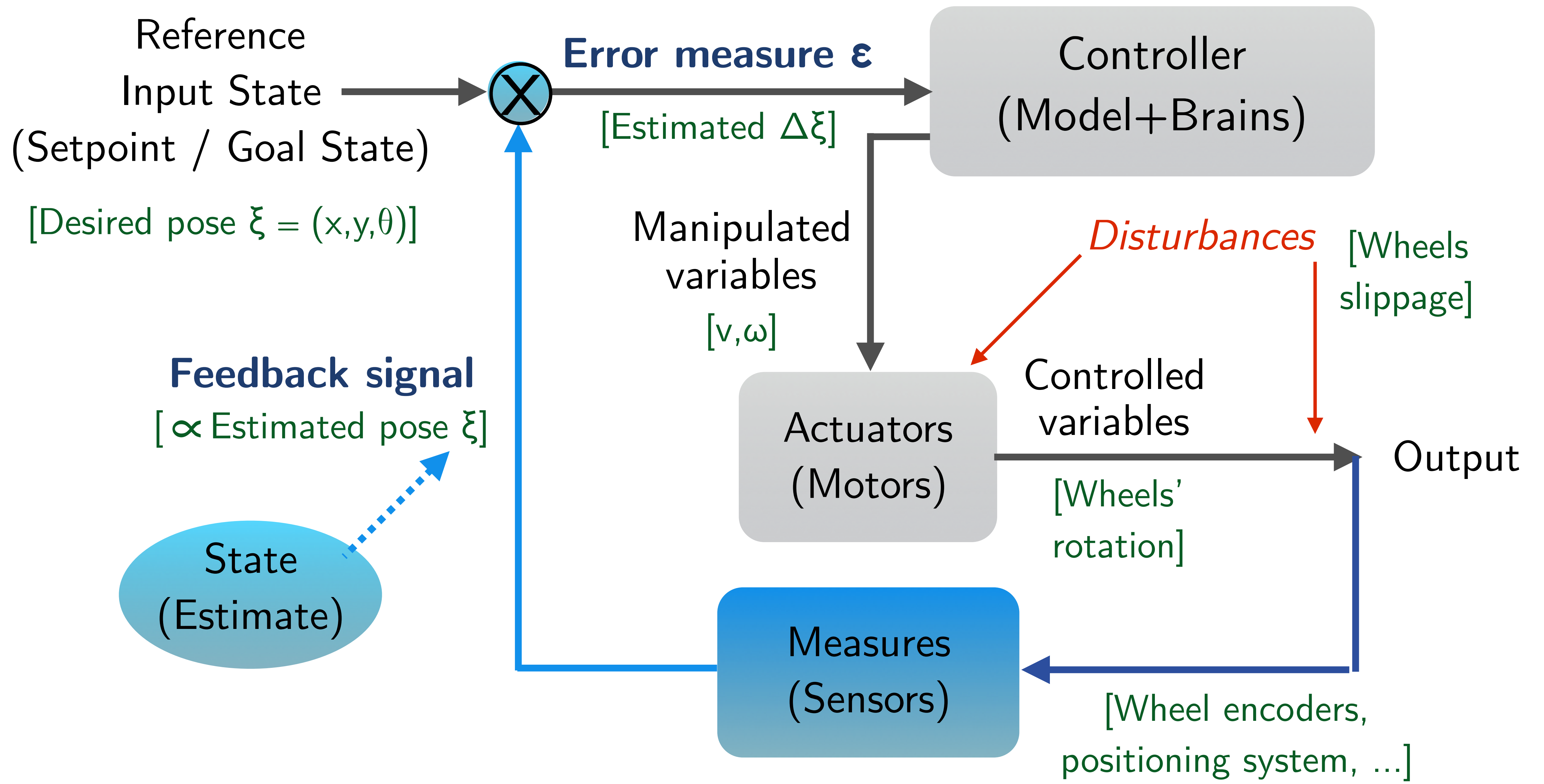
16-311-Q Introduction to Robotics

Lecture 9: Feedback-based Control I

Instructor:
Gianni A. Di Caro

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

CLOSED-LOOP VS. OPEN-LOOP CONTROL



Closed-loop: status information is *fed back* to the controller to evaluate the difference between the desired setpoint (goal) and the actual output, and to implement *corrective actions*, if needed

Open-loop: feedback information is *not* used to implement corrective actions, the assumption is that, given the inputs, the desired results will be *achieved*

TYPES OF GOAL STATES

A robot is a ***goal-driven*** physical entity

Achievement goals (typical of AI):
States the system tries to reach and once reached, the job is done

Exit from a maze,
reach a specific location or pose,
complete a construction

Maintenance goals (typical of Control):
Require a continual active work/tracking

Keep balance for a bipedal robot,
keep following a wall,
keep tracking a moving target

External goal states

Get to the kitchen
Balance a pole
Find a treasure (!)

