BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

7 – Line Following & PID

Robotics and Computer Vision (BPC-PRP)

Course supervisor: Ing. Adam Ligocki, Ph.D.

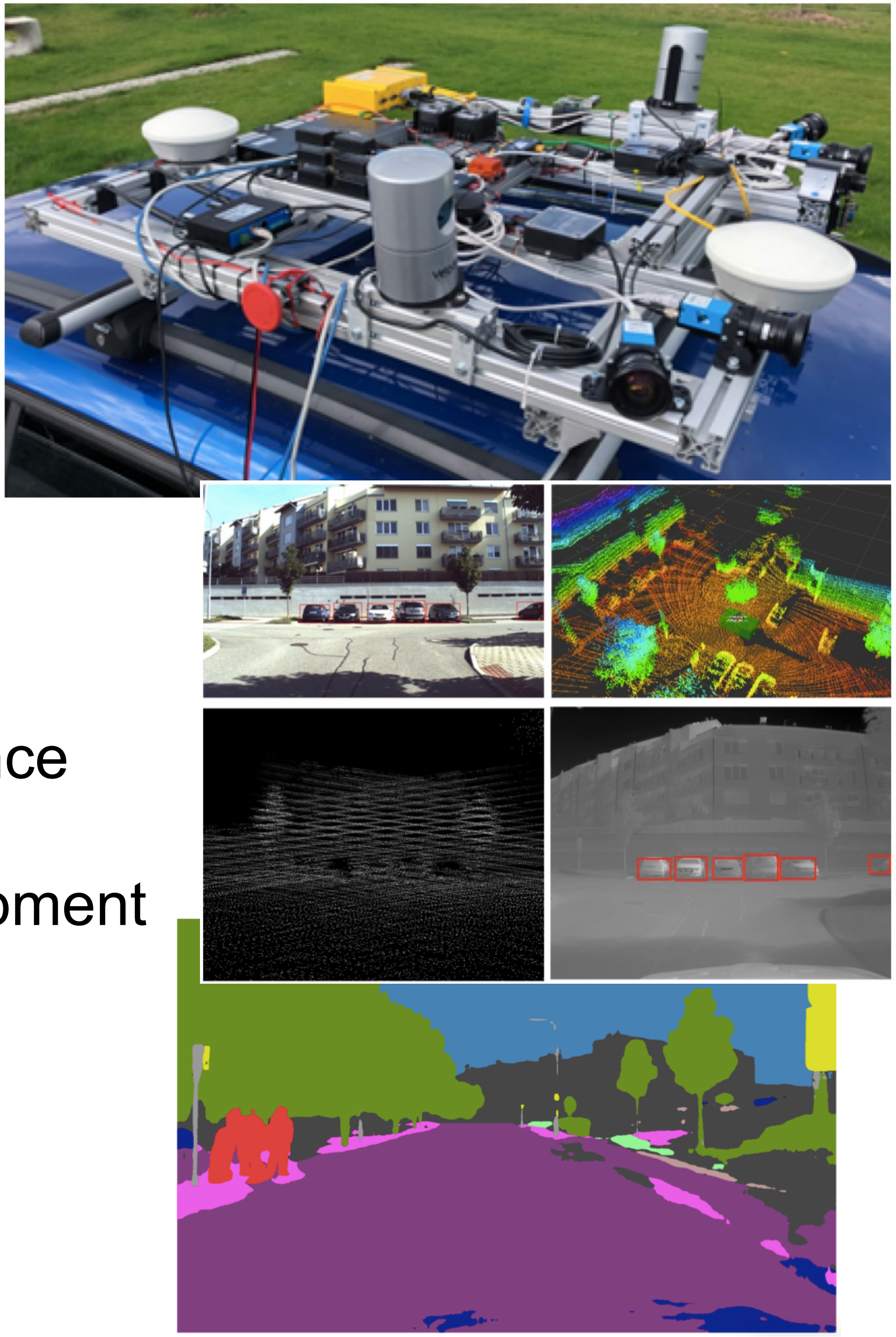
Ing. Adam Ligocki, Ph.D.
Ing. Tomáš Jílekm Ph.D.

Brno University of Technology
2025

Robotics and AI

Profile

Robotics and AI



Ing. Adam Ligocki, Ph.D.

Position: Assistant Professor

Research: Data Fusion

Room: SE1.102

Web: https://www.vut.cz/lide/adam-ligocki-154791

Background:

- Artificial Intelligence
- Neural Networks
- Software Development

Motivation

Line following is a **foundational robotics task** that combines sensing, control, and actuation in a closed feedback loop.

