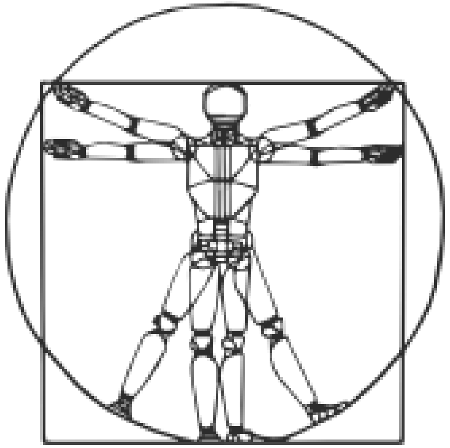
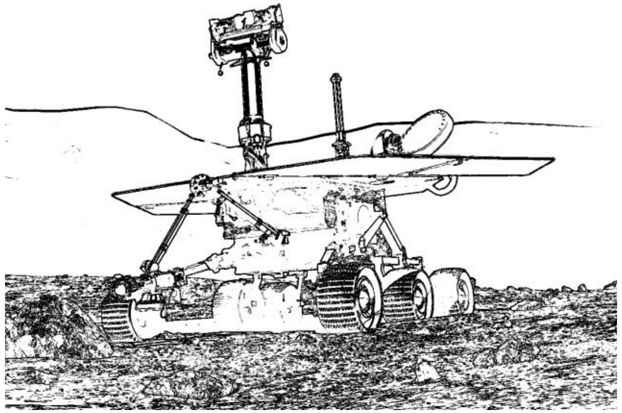




