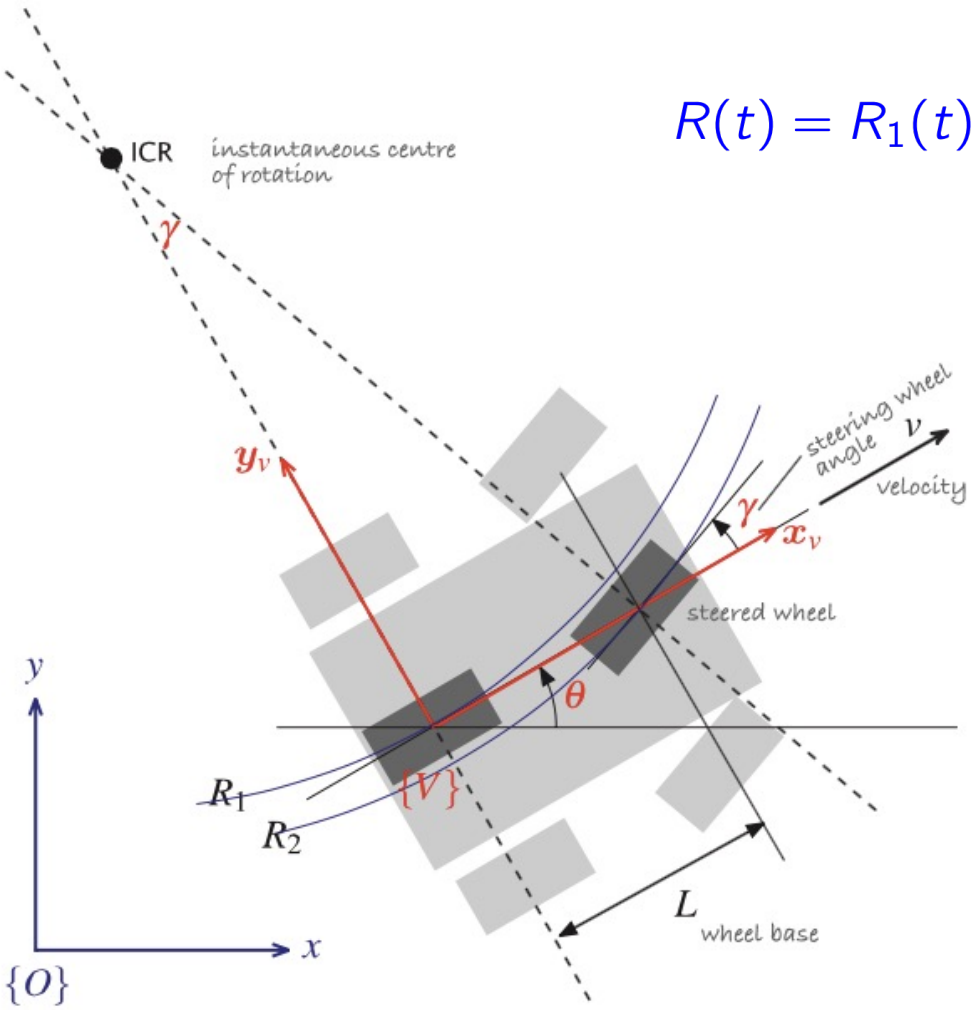
General concept: **tracking a *reference signal* keeping the robot at a specified distance / angle:**

- Wall / fence on one side, two sides, front
- Line on the road
- Person moving
- Another robot moving
- Group of people / robots moving
- Path expressed in Cartesian coordinates
- Trajectory as (Cartesian coordinates, pose angles, time)





Steered car-like robot, using the bicycle model



$$R(t) = R_1(t) = \frac{L}{\tan(\gamma(t))}, \quad \omega(t) = \frac{v(t)}{R(t)}$$

$$\dot{x} = v(t)\cos(\theta(t))$$

$$\dot{y} = v(t)\sin(\theta(t))$$

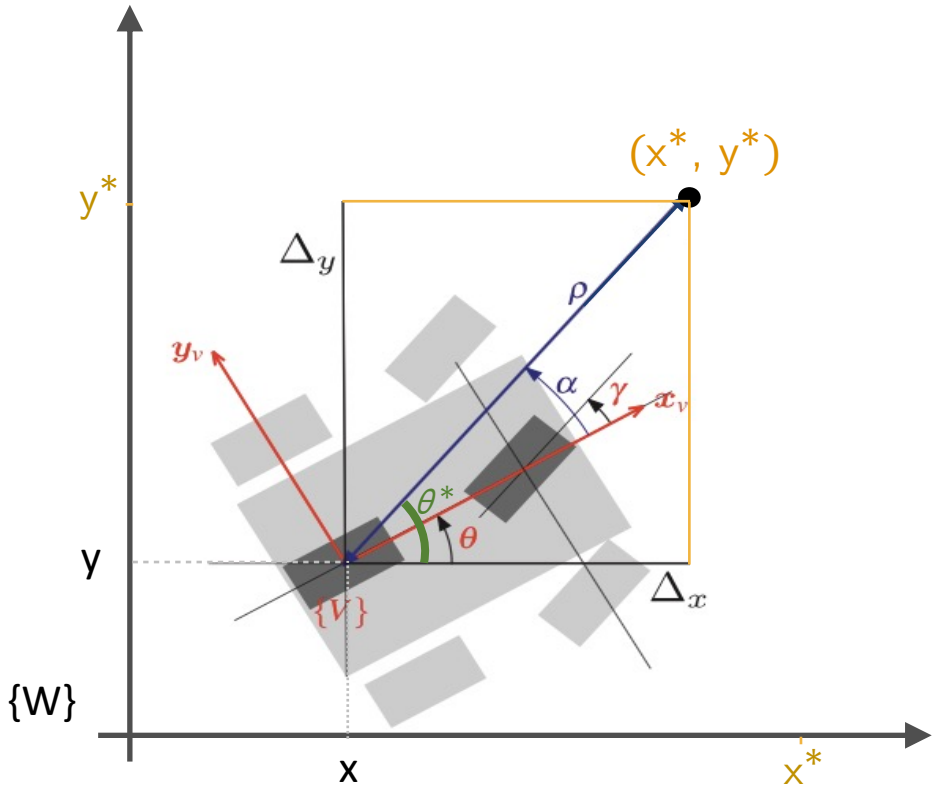
$$\dot{\theta} = \frac{v(t)}{L}\tan\gamma(t)$$

Control inputs:
v(t) and γ(t)

Moving to a goal point (P controller)

Goal: Move to a specific cartesian point $^{W}(x^*, y^*)$ in the plane

- **Control inputs**: v(t) and γ(t)
- **Current known state**: [x y θ](t) (in the W frame)
- **Error vector:** [distance from the goal, heading from the goal]



Velocity proportional to the linear distance ρ

$$v = K_v\sqrt{(x^* - x)^2 + (y^* - y)^2}, \quad K_v > 0$$

Steering proportional to the angular difference

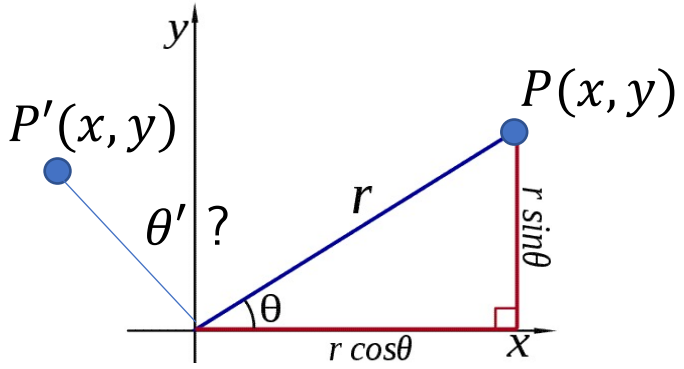
Angle of goal in W

$$\theta^* = \arctan\left(\frac{y^* - y}{x^* - x}\right) = \text{atan2}(y^* - y, x^* - x) \in [-\pi, \pi)$$

$$\gamma = K_\theta(\underbrace{\theta^* \ominus \theta}_{\alpha}), \quad K_\theta > 0$$

➢ Current γ must be accounted for in steering

Angle calculations and atan2(y, x) function

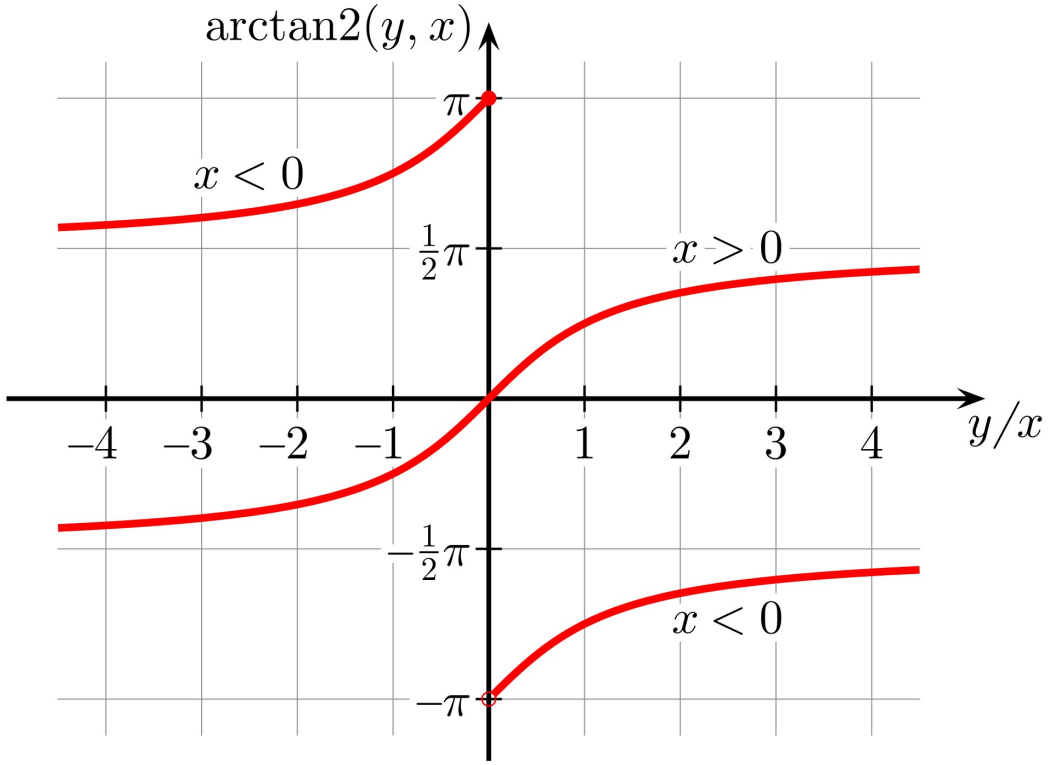


- Moved to coordinate y and x: What is my angle w.r.t. to origin?
- Move $(\Delta y, \Delta x)$ w.r.t. previous point → how much did the robot rotate?