Internal goal states

Keep battery levels in some range
Avoid excessive torque on the effectors

TYPES OF FEEDBACK-BASED CONTROLLERS

The goal of any control system is to minimize the ***Error***:
the difference between the current (as measured) state and the desired goal state

The adopted representation of the error:

has a *magnitude*
has a *direction*
has a *scale,interval*

can be *quantitative*:
continuous, discrete, binary

can be *ordinal*:
ranking, binary

The different ways the error is represented and is treated give raise to a number of different frameworks for control, we will focus on the case of <u>quantitative</u> errors in the context of PID controllers (and give a look to Bang-Bang controllers too).

A BASIC CONTROLLER FOR 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, ...).
This is a *maintenance* goal

First, simple feedback-based solution:

```
If DistanceToWall() == D
  keep moving forward
If DistanceToWall() > D
  turn by θ degrees toward the wall
Else
  turn by θ degrees away from the wall
```

Two control parameters:

- Turning angle, θ
- Sampling rate, ΔT

What do expect for:
$\theta = \pi/4$, $\Delta T = 1s$, $V = 0.5$ m/s?

Big oscillations
around the desired state!

Let's use the error!

PROPORTIONAL CONTROLLER FOR WALL FOLLOWING

Proportional controller (P):

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

$$\text{Output} = K_p \varepsilon$$

```
d = DistanceToWall()
ε = D-d
If ε < 0
  turn by Kθ ε degrees toward the wall
  move with a speed v = Kv ε
Else If ε > 0
  turn by Kθ ε degrees away from the wall
  move with a speed v = Kv ε
Else
  keep moving forward
```

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

Oscillations are still there!

DERIVATIVE CONTROLLER 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 <u>proportion to the derivative of the error</u>, using both *magnitude* and *direction* of the error

$$\text{Output} = K_d \frac{d\varepsilon}{dt}$$

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

INTEGRAL CONTROLLER

Integral controller (I):

The controller keeps track of its errors over a time window and integrates them; if the sum reaches some predefined value/threshold, a proportional corrective action is issued

$$\text{Output} = \text{K}_\text{i} \int \varepsilon(t)dt$$

- Linear response in the ***cumulative sum of errors***
- The system keeps track of repeatable/fixed errors, that are also called *steady state errors,*
- Provides a **smooth(er) response** *since past 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 (see the *Path following controller*)
- A good scenario for using error integration is that of a lawn moving robot with a consistent error in its turning mechanisms, such as some parts of the lawn are systematically left uncovered. If the robot has a mean to detect and measure this error and it cumulates over a certain threshold then a corrective action can be triggered.

PID CONTROLLERS

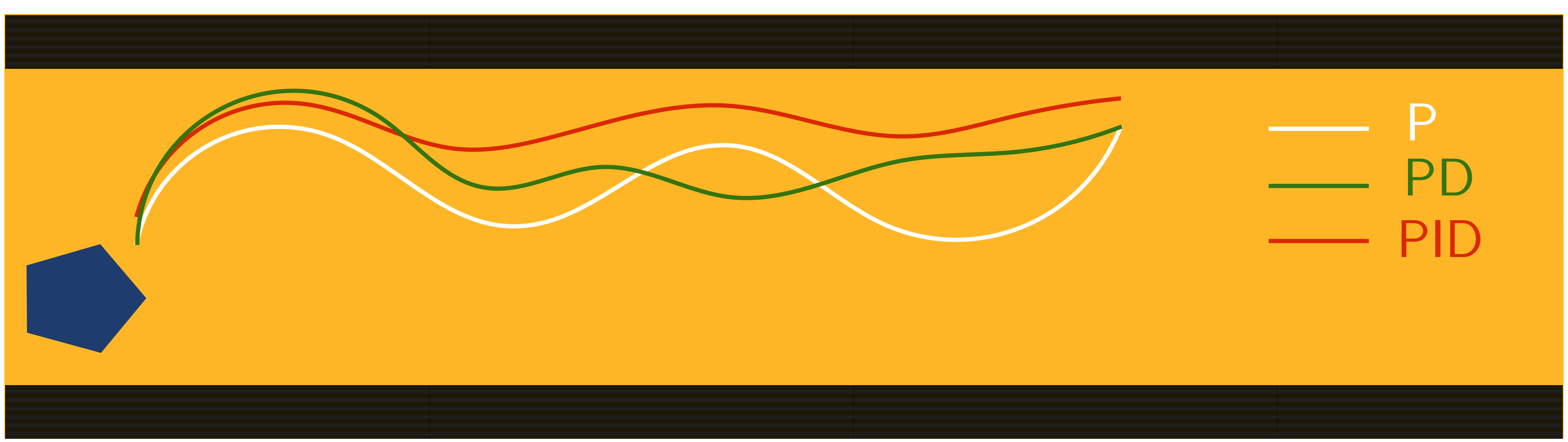
Proportional + Derivative + Integral controller (PID):

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

$$\text{Output}_{PID} = K_p \varepsilon + K_d \frac{d\varepsilon}{dt} + K_i \int \varepsilon(t) dt$$

- The three gains have to be determined and tuned jointly ... not simple to do
- The most employed type of controllers around ...