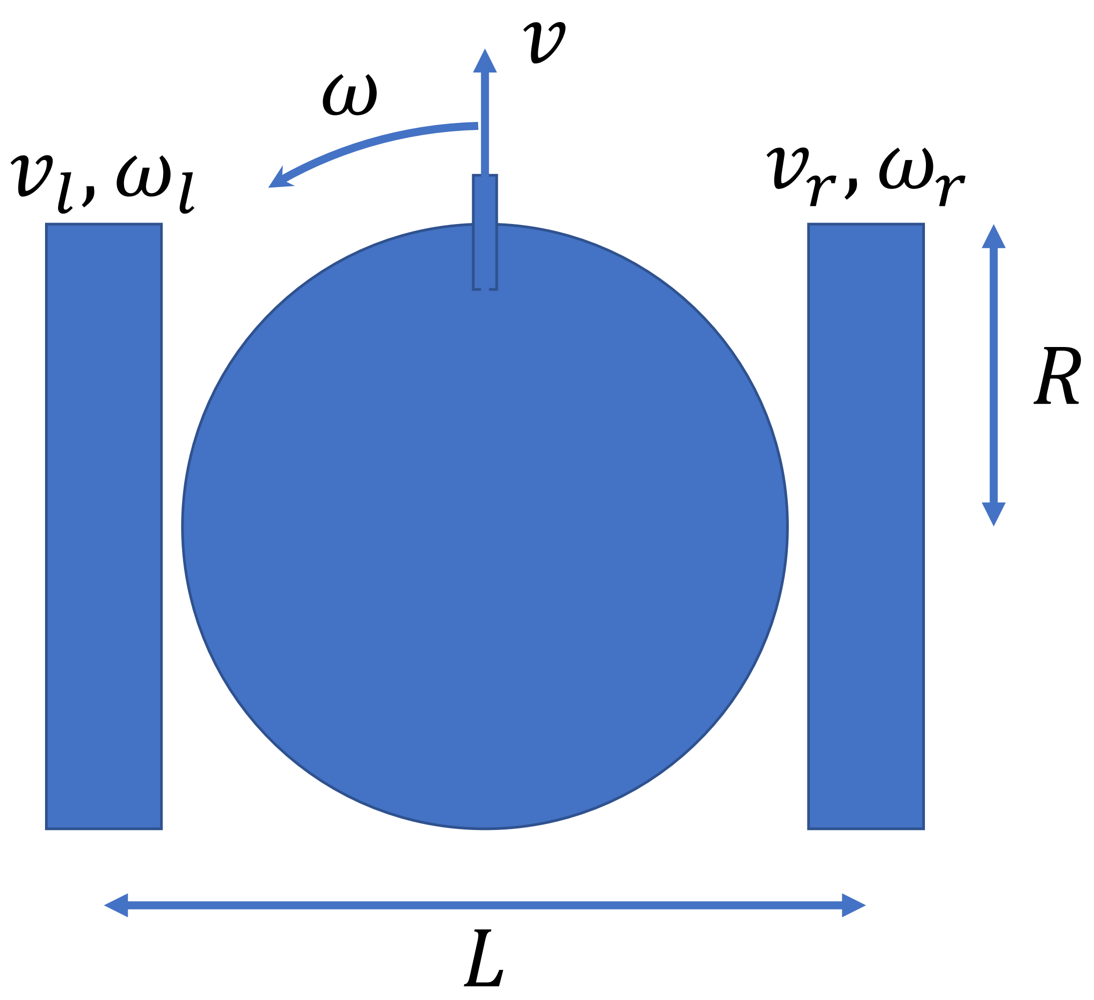
Technical Skills Developed

- **Sensor Integration**: Use of IR or vision-based line sensors for environment perception
- **Feedback Control**: Implementation and tuning of PID/PSD controllers
- **Kinematics**: Application of differential drive models and wheel speed control
- **Real-Time Processing**: Executing control loops within timing and computation constraints
- **Signal Filtering & Noise Handling**: Essential for stable motion in real-world conditions