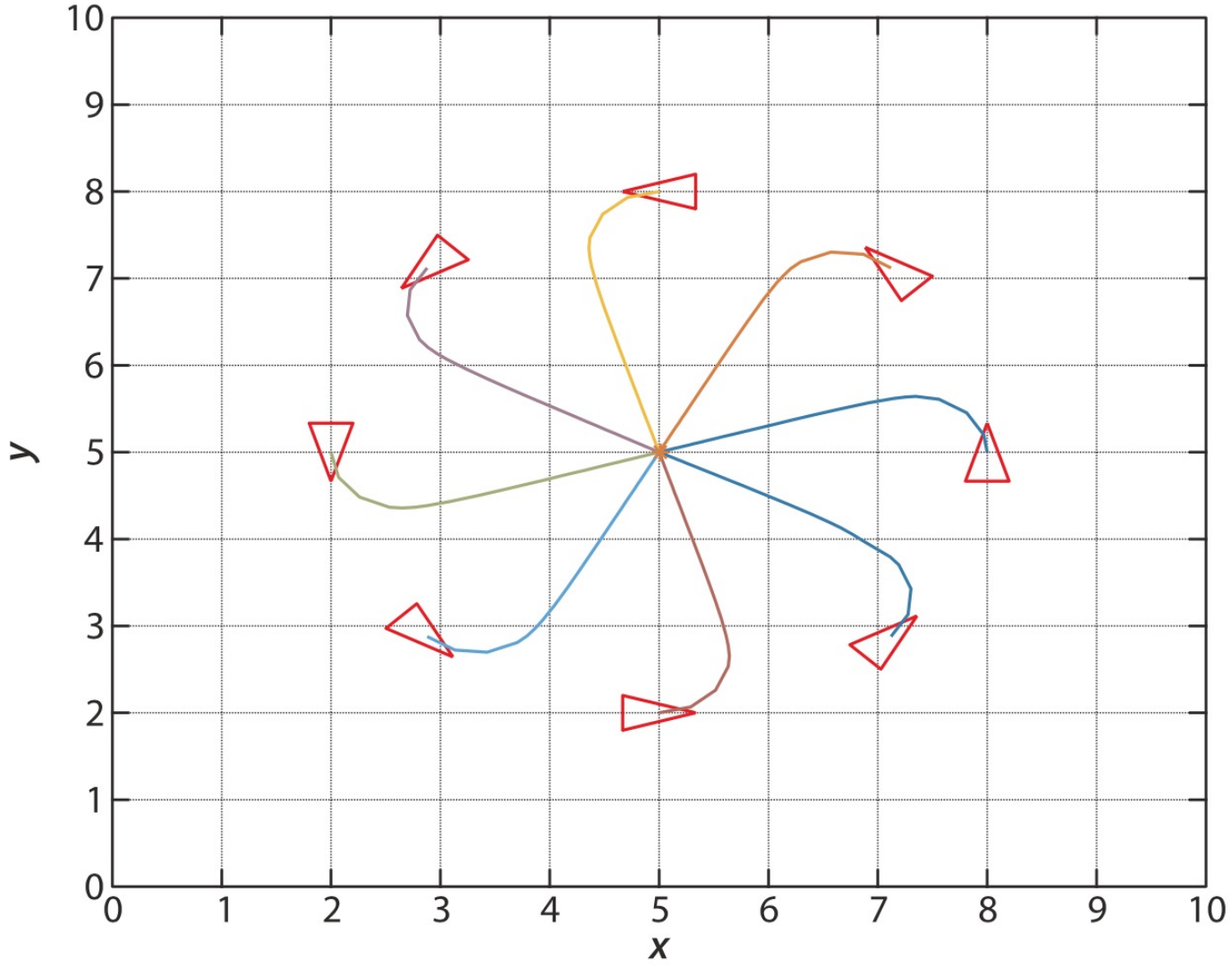
$atan2(y, x)$ returns a value of an angle θ, such that:

$$-\pi \leq \theta \leq \pi$$

$$\text{atan2}(y,x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0, \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$



Moving to a goal point (P controller)



$K_v = 0.5$

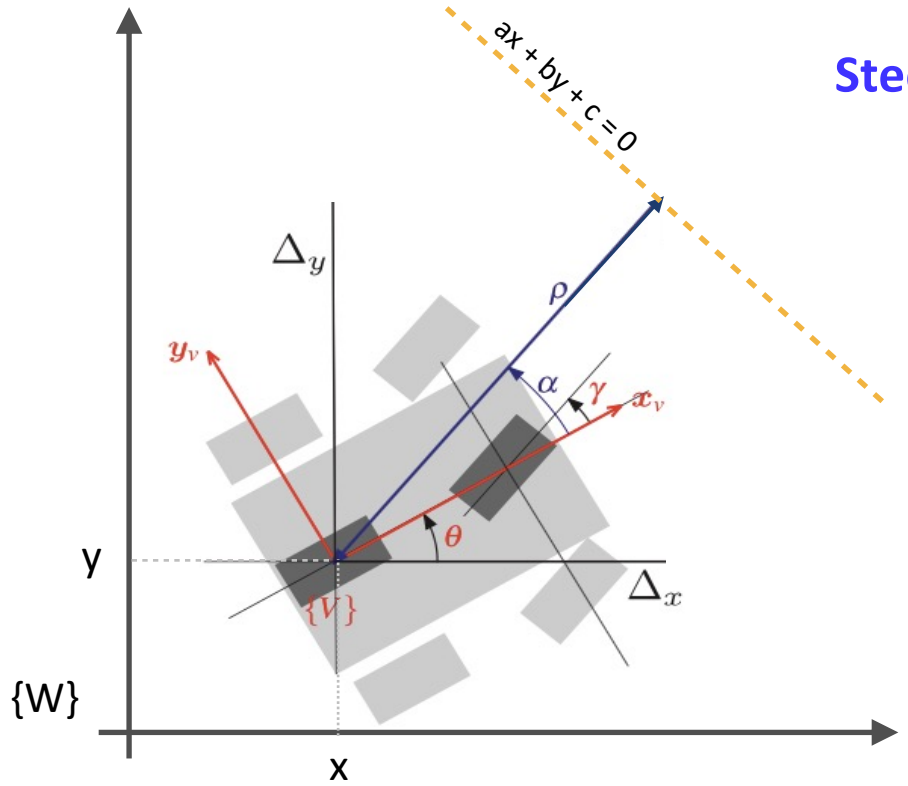
$K_\theta = 4$

The different initial positions are on a circle around the goal

Follow a line (P controller)

Goal: Follow a line in the plane, defined by the cartesian equation ax + by + c = 0

- **Control inputs**: γ(t), v(t) is constant for keep following the line
- **Current known state**: pose $[x \;\; y \;\; \theta](t)$ (in the W frame)
- **Error vector** (for the heading)**:** [distance from the line, alignment with the line]



Steer to minimize perpendicular distance ρ of V's origin from the line

$$\rho = \frac{(a, b, c) \cdot (x, y, 1)}{\sqrt{a^2 + b^2}} \qquad \alpha_\rho = -K_\rho \rho, \quad K_\rho > 0$$

Steer to make the robot parallel to the line: proportional to angular difference between robot's orientation and line's slope

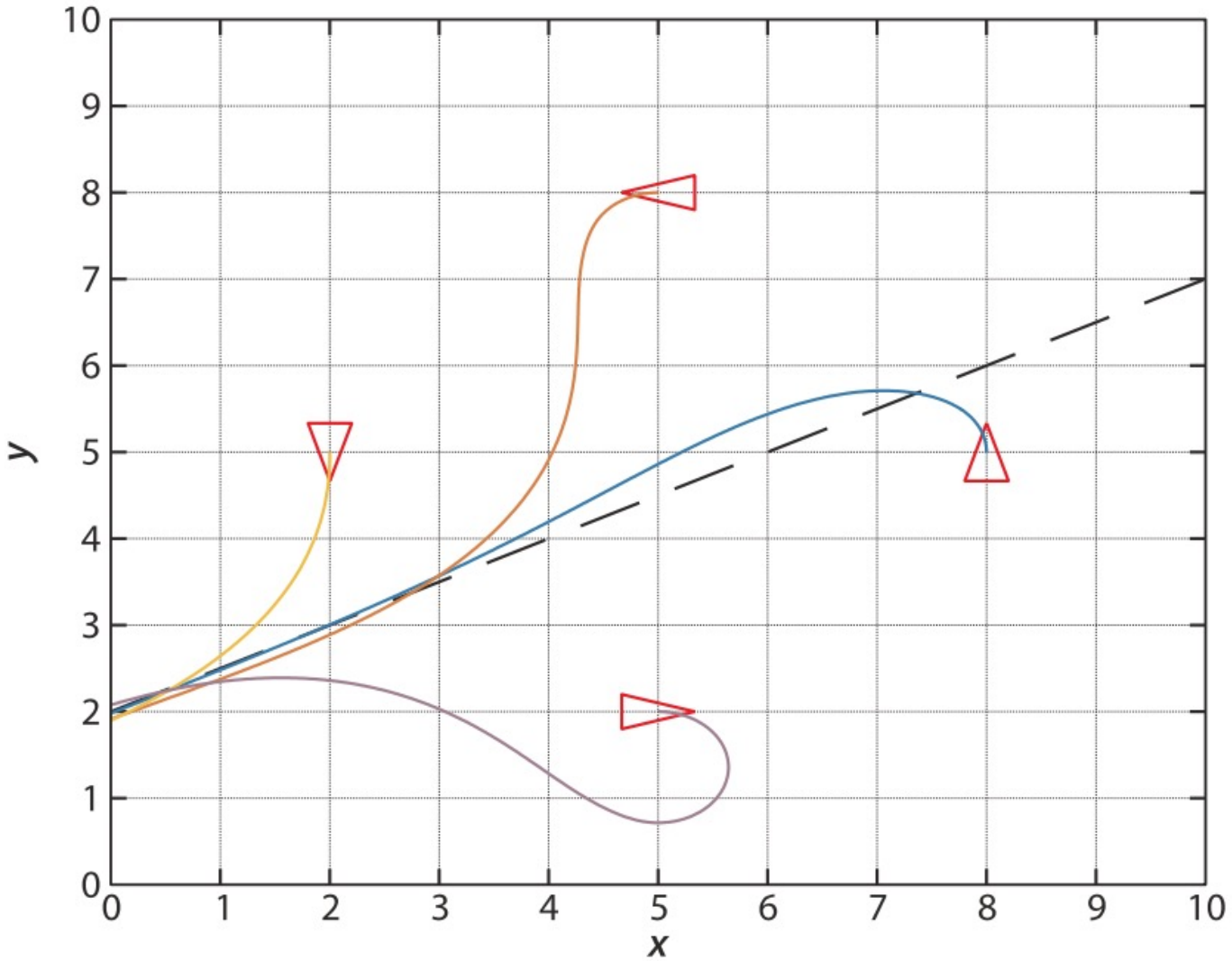
Orientation angle of the line in W

$$\theta^* = \arctan\left(\frac{-a}{b}\right) = \text{atan2}(-a, b) \in [-\pi, \pi)$$

$$\alpha_\theta = K_\theta(\theta^* \ominus \theta), \quad K_\theta > 0$$

Control law: $\gamma = -K_\rho \rho + K_\theta(\theta^* \ominus \theta)$

Following a line (P controller)



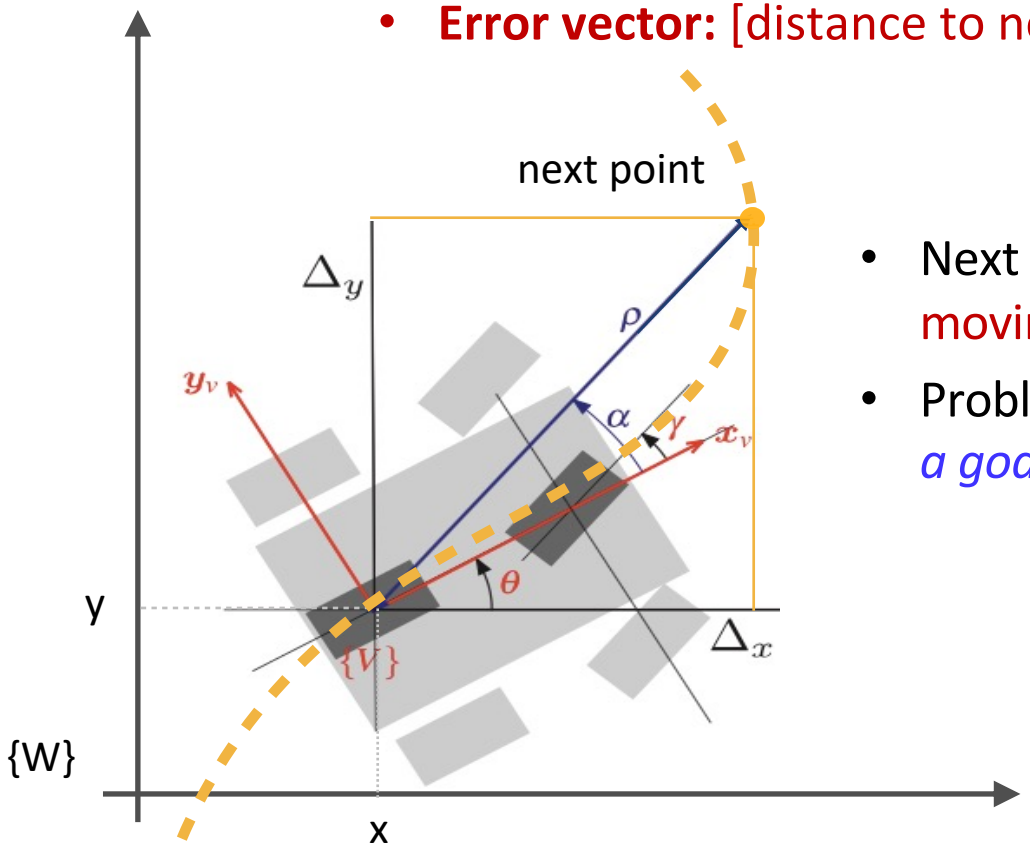
$K_\rho = 0.5,\ K_\theta = 1$

$x - 2y + 4 = 0$

Follow a path (PI controller)