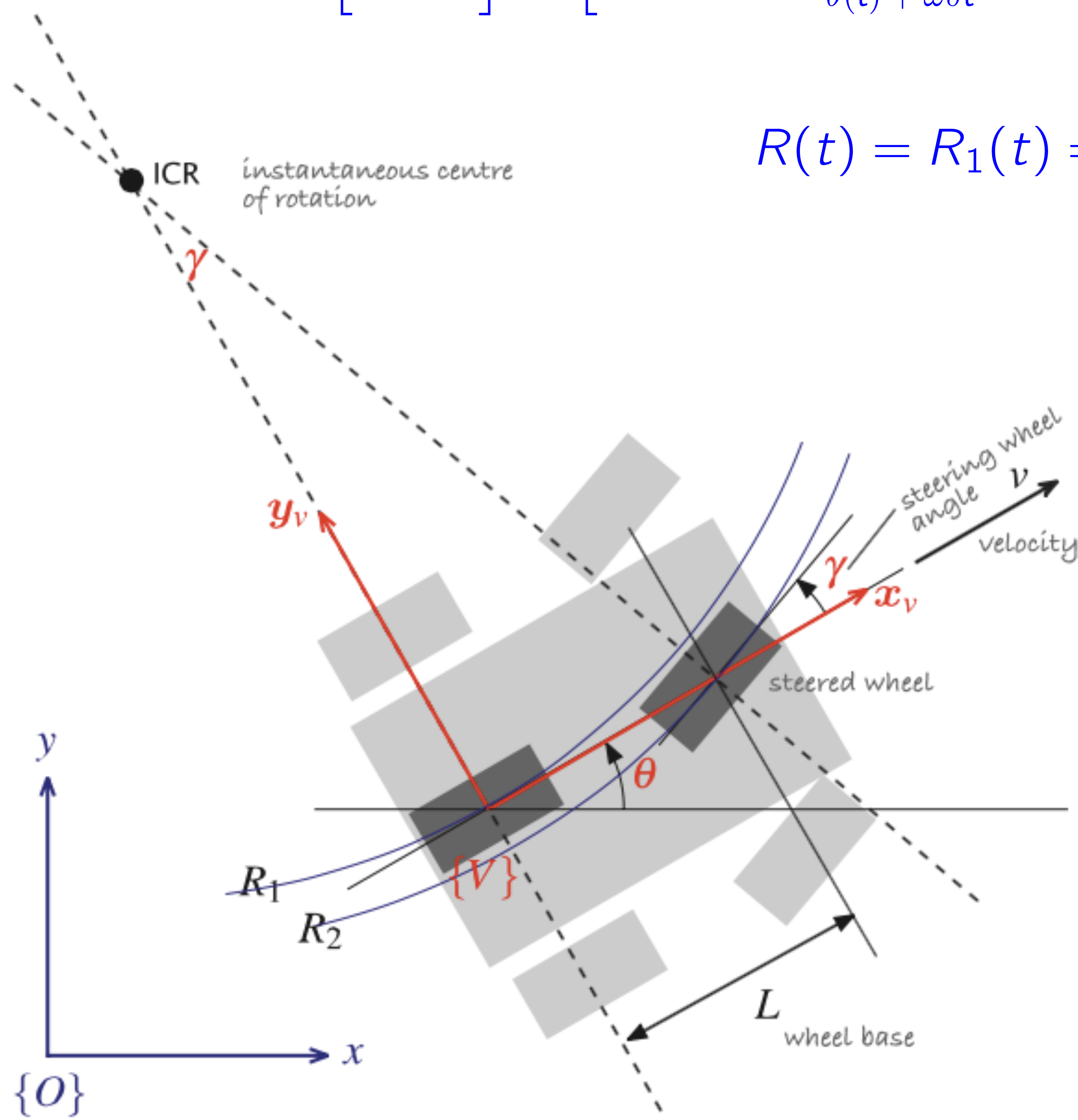
Plausible trajectories for wall following