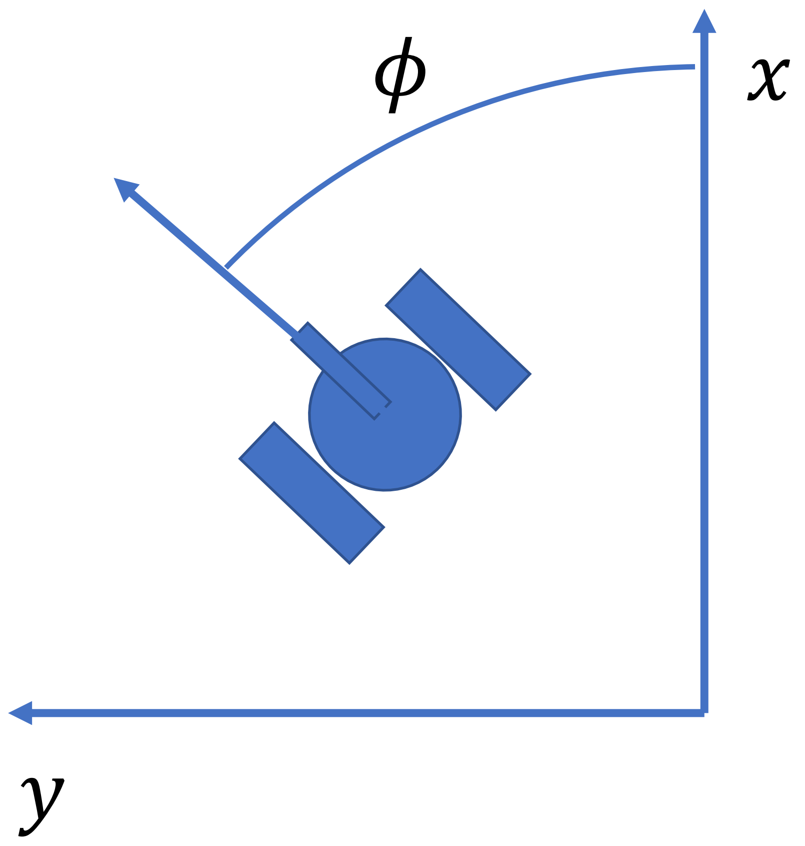
Real-World Relevance

- Used in **warehouse robots**, **automated guided vehicles (AGVs)**, and **production lines**
- Reflects principles used in **autonomous driving**, **lane keeping**, and **low-level navigation**
- Helps understand how **differential drive** systems are controlled in commercial platforms

Differential Chassis Review



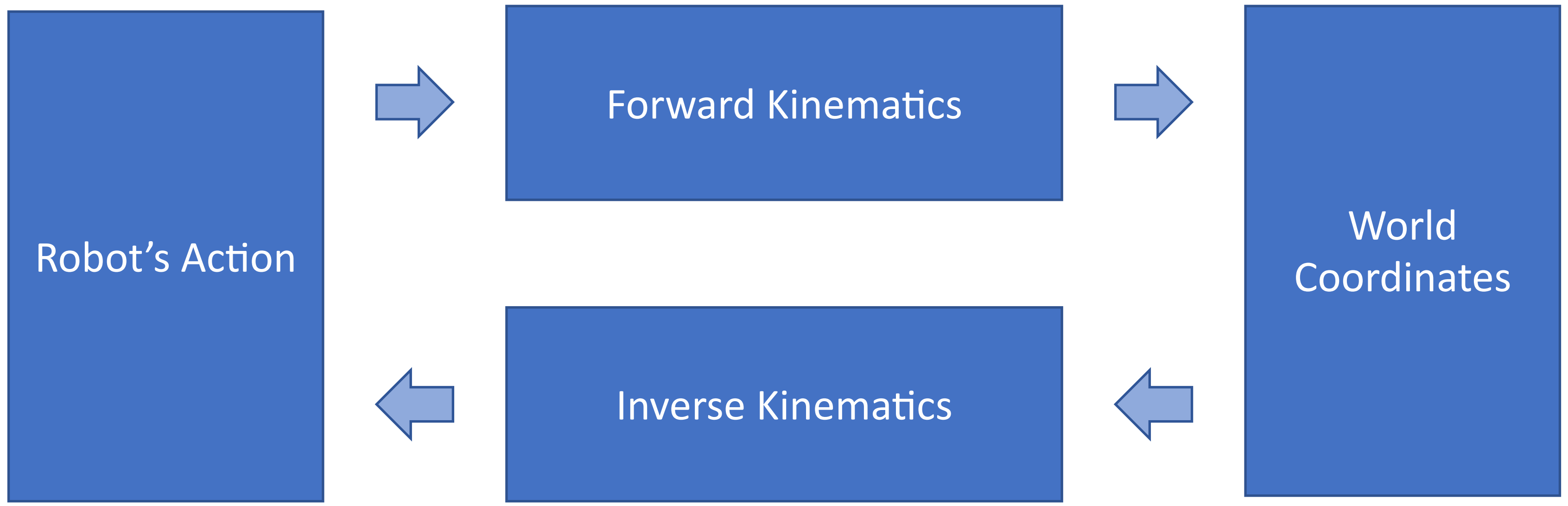
$$\omega_x * \mathrm{R} = v_x$$



Forward and Backward Kinematics

Forward kinematics: how the robot's joints (wheels) movement effects it's world coordinates.

Inverse kinematics: how to move robot's joints (wheels) to reach required world's coordinates.



Line Following Task

General description

- Develop a control algorithm that uses two line sensors (analog or digital) to adjust the left/right wheel speeds, keeping the robot centered on a visible track (typically black on white or vice versa).

Discrete approach (Bang Bang Control)

- Sensors output binary values (line / no line).
- Predefined actions (e.g., turn left, turn right, go straight).
- Simple logic, easy to implement.
- Jerky motion, poor performance at higher speeds or sharp turns.