Goal: Follow a general parametric curve (x*(t),y*(t)) on the plane
(e.g., generated by a path planner or resulting from sensors)

- **Control inputs**: v(t), γ(t)
- **Current known state**: pose [x y θ](t) and next point in path (in the W frame)
- **Error vector:** [distance to next point, angle to next point]



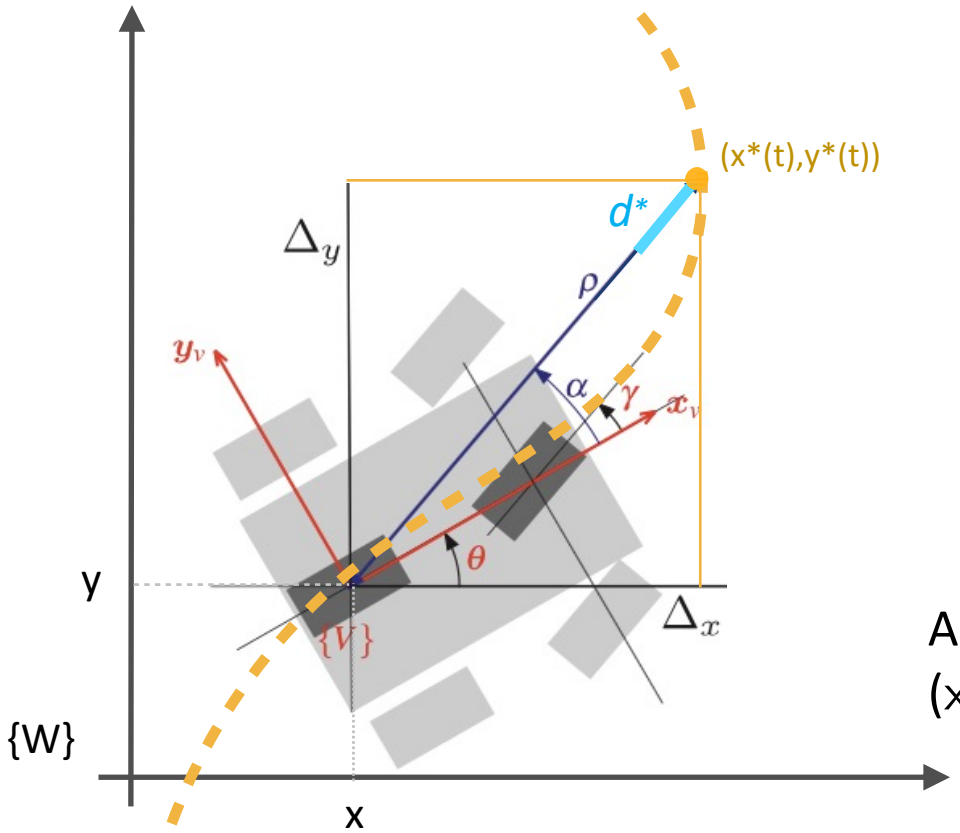
Pursuit strategy:

- Next point (x*(t),y*(t)) on the path defines the goal state and keeps moving along the goal path, with the robot keeping following it.
- Problem becomes therefore equivalent to a sequence of *moving to a goal point* problems, arriving at each point with non-zero velocity

Carrot on a stick!

Following a path (PI controller)

- **Control inputs**: v(t), γ(t)
- **Current known state:**
  pose [x y θ](t) and next point in path (in the W frame)
- **Error vector:**
  [distance to next point with offset, angle to next point]



➢ **Liner velocity** proportional (P) to the linear distance ρ from the pursuit point, that should be maintained at a defined distance d^*

$$\varepsilon_\rho(t) = \sqrt{(x^*(t) - x(t))^2 + (y^*(t) - y(t))^2} - d^*$$

➢ The **linear velocity PI controller** aims to keep the robot at the distance d^* from the next point. The integral term guarantees a **finite velocity** when the error on distance goes to 0

$$v^*(t) = K_\rho \varepsilon_\rho(t) + K_i \int \varepsilon_\rho(t)dt$$

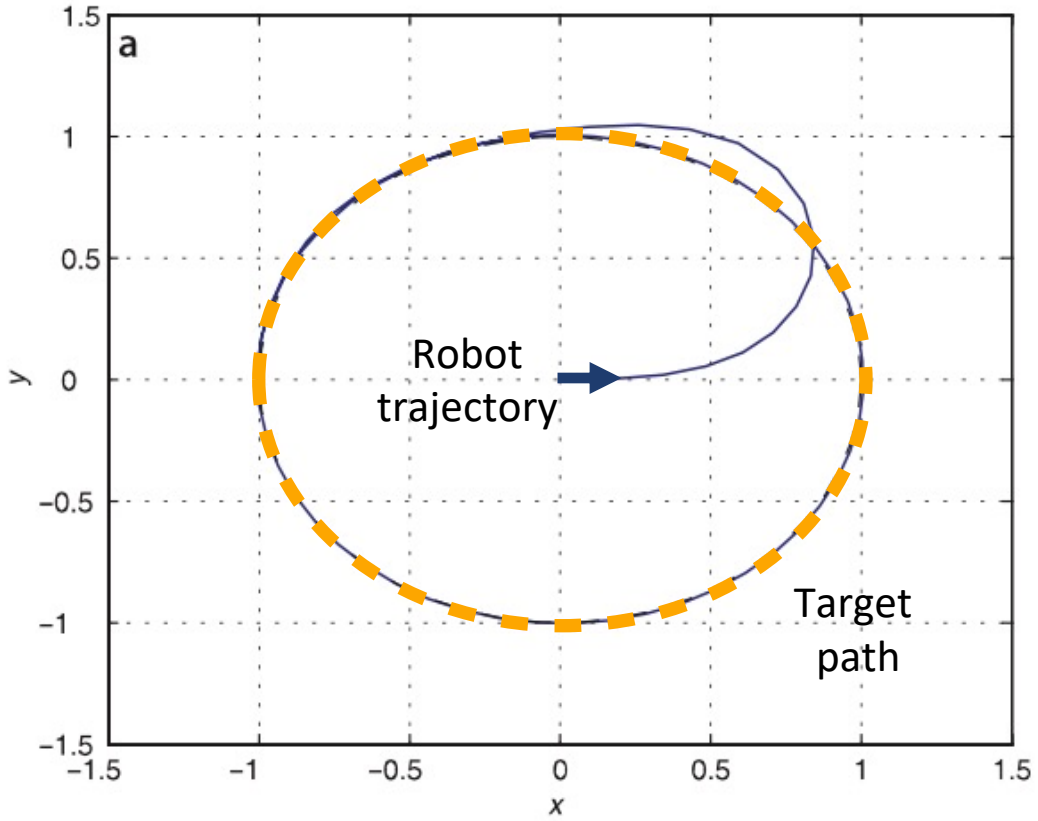
➢ **Steering** proportional to the angular difference between robot and target point

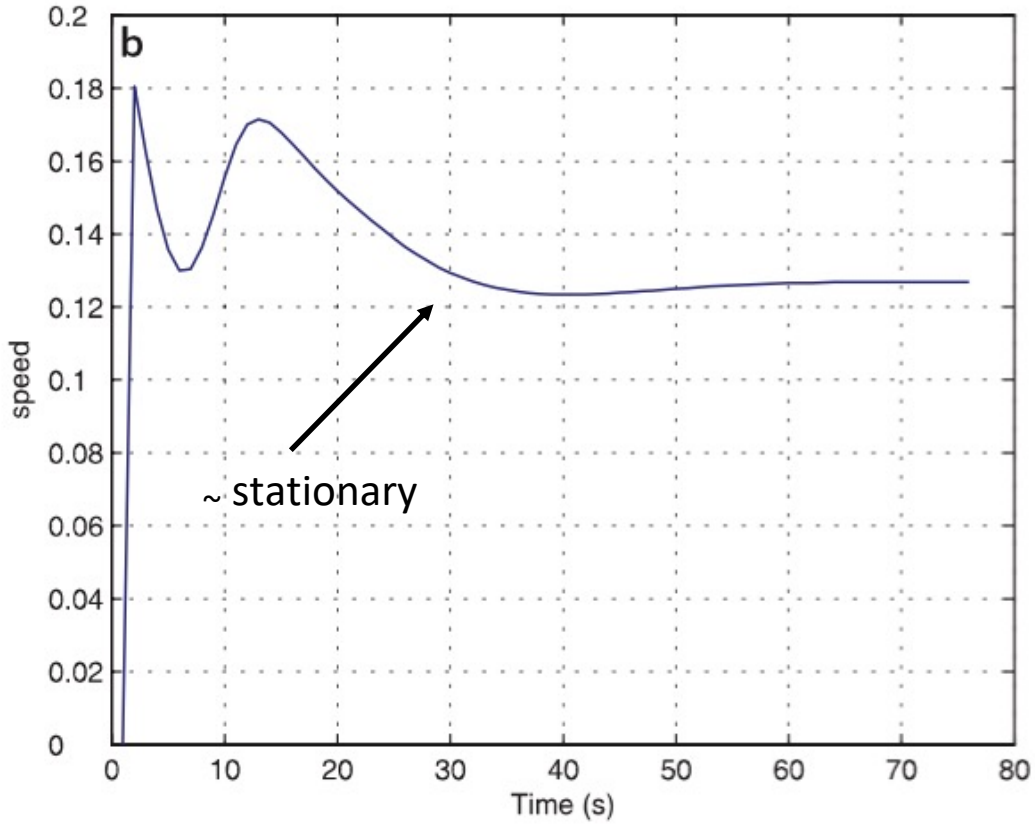
Angle of goal $(x^*,y^*)(t)$ in W

$$\theta^* = \arctan\left(\frac{y^* - y}{x^* - x}\right) = \text{atan2}(y^* - y, x^* - x) \in [-\pi, \pi)$$

$$\gamma = K_\theta(\theta^* \ominus \theta), \quad K_\theta > 0$$

Following a path (PI controller)