LET'S CONTROL A STEERED ROBOT / CAR-LIKE VEHICLE, USING THE BICYCLE MODEL

$$
{}^{W}\begin{bmatrix} x(t+\delta t) \\ y(t+\delta t) \\ \theta(t+\delta t) \end{bmatrix} = \begin{bmatrix} x(t) + R(t)\Big(\sin\big(\theta(t)+\omega\delta t\big) - \sin(\theta(t))\Big) \\ y(t) - R(t)\Big(\cos\big(\theta(t)+\omega\delta t\big) - \cos(\theta(t))\Big) \\ \theta(t)+\omega\delta t \end{bmatrix} = \begin{bmatrix} x(t) + R(t)\Big(\sin\big(\theta(t)+\Delta\theta(t+\delta t)\big) - \sin(\theta(t))\Big) \\ y(t) - R(t)\Big(\cos\big(\theta(t)+\Delta\theta(t+\delta t)\big) - \cos(\theta(t))\Big) \\ \theta(t)+\Delta\theta(t+\delta t) \end{bmatrix}
$$

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



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

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

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

Control inputs:

v(t) and γ(t)

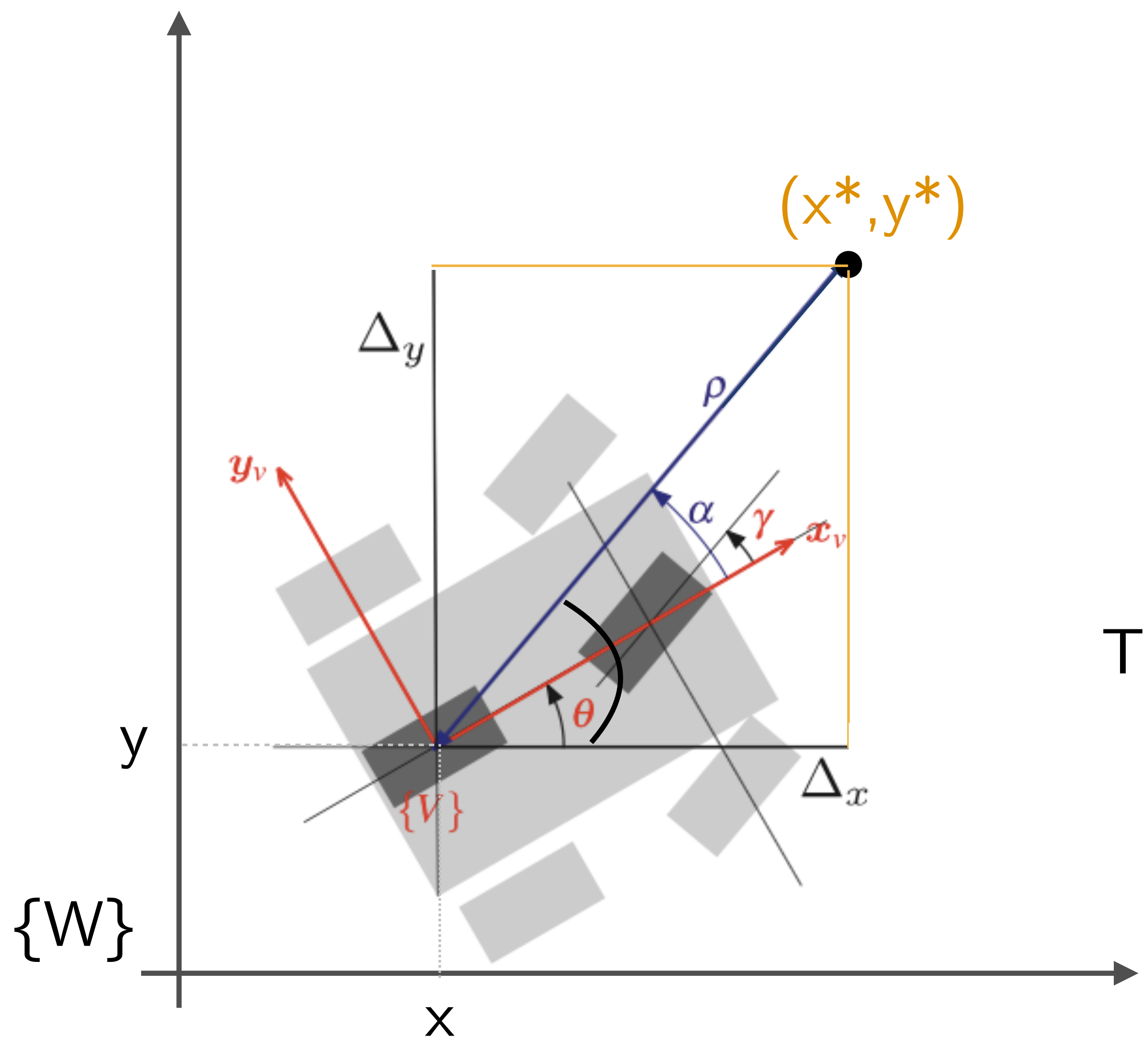
MOVING TO A GOAL POINT (P CONTROLLER)