Continuous Approach (PID Control)

- Sensors provide analog values indicating line position.
- Calculate error from center and apply PID (Proportional, Integral, Derivative) control.
- Smooth, responsive control, handles curves and speed changes.
- Requires tuning of PID parameters.

Line Following - Bang Bang Control

Bang-Bang regulation is a form of discrete-time, non-linear control where actuator outputs are driven by threshold-based sensor decisions rather than continuous feedback.
In the context of line following, control actions are determined by binary sensor outputs, leading to a piecewise control policy.

Characteristics

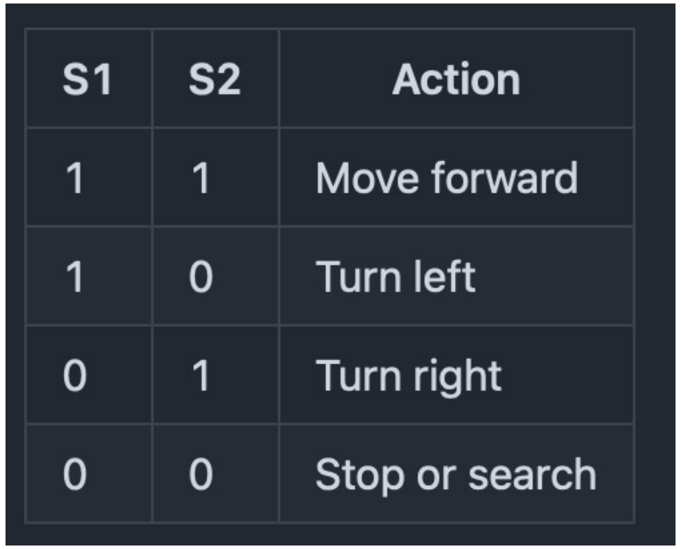
- Non-linear, discontinuous control output
- Zero hysteresis: small disturbances can cause frequent switching
- No error minimization: operates without measuring deviation from line center

Advantages

- Minimal computational complexity
- Fast response time
- Suitable for constrained systems with no floating-point support

Limitations

- Sensitive to sensor noise and misalignment
- Oscillatory behavior near edges of the line
- Inability to represent or correct partial deviation