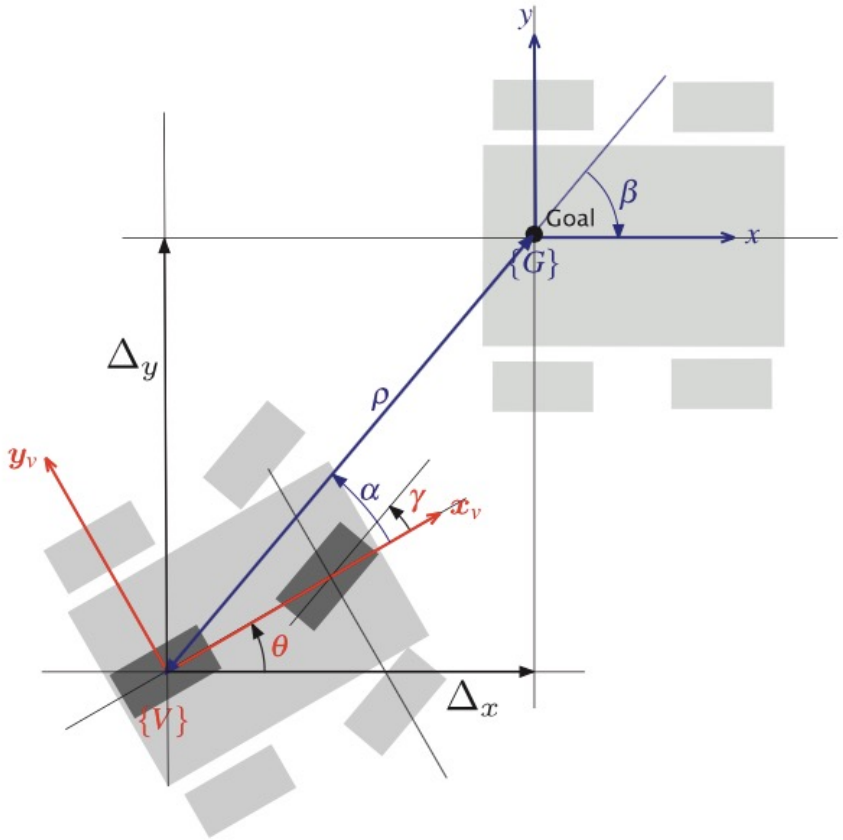


$$d^* = 0.05,\ K_\rho = 4,\ K_i = 2,\ K_\theta = 5$$

Moving to a pose (P controller)

Goal: Move to a specific pose $^{W}(x^*, y^*, \theta^*)$ in the plane

- **Control inputs**: v(t), γ(t)
- **Current known state**: robot's pose $[x \;\; y \;\; \theta](t)$ and goal pose (in the W frame)
- **Error vector:** [distance from goal's position, orientation w.r.t. goal pose orientation]



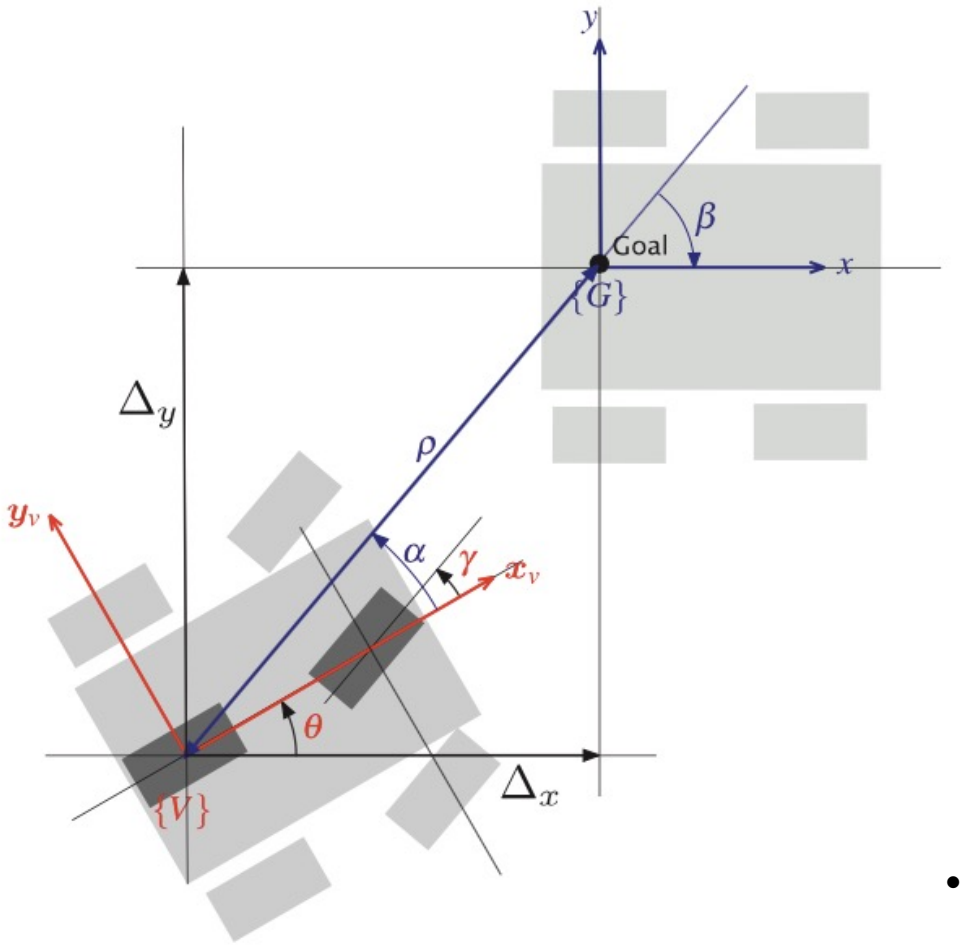
- Without loss of generality, the goal pose {G} is assumed to be coinciding with the world inertial frame {W}.
- ✓ Therefore, moving to the goal pose is the same as to make the transformation from frame {V} to frame {G}

✓ **Polar coordinates** are convenient to represent the transformation between {V} and {G}:

- $\rho > 0$ is the the error distance between robot and goal state
- α is the the orientation of the vector $\vec{\rho}$ w.r.t. V
- $\beta = -\theta - \alpha$ is the orientation of the vector $\vec{\rho}$ w.r.t. G

Moving to a pose (P controller)

- **Control inputs**: v(t), γ(t)
- **Current known state:** robot's pose [x y θ](t) and goal pose (in the W frame)
- **Error vector:** [goal distance ρ, orientation α of $\vec{\rho}$ w.r.t. to {V}, orientation β of $\vec{\rho}$ w.r.t. to {W}]



Cartesian to Polar coordinates:

$$\begin{bmatrix}\rho\\ \alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\sqrt{\Delta_x^2+\Delta_y^2}\\ \operatorname{atan}\left(\frac{\Delta_y}{\Delta_x}\right)-\theta\\ -\alpha-\theta\end{bmatrix}$$

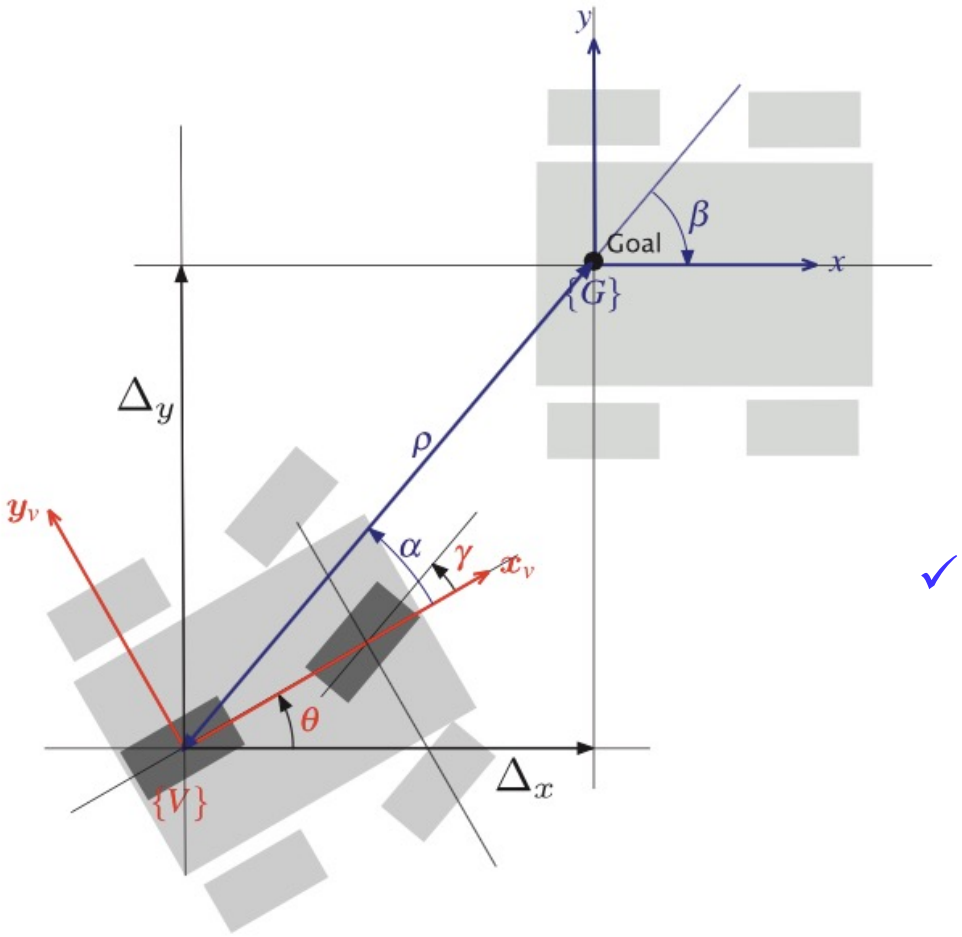
$$\dot{x} = v(t)\cos(\theta(t))$$
$$\dot{y} = v(t)\sin(\theta(t))$$
$$\dot{\theta} = \frac{v(t)}{L}\tan\gamma(t) = \omega$$

$$\begin{bmatrix}\dot{\rho}\\ \dot{\alpha}\\ \dot{\beta}\end{bmatrix} = \begin{bmatrix}-\cos(\alpha) & 0\\ \frac{\sin(\alpha)}{\rho} & -1\\ -\frac{\sin(\alpha)}{\rho} & 0\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}$$

- Holds for {G} in front of {V} $\leftrightarrow \alpha \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$

- For {G} not in front of {V} ↦ the forward direction is redefined by setting v = -v, ω = -ω, and all matrix coefficients change sign

Moving to a pose (P controller)



$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos(\alpha) & 0 \\ \frac{\sin(\alpha)}{\rho} & -1 \\ -\frac{\sin(\alpha)}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

- For {G} in front of {V} $\leftrightarrow$ $\alpha \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$

✓ **Control laws** on v and γ that aim to bring to zero the polar coordinates → {V} ≡ {G}

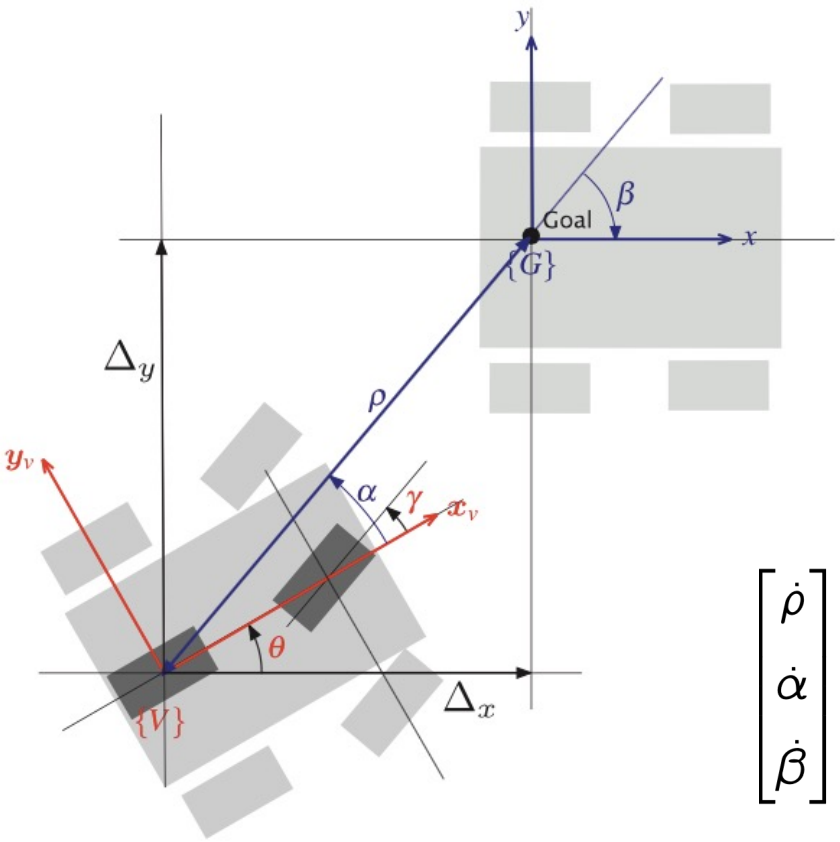
$$v(t) = K_\rho \rho(t)$$

$$\gamma(t) = K_\alpha \alpha(t) + K_\beta \beta(t)$$

○ For reaching a generic goal pose, instead of $(0,0,0)$, the following transformation can be used:

$$x' = x - x^*,\ y' = y - y^*,\ \theta' = \theta,\ \beta = \beta' + \theta^*$$

Moving to a pose (P controller)