COMP417
Introduction to Robotics and Intelligent Systems

Lecture 5: PID Control

Florian Shkurti

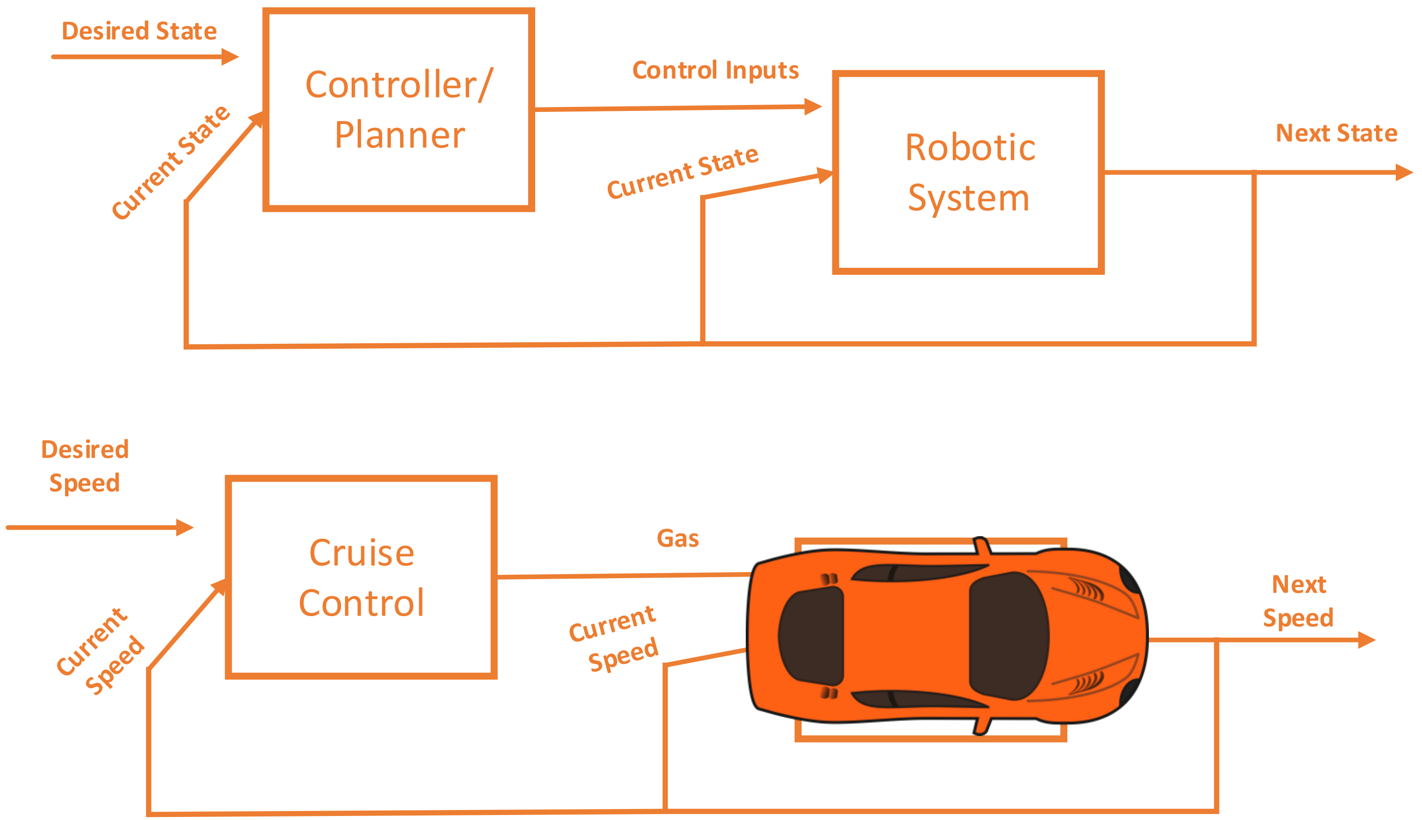
Computer Science Ph.D. student

florian@cim.mcgill.ca

McGill

MRL Mobile Robotics Lab at McGill University

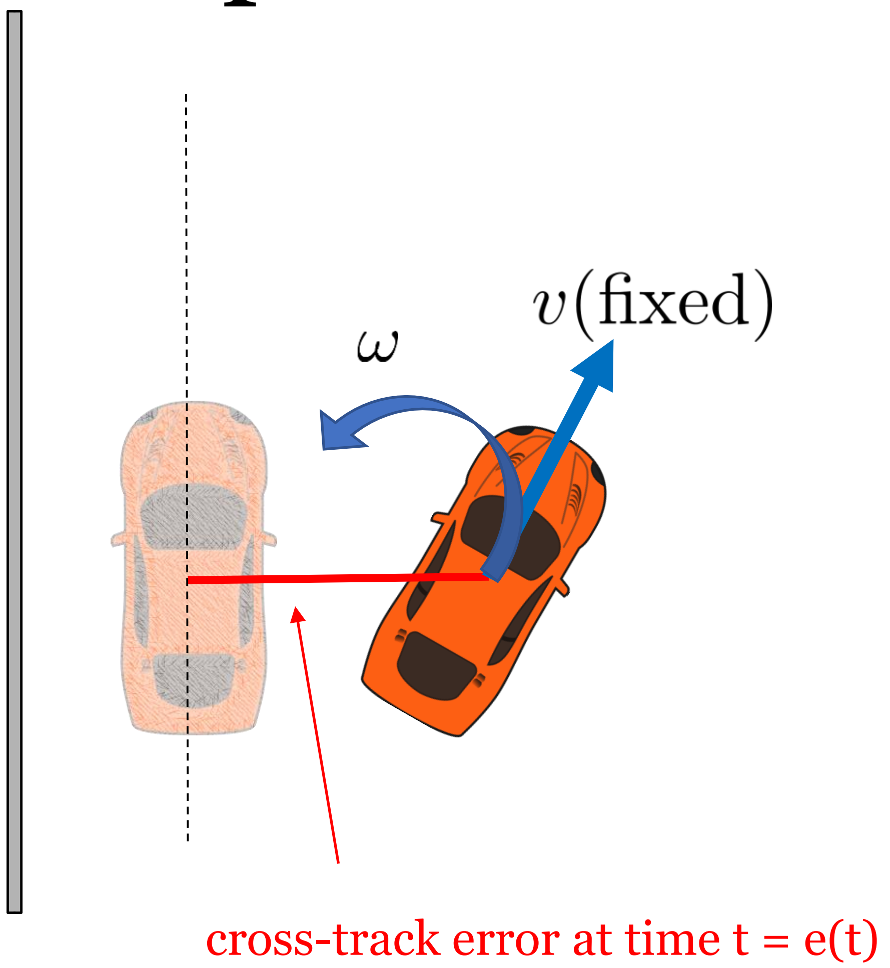
Feedback Control



Main question: what are the controls that will take the system from state A to B?