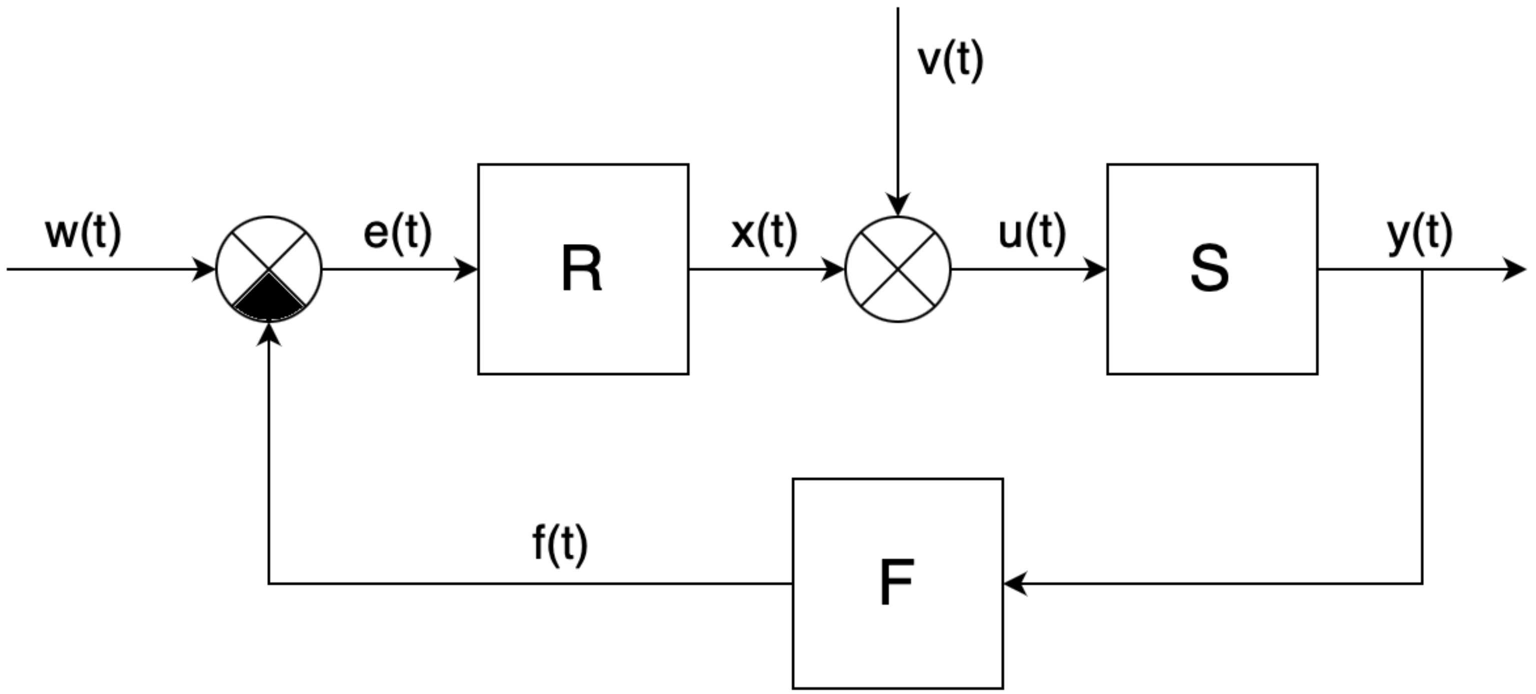


S1	S2	Action
1	1	Move forward
1	0	Turn left
0	1	Turn right
0	0	Stop or search

Line Following – PID Control Loop

Proportional-Integral-Derivative (PID) control is a fundamental method in feedback control systems, aiming to minimize the error between a desired state (centered on the line) and the measured state (sensor output).

In line following, the PID controller uses analog sensor data to continuously adjust motor speeds, resulting in smooth and stable trajectory tracking.



Line Following – PID Control Loop

The control signal $x(t)$ is computed as:

$$x(t) = K_P * e(t) + K_I * \int e(t)dt + K_D * \frac{d}{dt} e(t)$$

Characteristics

- Continuous control enables high-resolution path correction
- Handles curved paths and high-speed tracking effectively
- Requires careful gain tuning to balance responsiveness and stability

Advantages

- Precise error correction
- Smooth motor control with minimal oscillation
- Adaptive to varying track geometries

Limitations

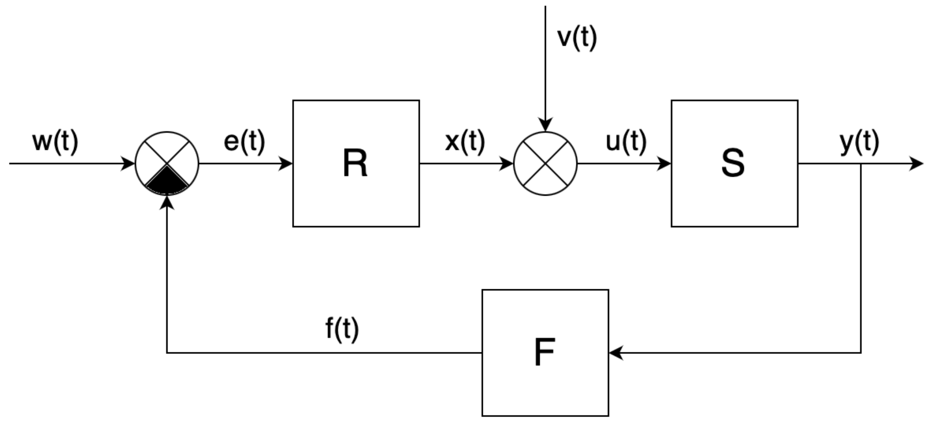
- Requires analog or high-resolution digital sensors
- Sensitive to noise in derivative term
- Tuning parameters can be time-consuming and environment-specific