- What are good values for the **gains**?
- Is the dynamical system **stable**?
- Does it **converge**?

$$v(t) = K_\rho \rho(t)$$
$$\gamma(t) = K_\alpha \alpha(t) + K_\beta \beta(t)$$

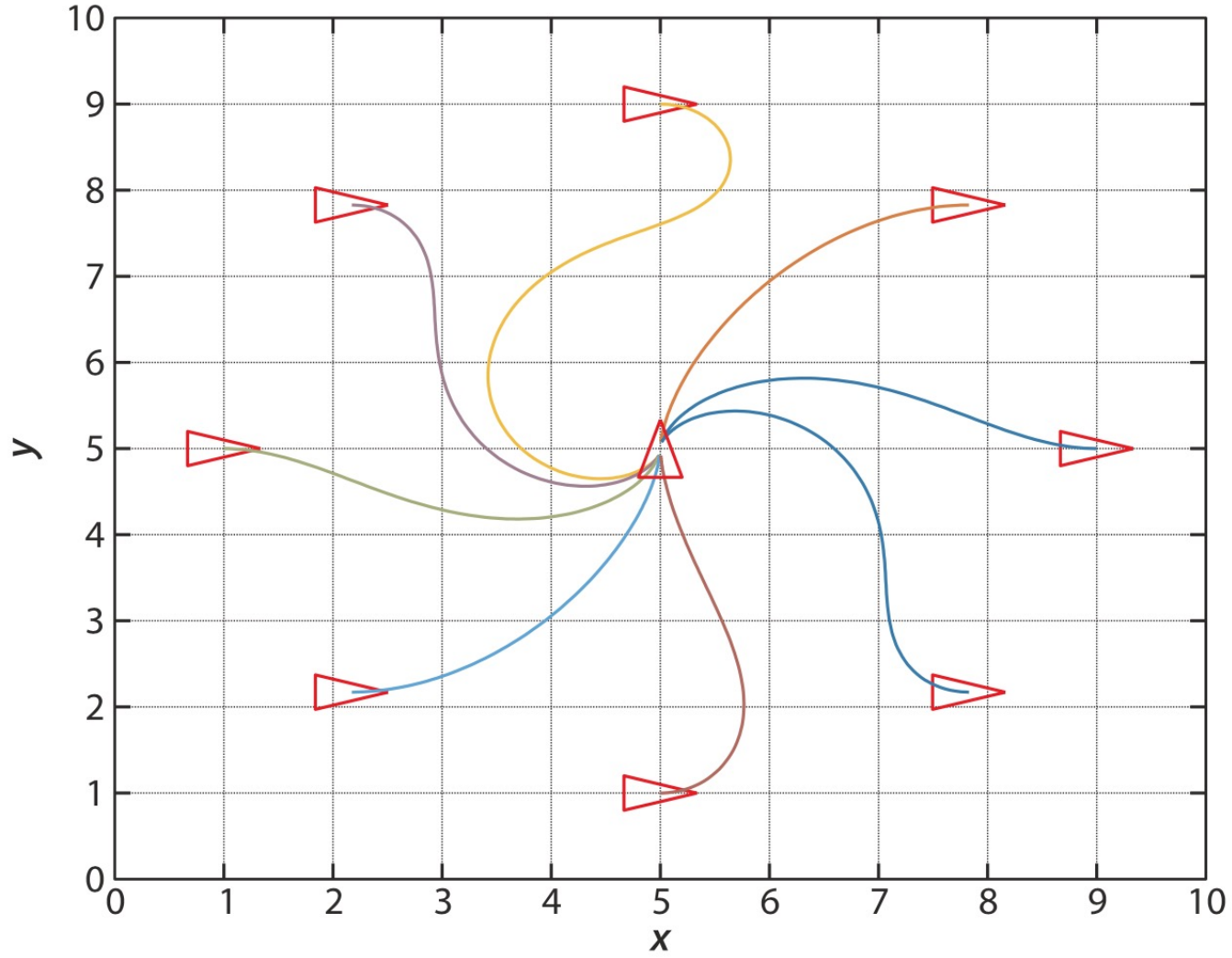
$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos(\alpha) & 0 \\ \frac{\sin(\alpha)}{\rho} & -1 \\ -\frac{\sin(\alpha)}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -K_\rho \rho \cos(\alpha) \\ -K_\rho \sin(\alpha) - K_\alpha \alpha - K_\beta \beta \\ -K_\rho \sin(\alpha) \end{bmatrix}$$

Non-linear Dynamical system

Asymptotically stable as long as:

$$K_\rho > 0, \quad K_\beta < 0, \quad K_\alpha - K_\rho > 0$$

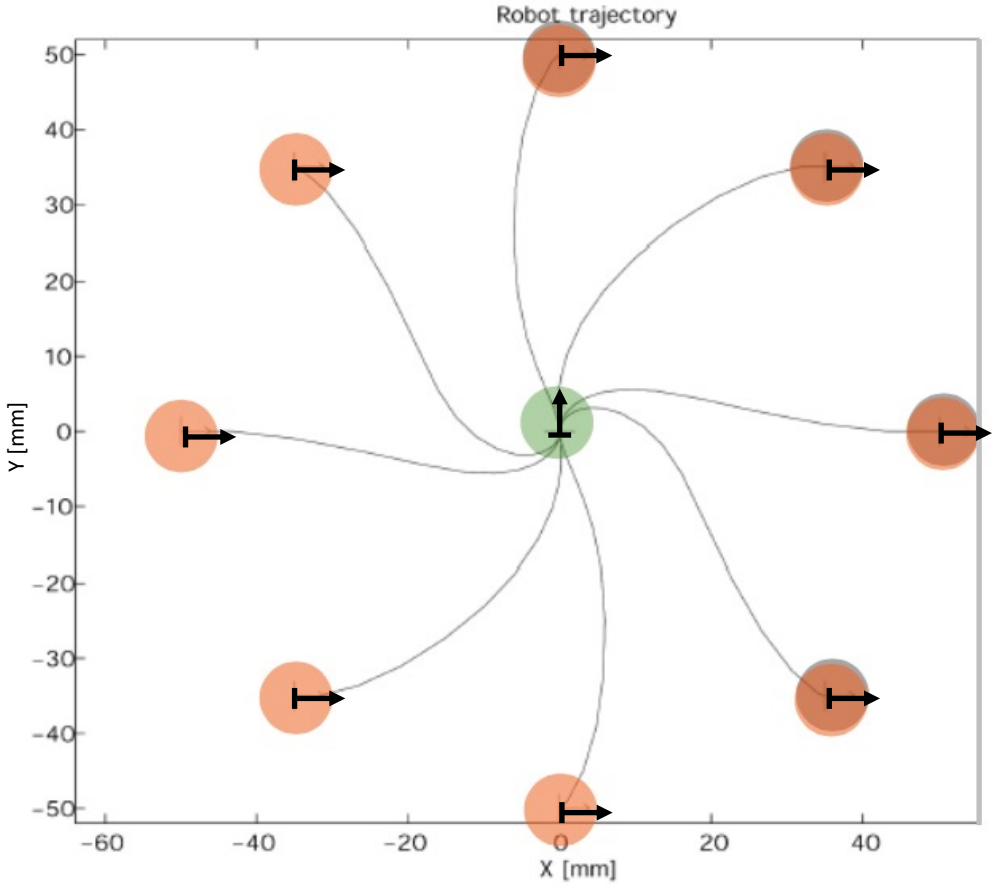
Moving to a pose (P controller)

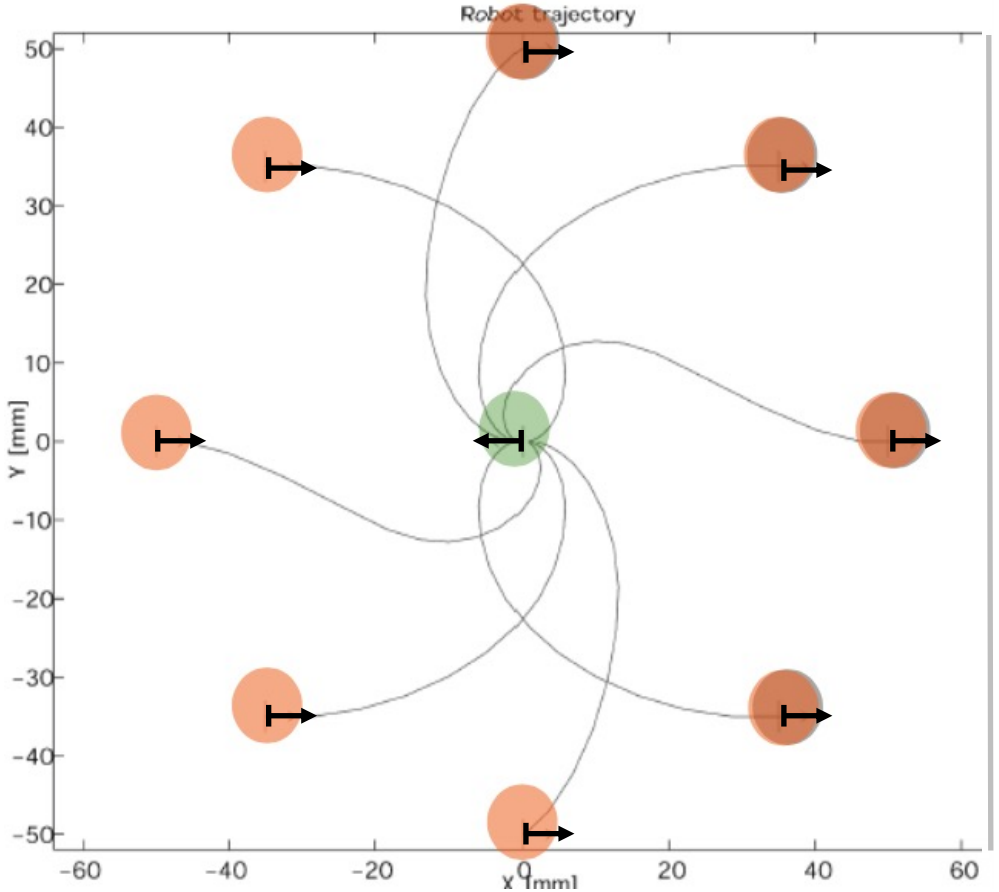


In some cases the robot has backed into the goal pose

$$K_\rho = 3, \quad K_\beta = -3, \quad K_\alpha = 8$$

Moving to a pose (P controller)





Different goal orientations produce
rather different trajectories

$$K_\rho = 3, \quad K_\beta = -1.5, \quad K_\alpha = 8$$

Issues and next steps….