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

Control inputs: v(t) and γ(t)

Current known state (pose): [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

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

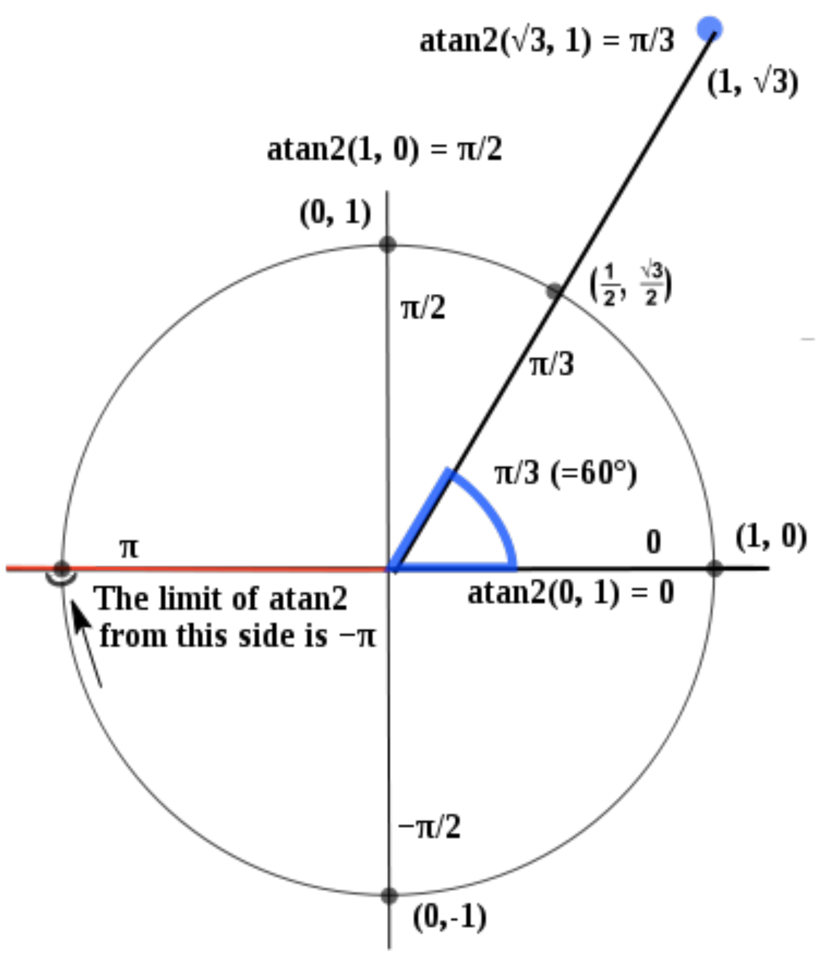
Steering proportional to the angular difference

The goal's angle $\quad \theta^* = \text{atan2}\left(\dfrac{y^* - y}{x^* - x}\right) \quad [-\boldsymbol{\pi},\boldsymbol{\pi})$

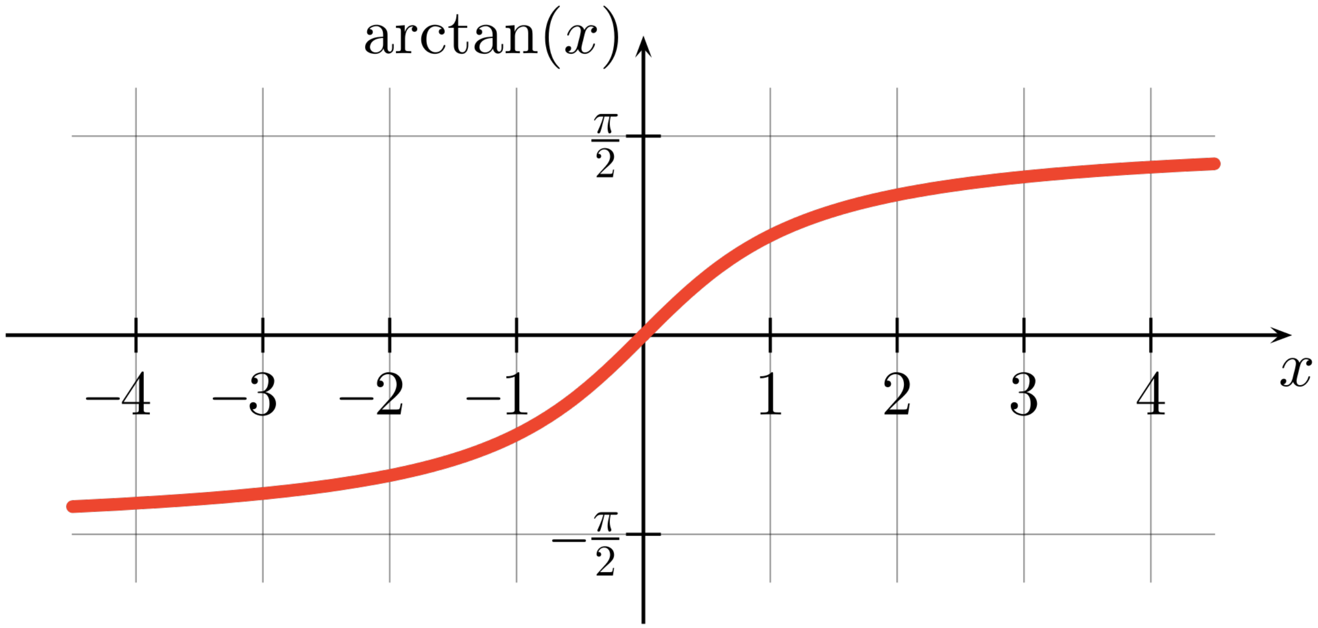
$$\gamma = K_\theta(\theta^* \ominus \theta), \quad K_\theta > 0$$