Example: wall/line following at fixed speed

You have to control the angular velocity ω



Why bother with wall/line following?

- Because it enables arbitrary path following where the line is the local tangent of the (curvy) path

Idea 1: bang-bang control

$$\omega = \begin{cases} \omega_{\max} & \text{if} \quad \text{CTE} > 0 \\ -\omega_{\max} & \text{if} \quad \text{CTE} < 0 \\ 0 & \end{cases}$$

What's wrong with this?

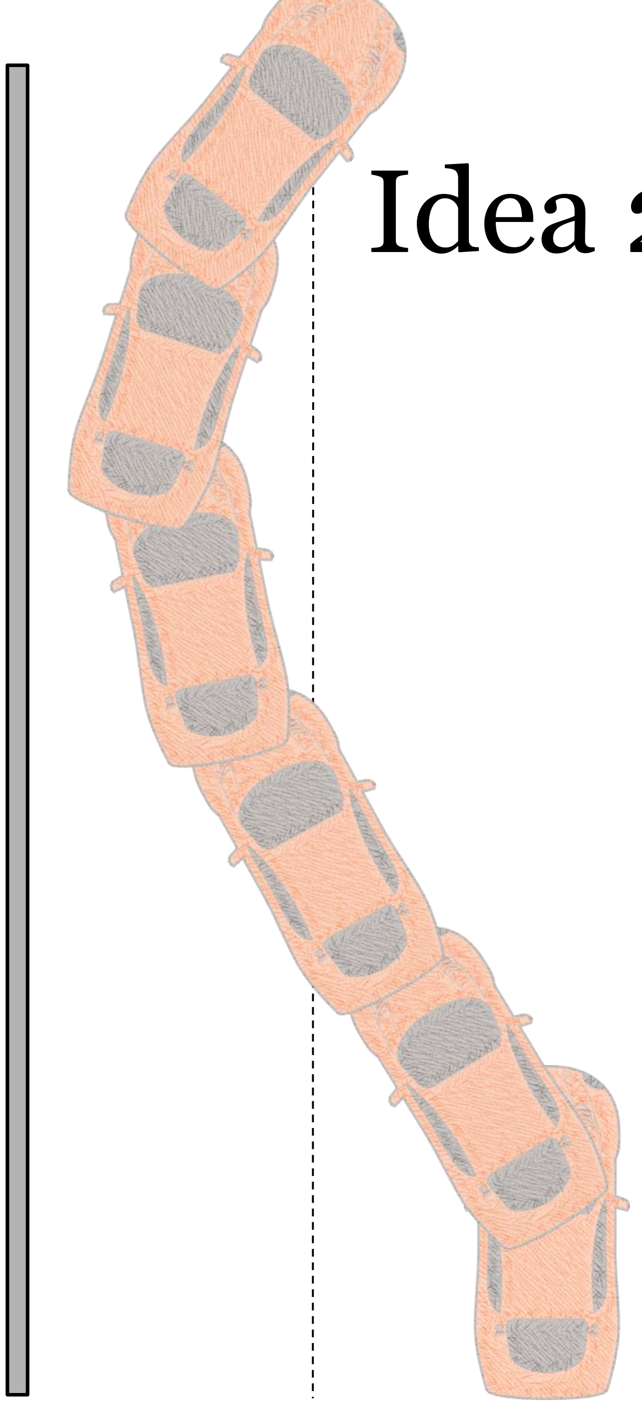
Idea 2: proportional (P-)control

$$\omega = K_p e(t)$$

Will the car reach the target line?