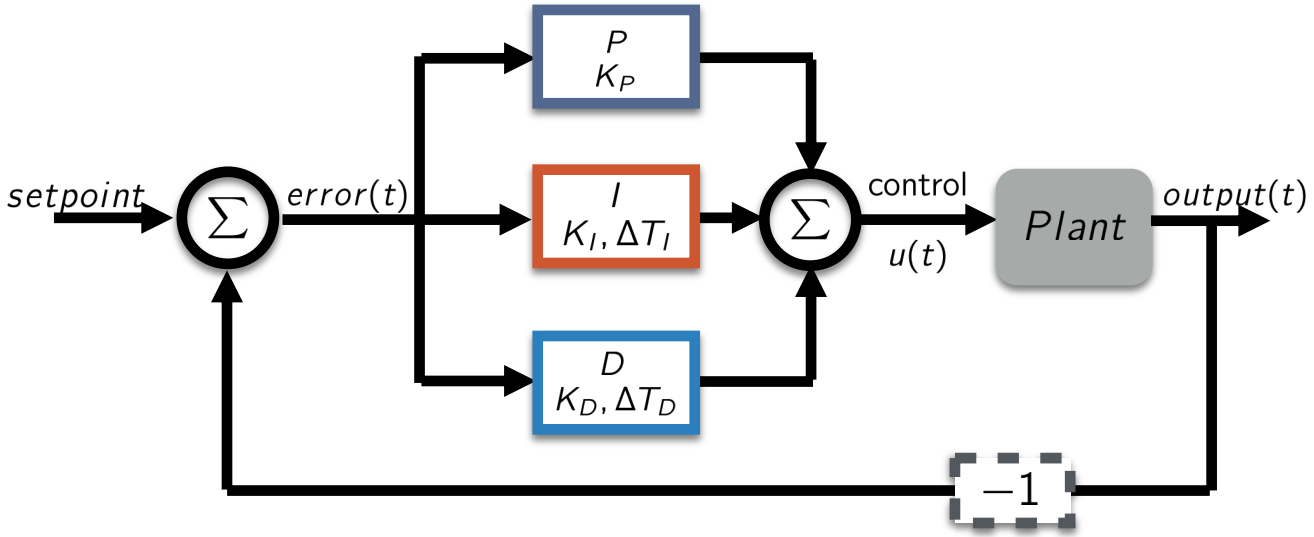
- The systems describing robot and error dynamics used to solve inverse general kinematics problems (with feedback-based control) are not linear
- Feedback-based approaches require setting a number of gain parameters when adopting a PID controller: how do we set the gain values?
- Is the (linear) PID approach appropriate, in general, for the type of inverse kinematics problems we are dealing with?
- How do we get some guarantees about the trajectory followed by the robot: does the robot converge to the desired setpoint? Once reached, is the setpoint stable? Will convergence happen in finite time?

Note: Next slides haven't been covered in the class, however they shall be quite accessible. I will try to make a short video to go through them

Limitations of PID controllers

$$u(t) = K_p e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt}$$

$$u(t) = K_p \left(e(t) + \frac{1}{\Delta T_I} \int_0^t e(t)dt + \Delta T_D \frac{de(t)}{dt} \right)$$

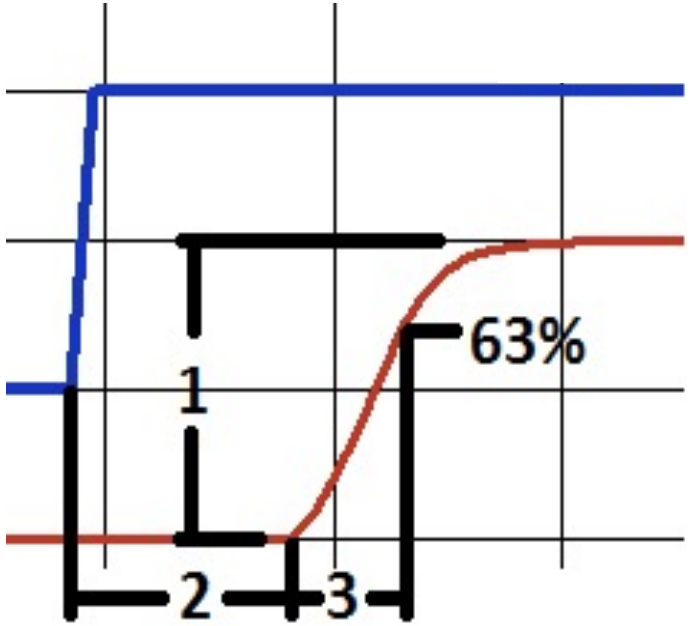


- Work without the help/need of a model of the dynamics of the system/plant being controlled
  → Potentially (usually) sub-optimal
- Provide a feedback-based linear control, since the control signal u(t) is linear in the error (and in linear operators of the error)
- In general, sub-optimal when plant's dynamics is *not* linear:

$$o(t) = A(t)o(t-1) + u(t) + \nu(t) \quad \text{Goal:} \quad |s(t) - o(t)| \to 0$$

$$\Rightarrow u(t) \approx s(t) - A(t)o(t-1)$$

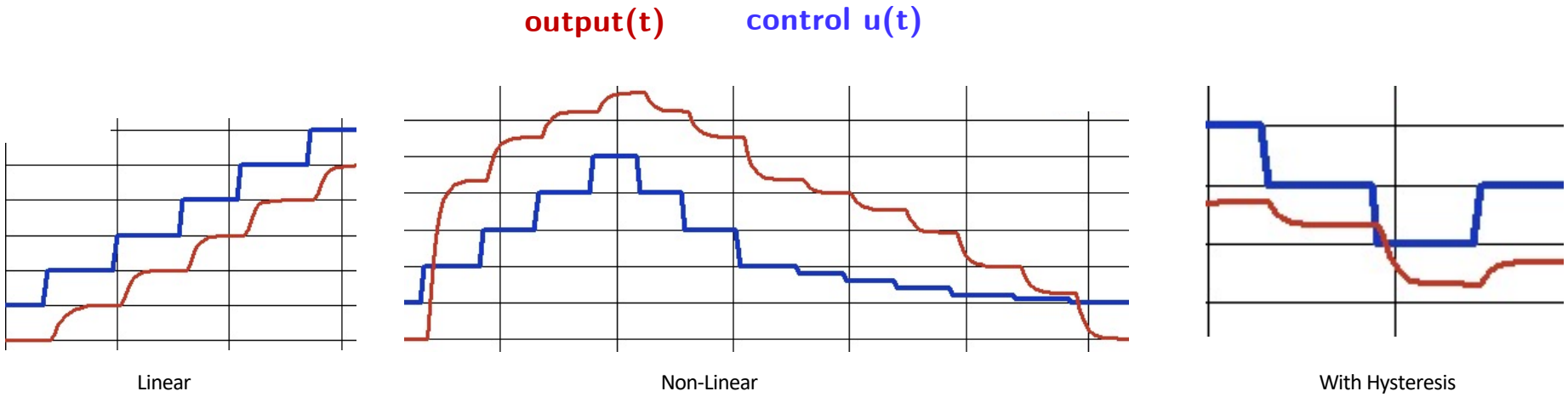
PID's response figures



Response factors:

1. **Step response gain***:* Amount of output change resulting from a **step unit change** in control u(t)
2. **Step response time***:* Time from when u(t) changes until the change **starts to be seen** in output(t)
3. **Step characteristic time***:* The time for output(t) to reach its new level. Because the output usually approaches its new level asymptotically, the **characteristic time constant** is the time it takes the system's step response to reach (-1)1/e, or about 63% of the distance from the initial to its final value

PID's response figures

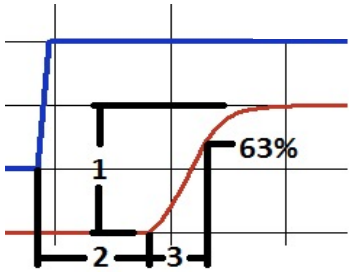


Response type:

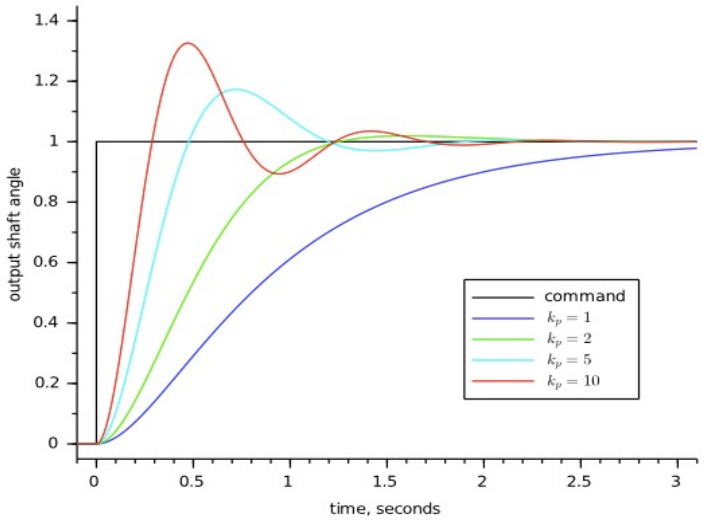
1. **Linearity of response**: The same change in u(t) through the whole scale results in a similar change in output(t) at each point
2. **Non-Linearity of response**: A change on one part of the u(t) range results in more output(t) change than the same u(t) change in a different range
3. **Hysteresis of response**: A different output(t) is produced for the same control input u(t) depending on *history*, whether the u(t) went up or down right before

Tuning PID's gains: P gain

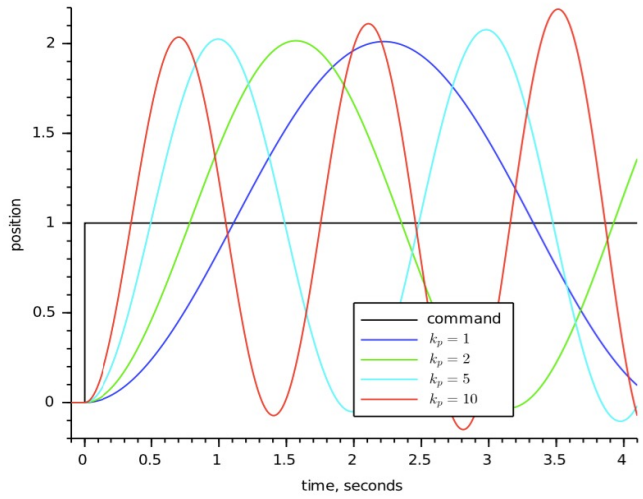
- **Proportional gain:** Contributes to the control signal u(t) with a value proportional to e(t), or, equivalently to output(t) (it's just a matter of scale given that the (setpoint - output(t)) is linear). P corrects u(t) based on upsets as they happen and proportionally to them (reactive control)
  - **High gain:** Strong reaction to error, Lower steady-state error, Higher overshot
  - **Low gain:** Less sensitivity, High steady-state error