math.atan2(y, x)

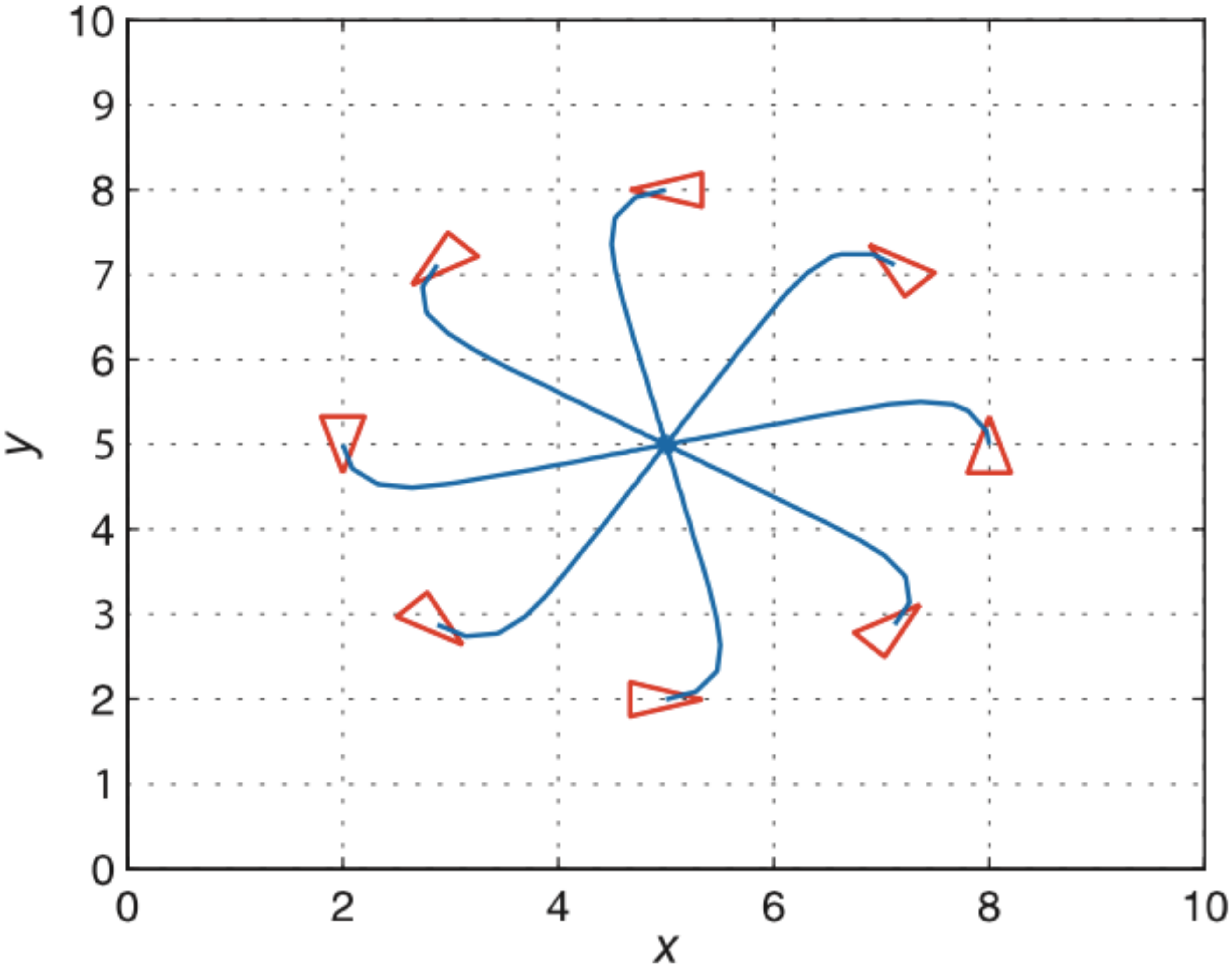
Return atan(y / x), in radians. The result is between -π and π. The vector in the plane from the origin to point (x, y) makes this angle with the positive X axis. The point of atan2() is that the signs of both inputs are known to it, so it can compute the correct quadrant for the angle. For example, atan(1) and atan2(1, 1) are both $\pi/4$, but atan2(-1, -1) is -3π/4.