Will the car overshoot the target line?

Is the asymptotic (steady-state) error zero?



Idea 2: proportional (P-)control

$$\omega = K_p e(t)$$

Will the car reach the target line? YES

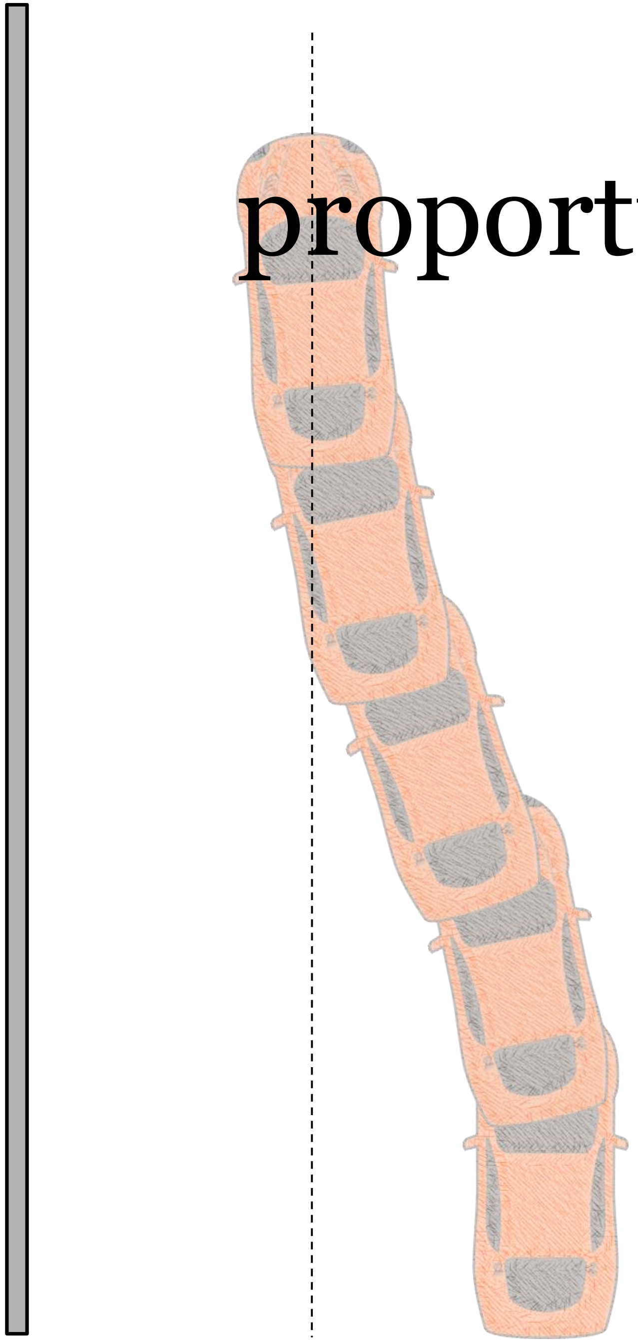
Will the car overshoot the target line? YES

Is the asymptotic (steady-state) error zero? NO

Addressing the oscillation problem

- Need to reduce turning rate **well before** the line is approached
- Idea: have a small proportional gain K_p
  Problem: that means the car doesn't turn very much
- Idea: need to **predict the error** in the near future
  This is good, as long as the error does not oscillate at a very high frequency

Idea 3:
proportional-derivative (PD-)control



$$\omega = K_p e(t) + K_d \dot{e}(t)$$

How do we set the gains?

What if there are systematic errors/biases?

What if the error estimate is very noisy?