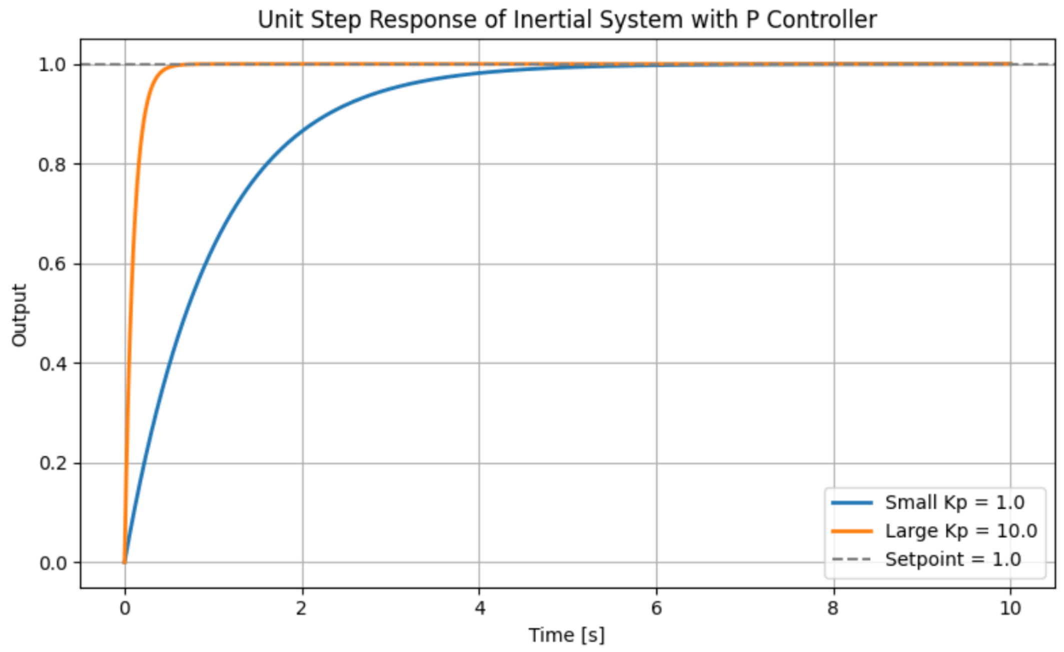
PID - Proportional Component

The **Proportional (P) controller** is the simplest form of feedback control. It generates a control signal proportional to the **instantaneous error** between the desired setpoint and the measured output.

$$x(t) = K_P * e(t)$$

- Reduces error proportionally to its size
- Faster response with higher gain
- May cause overshoot or instability if K_P is too high
- Cannot eliminate steady-state error in most systems
  - 0 type system (no integration in system)
  - External **disturbance**



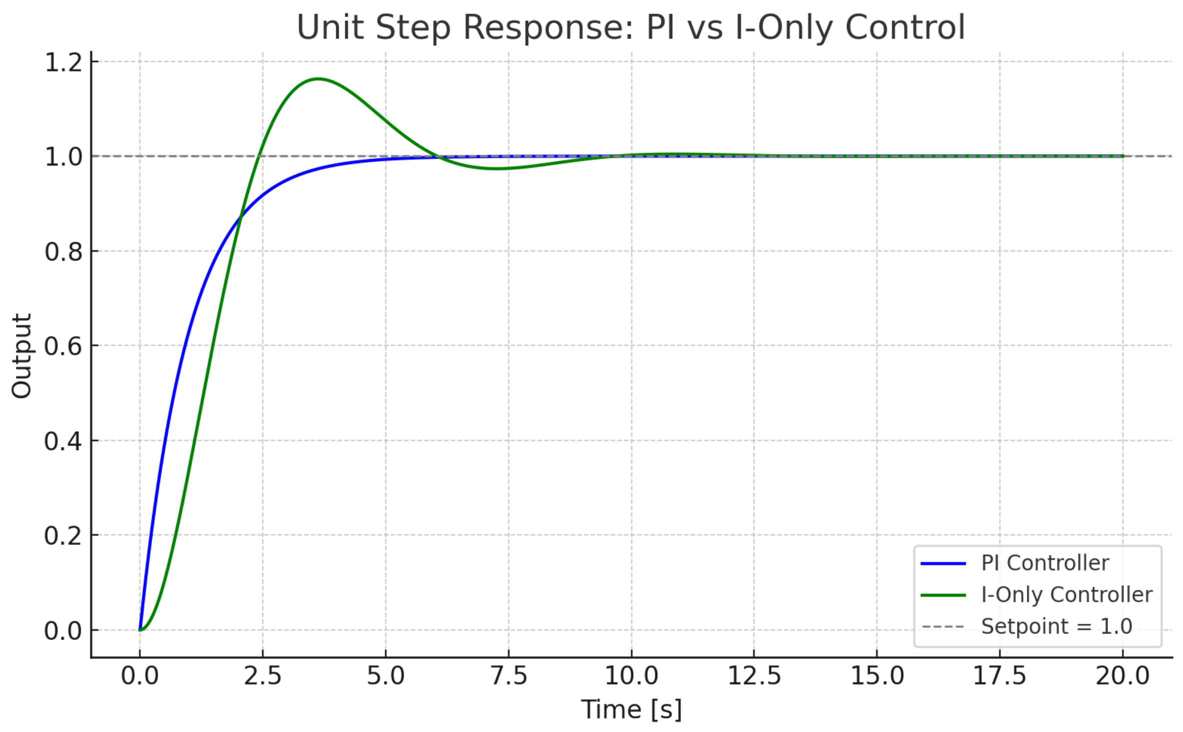


PID - Integral Component

The Integral controller generates a control action based on the accumulated error over time. Unlike proportional control, which reacts to current error, the integral term continues to act until the error is fully eliminated, making it especially effective in reaching the setpoint with zero steady-state error.

$$x(t) = K_I * \int e(t)dt$$

- Automatically corrects offsets and drift
- Responds slowly due to the accumulation process
- Can lead to overshoot and oscillations if not damped