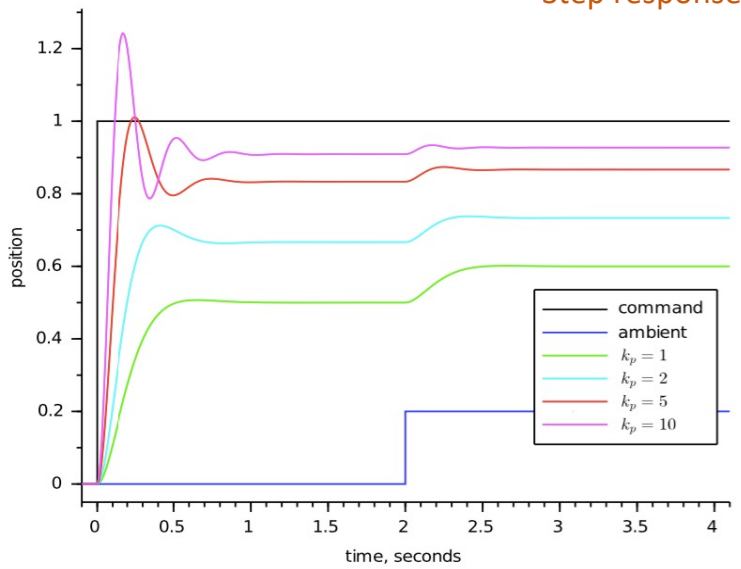
Step response



Motor shaft control by DC
Short step response time



Precision actuator
Long step response time,
Non-linear dynamics

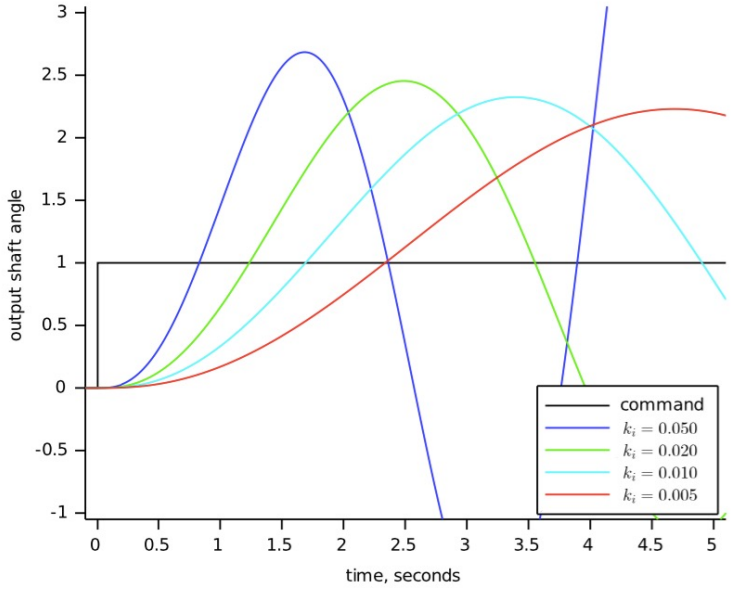


Device temperature control
Short step response time,
Steady-state error, u(t) not enough)
External disturbance

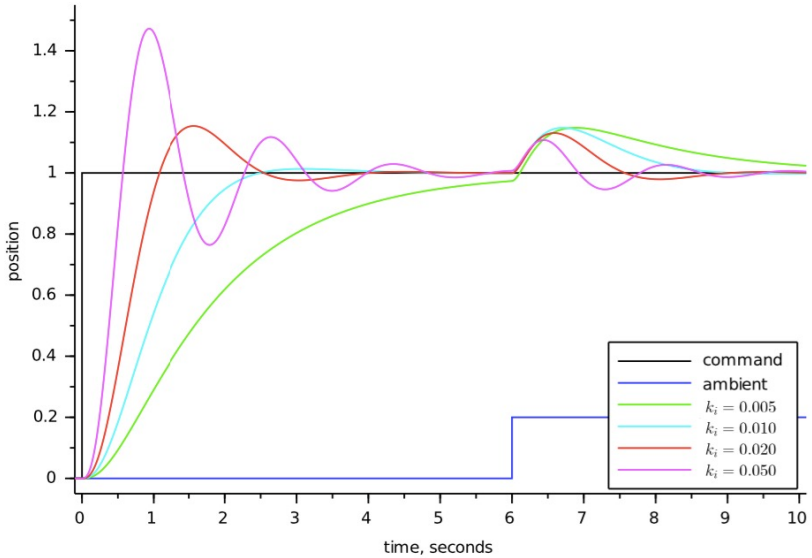
Tuning PID's gains: I Gain

- **Integral gain:** Removes steady-state error by integrating the error over time. It is used to add long-term precision to a control loop, and it is almost always used in conjunction with proportional control. Alone, it usually doesn't drive the system to stability since it brings "inertia"

  - **High gain:** More oscillatory, Faster reduction of steady-state error
  - **Low gain:** Less oscillatory, Slower reduction of steady-state error



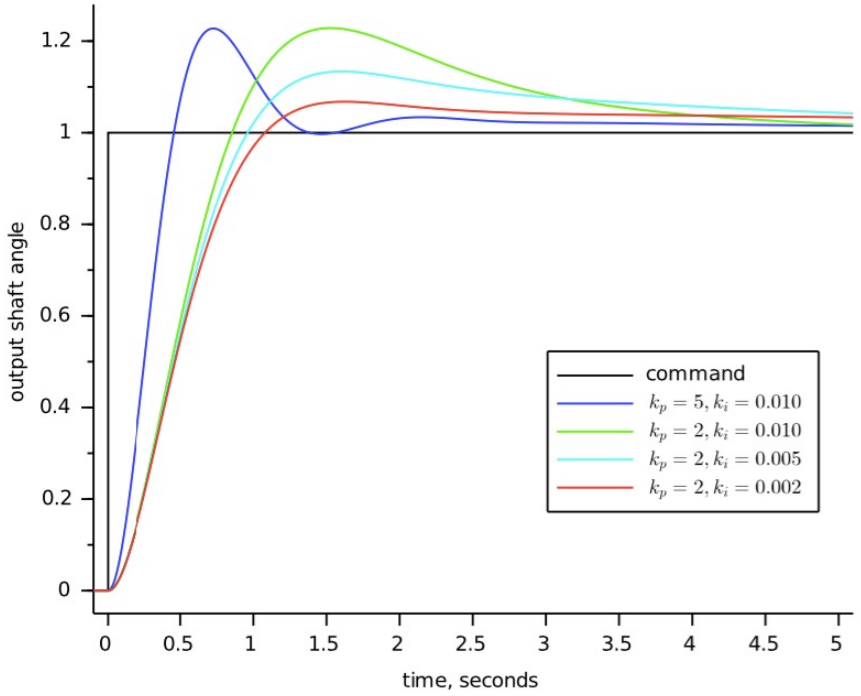
Motor shaft control by DC



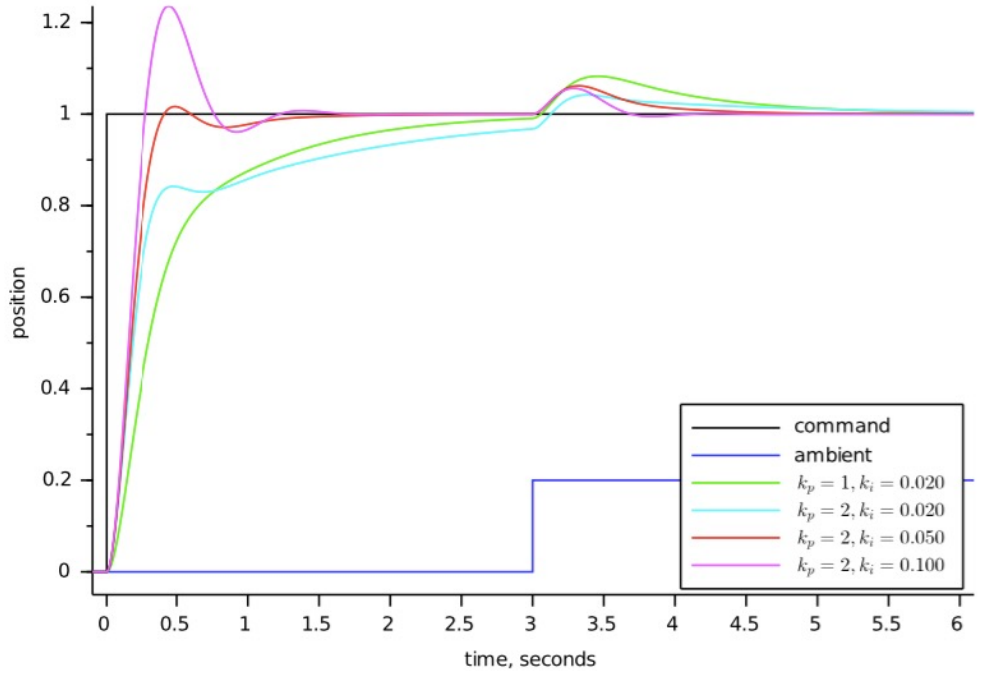
Device temperature control

Tuning PID's gains: P+I Gain

- **Integral Gain + Proportional Gain:** React fast (P) and let the steady-state error going to zero (I), but overall hits the setpoint later than when using P



Motor shaft control by DC

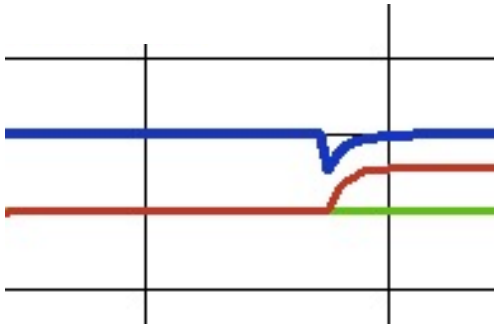


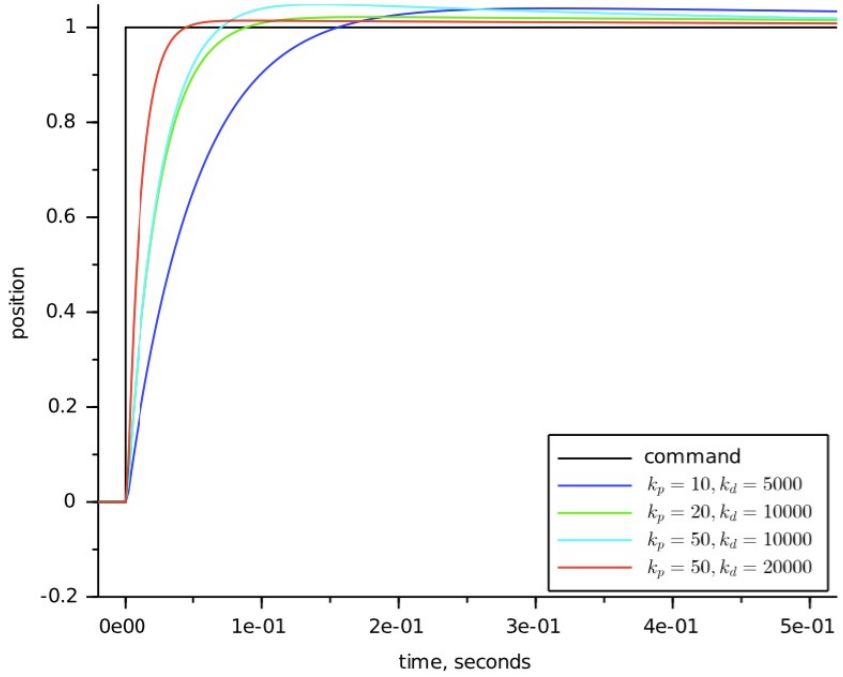
Device temperature control

Tuning PID's gains: D Gain

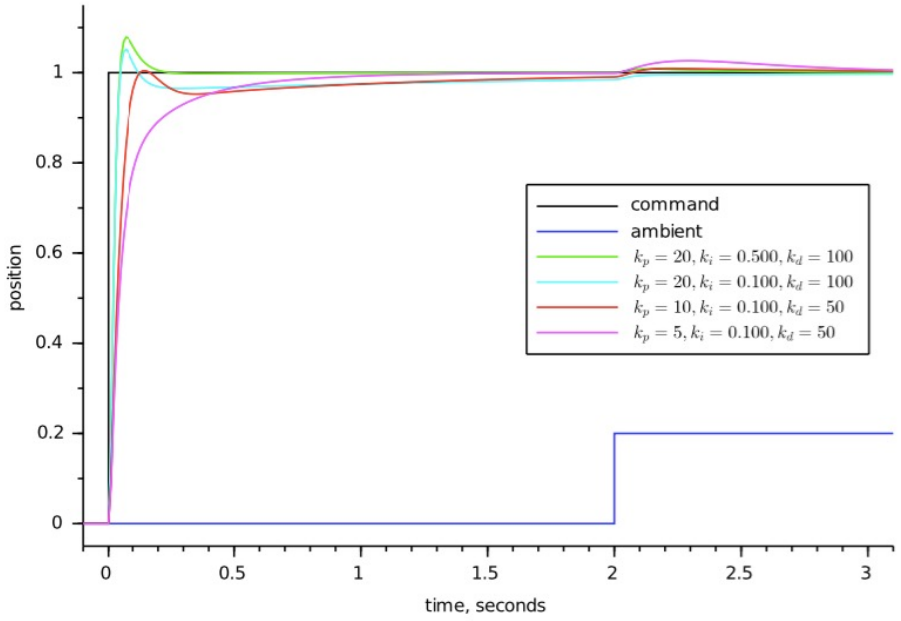
- **Differential gain:** It is also called preact because it allows the loop to anticipate upsets as they begin to happen, spot by a rate of change in the error, and then react quicklv.