One argument



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

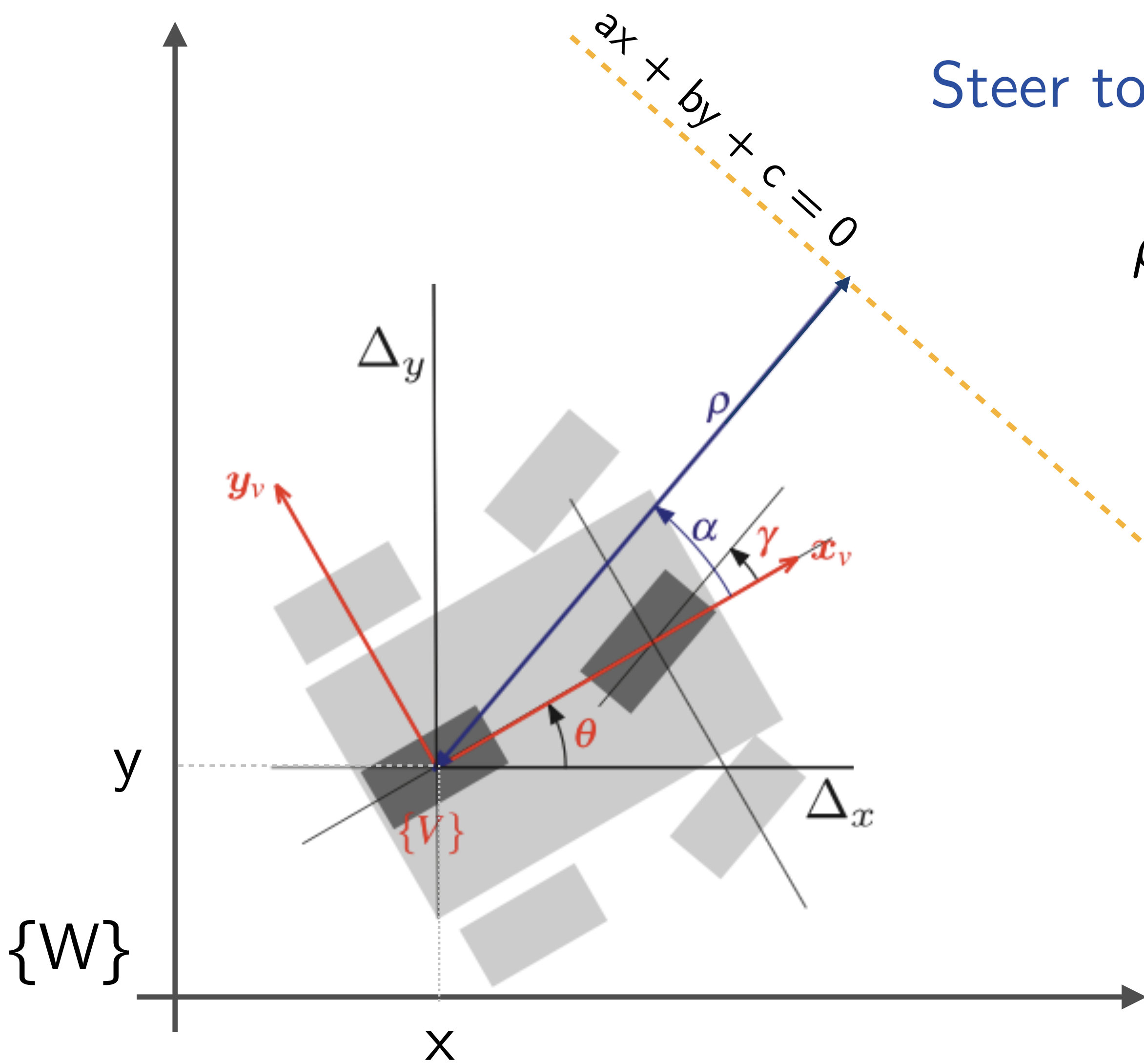
FOLLOWING A LINE (P CONTROLLER)