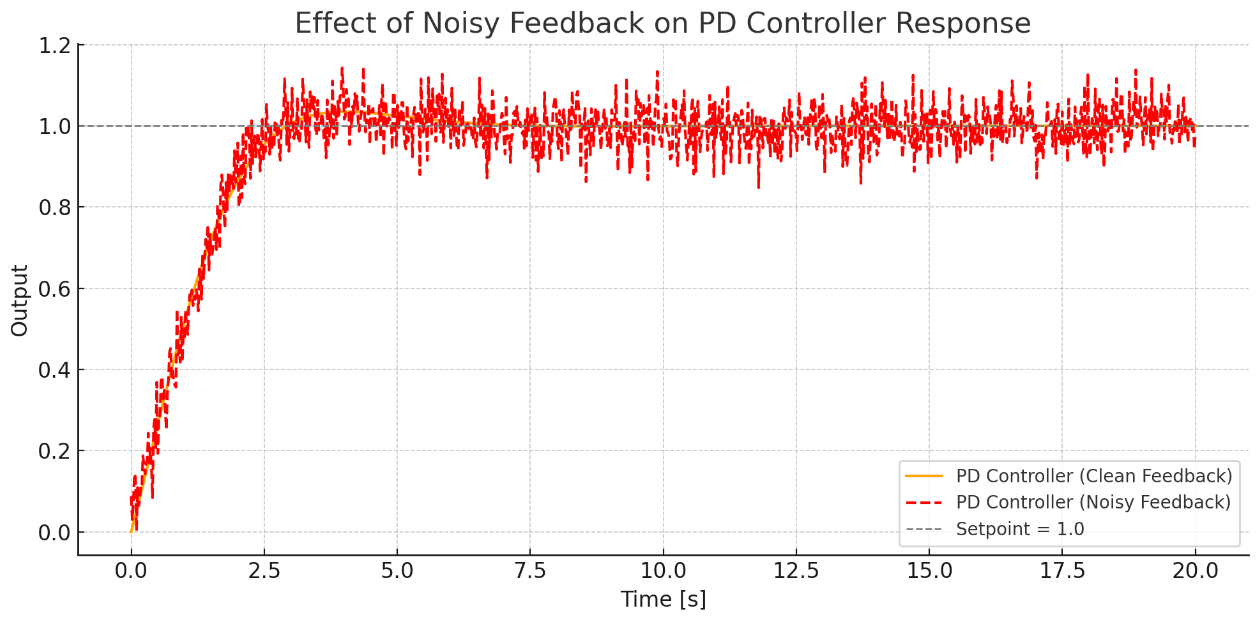


PID - Derivative Component

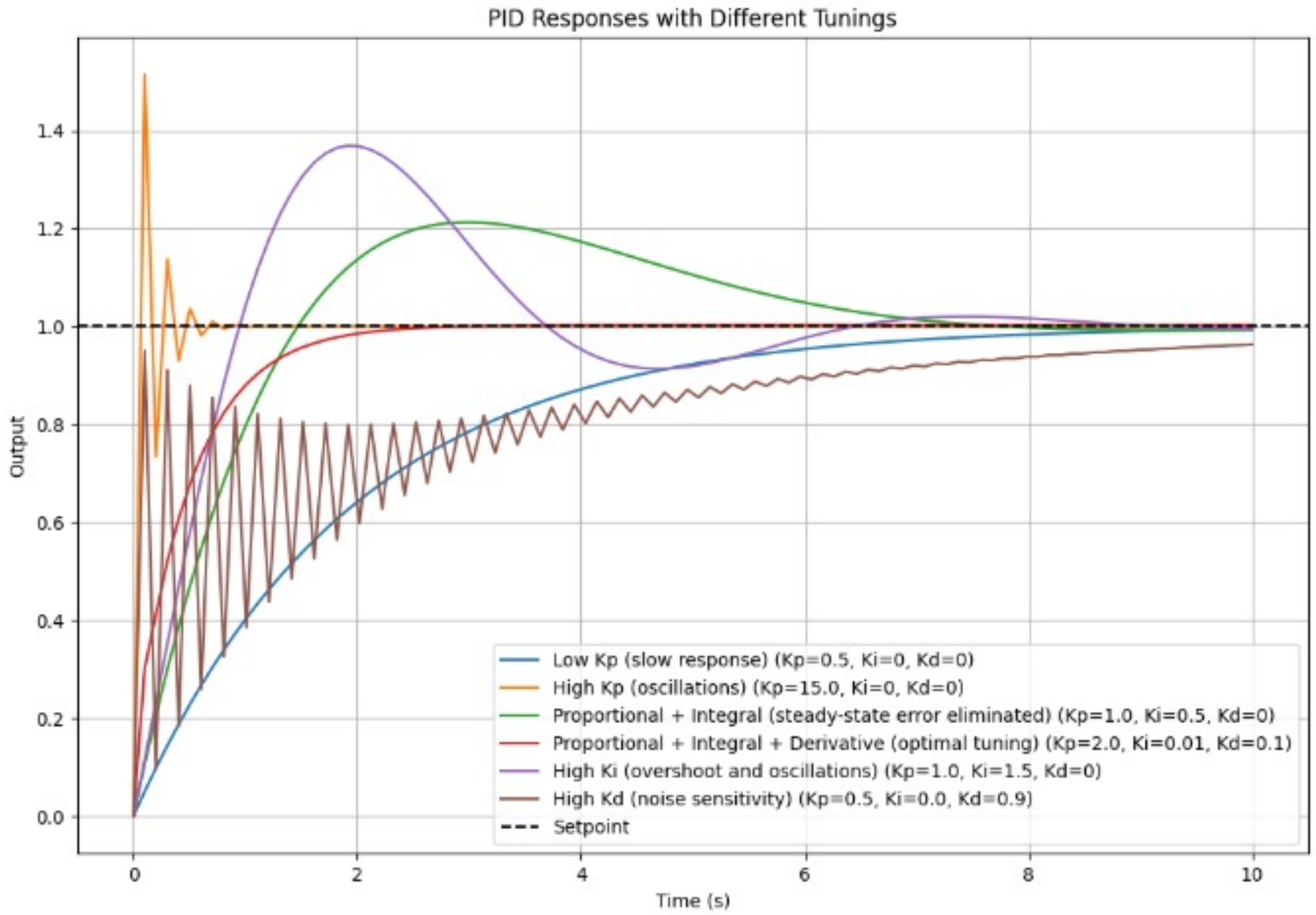
The Integral controller generates a control action based on the accumulated error over time. Unlike proportional control, which reacts to current error, the integral term continues to act until the error is fully eliminated, making it especially effective in reaching the setpoint with zero steady-state error.