✓ Reduces the oscillatory behavior and overshot of the response.





Precision actuator with PD



Temperature control with PID

Heuristics for P,I,D tuning

TABLE 1 Effects of independent P, I, and D tuning on closed-loop response.
For example, while K_I and K_D are fixed, increasing K_P alone can decrease rise time, increase overshoot, slightly increase settling time, decrease the steady-state error, and decrease stability margins.

	Rise Time	Overshoot	Settling Time	Steady-State Error	Stability
Increasing K_P	Decrease	Increase	Small Increase	Decrease	Degrade
Increasing K_I	Small Decrease	Increase	Increase	Large Decrease	Degrade
Increasing K_D	Small Decrease	Decrease	Decrease	Minor Change	Improve

Challenge: Their effects are intertwined; they need to be set altogether!

(One typical, manual) Heuristic recipe:

1. Start with a P controller and set P gain to a value K_P that's low enough to prevent appearing of major oscillations in system's response.
2. Add derivative D, that will help to damp oscillations when P gain will be increased (to get faster response). Start with $K_P = cK_P$, if not oscillations appear, increase it, until oscillations happen, or, if oscillations are there, decrease it until they disappear (e.g., $c = 10$)
3. Adjust P's gain to possibly increase it (by a factor 2 or 3), until oscillations appear.
4. Start the I gain setting it to about 1/100 of P's gain. With oscillations, decrease K_I; with no oscillations, increase it until oscillations happen.
5. … stop! → Try it out …

Twiddle heuristic (~coordinate ascent)

```
# Choose an initialization parameter vector
p = [0, 0, 0]
# Define potential changes
dp = [1, 1, 1]
# Calculate the error
best_err = A(p)

threshold = 0.001

while sum(dp) > threshold:
   for i in range(len(p)):
      p[i] += dp[i]
      err = A(p)

      if err < best_err:  # There was some improvement
         best_err = err
         dp[i] *= 1.1
      else:  # There was no improvement
         p[i] -= 2*dp[i]  # Go into the other direction
         err = A(p)

         if err < best_err:  # There was an improvement
            best_err = err
            dp[i] *= 1.05
         else  # There was no improvement
            p[i] += dp[i]
            # As there was no improvement, the step size in either
            # direction, the step size might simply be too big.
            dp[i] *= 0.95
```

A is a black-box function returning an error (e.g., a PID controller connected to plant)

The output of twiddle is a local minimum in PID parameter space

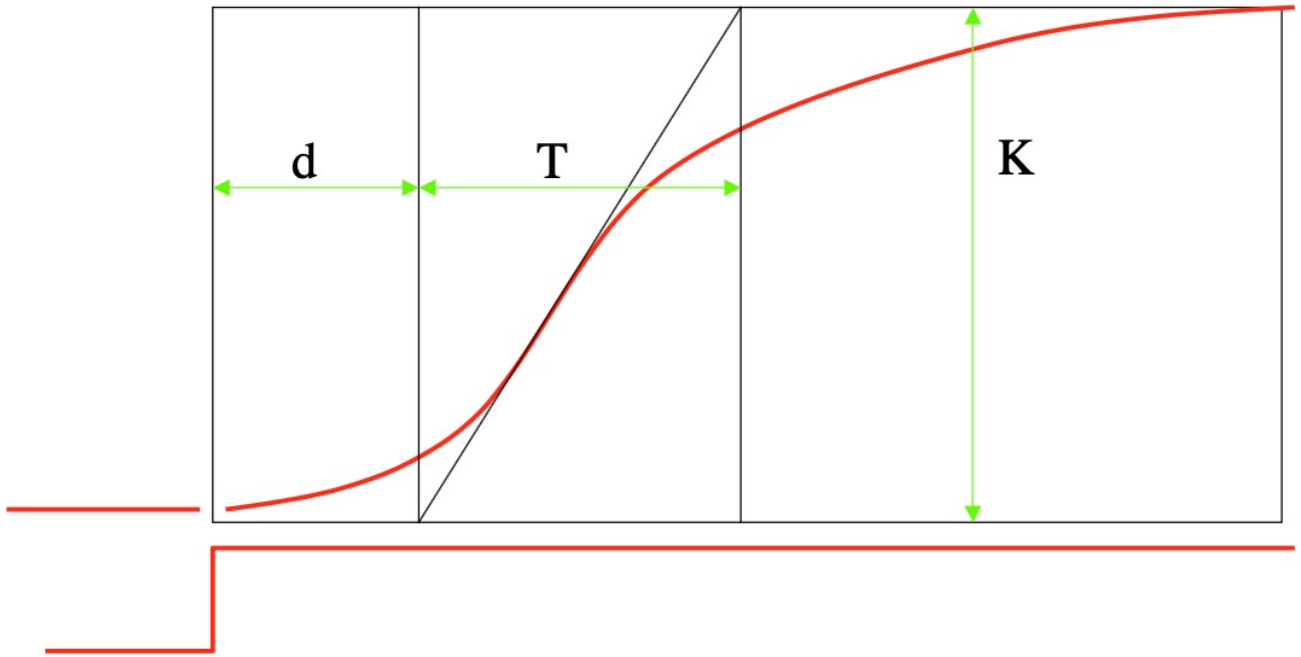
Any global heuristic optimization algorithm can be used. Good options include: Genetic Algorithms, Particle Swarm Optimization, Tabu Search …

Gain tuning of position domain PID control using particle swarm optimization
Pano, V., Ouyang, P., Robotica, 09/2014, Volume 760, Issue 6

Ziegler-Nichols Open loop tuning

Observe Open-loop response to a unit step increase.

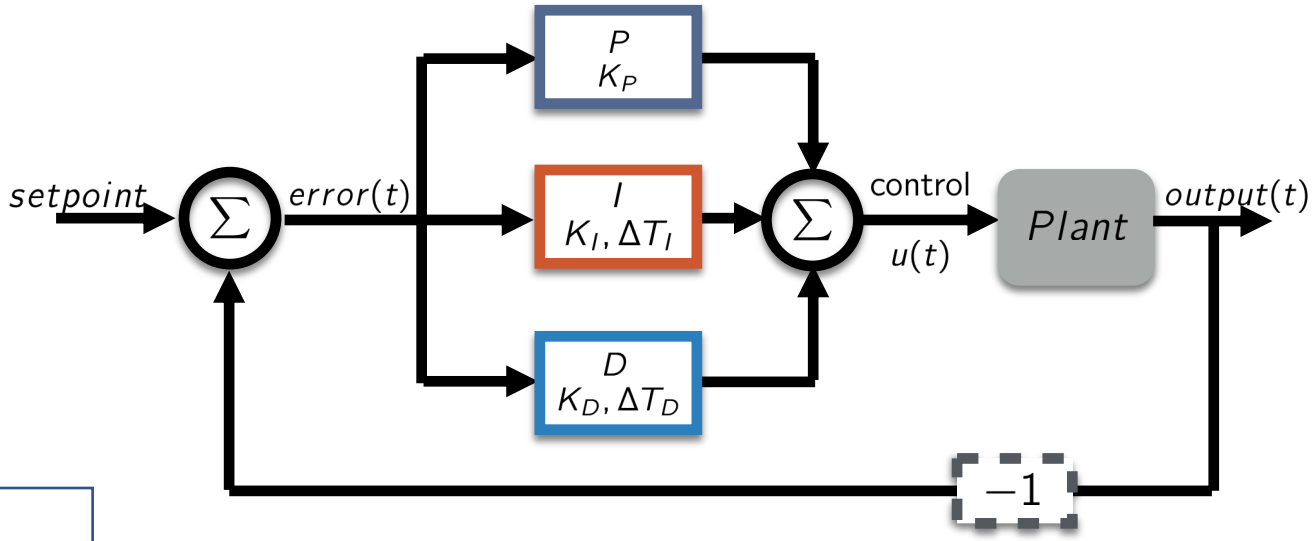
- d is deadtime
- T is the process time constant.
- K is the process gain.



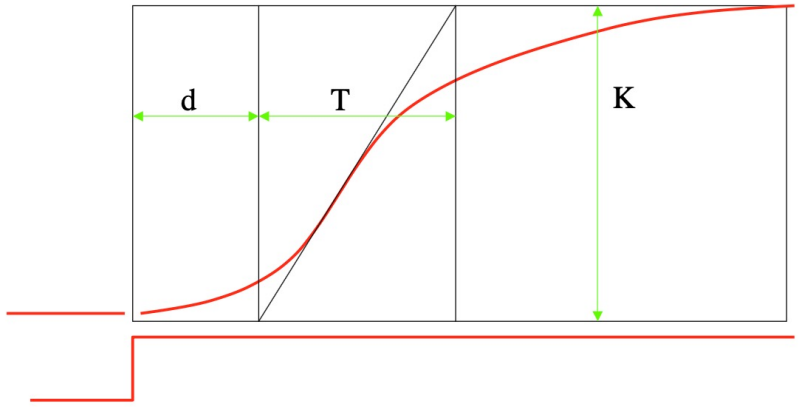
Ziegler-Nichols Open-loop tuning

$$u(t) = K_p e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt}$$

$$u(t) = K_p \left(e(t) + \frac{1}{\Delta T_I} \int_0^t e(t)dt + \Delta T_D \frac{de(t)}{dt} \right)$$



$$u(t) = -P\left[e(t) + T_I \int_0^t e\,dt + T_D \dot{e}(t)\right]$$

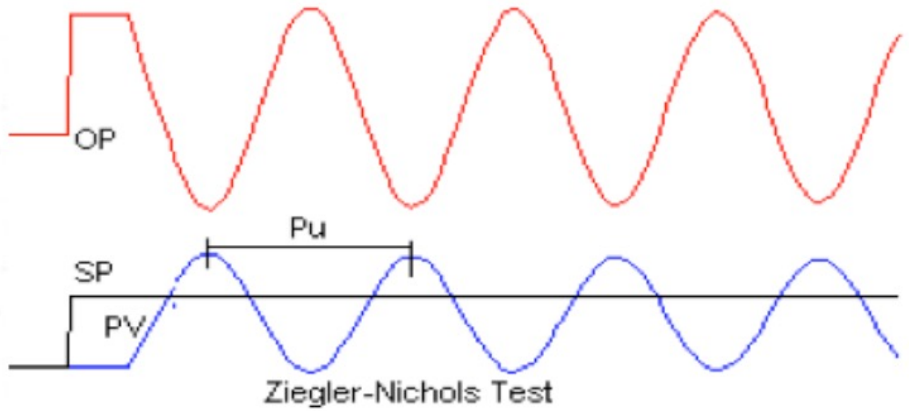


- Ziegler-Nichols says:

$$P = \frac{1.5 \cdot T}{K \cdot d} \qquad T_I = 2.5 \cdot d \qquad T_D = 0.4 \cdot d$$

Ziegler-Nichols Closed-loop tuning

1. Disable D and I action (pure P control).
2. Make a step change to the setpoint.
3. Repeat, adjusting controller gain until achieving a stable oscillation.
   - This gain is the "ultimate gain" K_u.
   - The period is the "ultimate period" P_u.



Ziegler-Nichols Closed-loop tuning

- A standard form of PID is:

$$u(t) = -P\left[e(t) + T_I \int_0^t e\,dt + T_D \dot{e}(t)\right]$$

- For a PI controller:

$$P = 0.45 \cdot K_u \qquad T_I = \frac{P_u}{1.2}$$

- For a PID controller:

$$P = 0.6 \cdot K_u \qquad T_I = \frac{P_u}{2} \qquad T_D = \frac{P_u}{8}$$