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

Control inputs: γ(t), while v(t) is kept constant

Current known state (pose): [x y θ](t) (in the {W} frame)

Error vector (for the heading): Distance from the line, alignment with the line



Steer to minimize the perpendicular distance ρ from the line

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

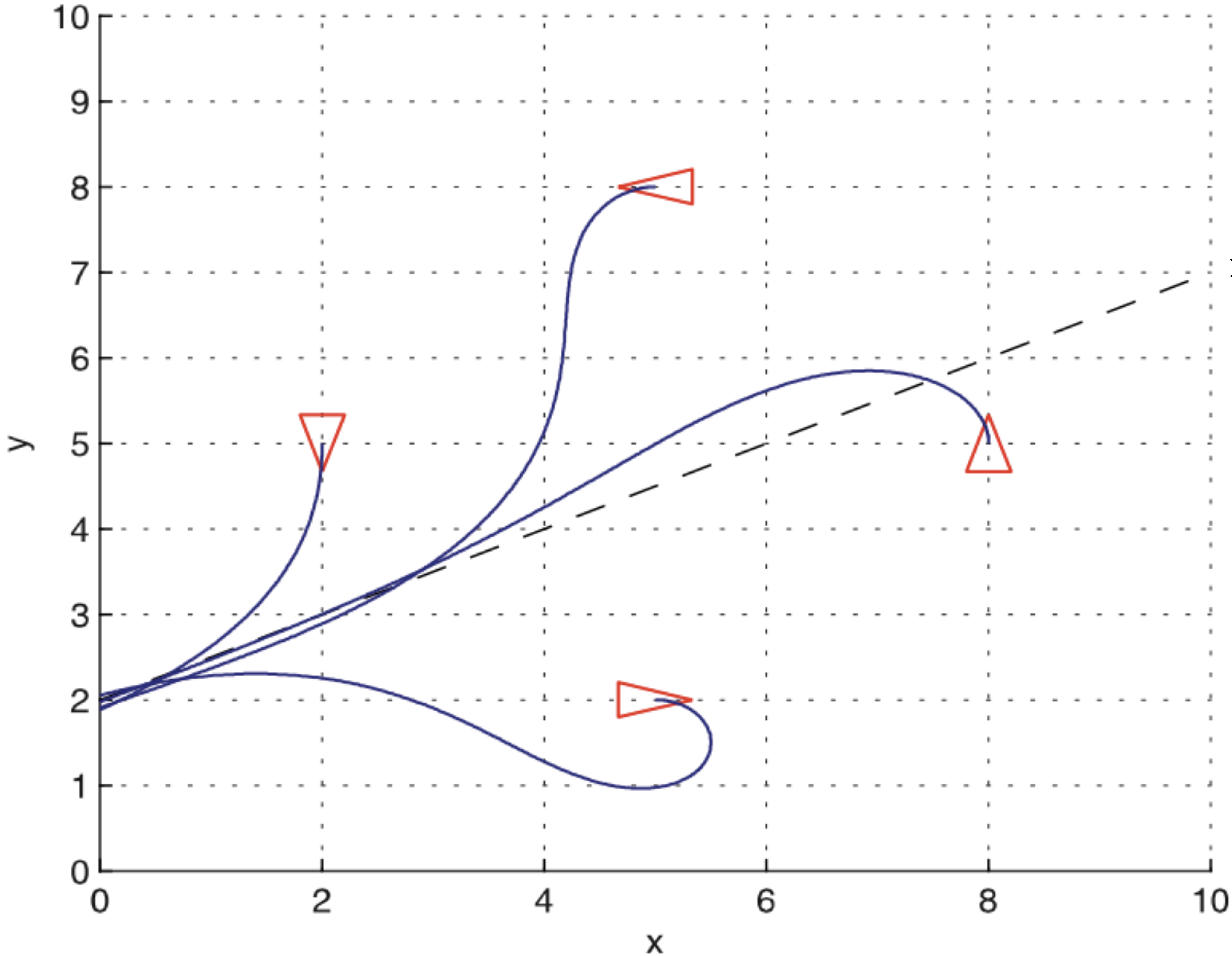
Steering to make the robot parallel to the line: proportional to the angular difference between robot's orientation and the slope of the line

The goal angle is $\theta^* = \text{atan2}\left(\frac{-a}{b}\right)$

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

Combined 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

TO BE CONTINUED ….

- Following a path
- Reaching a pose
- Doing other things …
- Stability and existence of solutions
- How to set PID gains