Handling systematic biases: the integral term

- Examples of systematic biases:
  - wheels are misaligned
  - car is much heavier on one side

- Add integral of the error from the beginning of time: $\int_{\tau=0}^{\tau=t} e(\tau)d\tau$

- Can the PI-controller have nonzero error asymptotically (in steady-state)?
  - NO. In steady state both the control ω_∞ and the error e_∞ must be constant. If the asymptotic error is nonzero then the control is not constant: $\omega_\infty = K_p e_\infty + K_i e_\infty t$

Potential problem: integrator windup

- What happens if the control variable reaches the actuator's limits?
- I.e. the car can't turn as fast as the controller commands it.
- Actuator may remain at its limit for a long time while the controller modifies its commands
- Error increases, integral term winds up while controller goes back to issuing commands in the feasible region.

Potential problem: integrator windup

- Heuristic fixes:
  - Limit integral error values
  - Stop integral error while the commands are in the non feasible region
  - Reduce gain of integral error term

PID controller

$$\omega(t) = K_p e(t) + K_d \dot{e}(t) + K_i \int_{\tau=0}^{\tau=t} e(\tau) d\tau$$

Perhaps the most widely used controller in industry and robotics.
Perhaps the easiest to code.

You will also see it as:

$$\omega(t) = K_p [e(t) + T_d \dot{e}(t) + \frac{1}{T_i} \int_{\tau=0}^{\tau=t} e(\tau) d\tau]$$

Tips: how to implement PID

- Assume time is discrete
- Identify your error function e(t)= current_state(t)- target_state(t)
- Is the measurement of the state reliable?
- If the measurement of the current state is very noisy you might want to smooth/filter it, using:
    - **Moving average filter** with uniform weights

      $$\hat{x}_t = \frac{x_t + x_{t-1} + ... + x_{t-k+1}}{k} = \hat{x}_{t-1} + \frac{x_t - x_{t-k}}{k}$$

      Potential problem: the larger the window of the filter the slower it is going to register changes.
    - **Exponential filter**

      $$\hat{x}_t = \alpha \hat{x}_{t-1} + (1-\alpha) x_t, \quad \alpha \in [0, 1]$$

      Potential problem: the closer α is to 1 the slower it is going to register changes.

Tips: how to implement PID

- Approximate the integral of error by a sum

- Approximate the derivative of error by:
  - Finite differences $\dot{e}(t_k) \approx \frac{e(t_k) - e(t_{k-1})}{\delta t}$
  - Filtered finite differences, e.g. $\dot{e}(t_k) \approx \alpha \dot{e}(t_{k-1}) + (1-\alpha)\frac{e(t_k) - e(t_{k-1})}{\delta t}$

- Limit the computed controls
- Limit or stop the integral term when detecting large errors and windup

Tips: how to tune the PID

- Manually:
  - First, use only the proportional term. Set the other gains to zero.
  - When you see oscillations slowly add derivative term
    - Increasing K_d increases the duration in which linear error prediction is assumed to be valid
  - Add a small integral gain

Tips: how to tune the PID

- Ziegler-Nichols heuristic:
  - First, use only the proportional term. Set the other gains to zero.
  - When you see consistent oscillations, record the proportional gain K_u and the oscillation period T_u

Ziegler–Nichols method[1]			
Control Type	K_p	T_i	T_d
P	$0.5K_u$	-	-
PI	$0.45K_u$	$T_u/1.2$	-
PD	$0.8K_u$	-	$T_u/8$
classic PID[2]	$0.6K_u$	$T_u/2$	$T_u/8$

Tips: how to tune the PID

- After manual or Z-N tweaking you might want to use coordinate ascent to search for a better set of parameters automatically:

See Sebastian Thrun's online class "AI for robotics" on Udacity for more details on this. He calls the algorithm Twiddle and it is in Lesson 5.

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

threshold = 0.001

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

When is PID insufficient?

- Systems with large time delays
- Controllers that require completion time guarantees
  - E.g. the system must reach target state within 2 secs
- Systems with high-frequency oscillations
- High-frequency variations on the target state

Example applications: self-driving cars

Cascading PID

- Sometimes we have multiple error sources (e.g. multiple sensors) and one actuator to control.

- We can use a master PID loop that sets the setpoint for the slave PID loop. Master (outer loop) runs at low rate, while slave (inner loop) runs at higher rate.

- One way of getting hierarchical control behavior.