$$x(t) = K_D * \frac{d}{dt} e(t)$$

- Improves transient response
- Reduces overshoot and settling time
- Provides phase lead which enhances stability
- Has no effect on steady-state error when used alone



PID

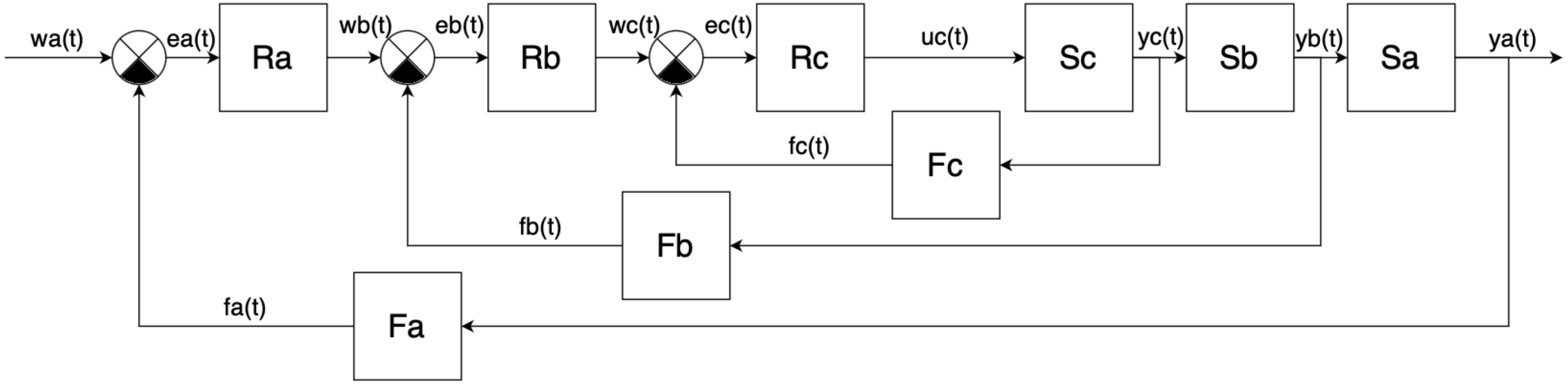


PID Responses with Different Tunings
Output
Time (s)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
2
4
6
8
10
Low Kp (slow response) (Kp=0.5, Ki=0, Kd=0)
High Kp (oscillations) (Kp=15.0, Ki=0, Kd=0)
Proportional + Integral (steady-state error eliminated) (Kp=1.0, Ki=0.5, Kd=0)
Proportional + Integral + Derivative (optimal tuning) (Kp=2.0, Ki=0.01, Kd=0.1)
High Ki (overshoot and oscillations) (Kp=1.0, Ki=1.5, Kd=0)
High Kd (noise sensitivity) (Kp=0.5, Ki=0.0, Kd=0.9)
Setpoint

Multiloop Control System

Multiloop (or nested-loop) control refers to a **hierarchical architecture** where multiple feedback loops are organized by control objective and response time. Each loop regulates a specific variable, using the output of outer loops as its reference input.

This approach is widely used in **electromechanical systems**, **process control**, and **robotics** where precision, stability, and responsiveness are required.



Multiloop Control System

Outer Loop – High-Level Control

- Regulates slow-changing variables (e.g., position, temperature, orientation)
- Provides setpoints to the intermediate loop
- Typically updated at a lower frequency

Middle Loop – Intermediate Dynamics

- Controls variables like speed, flow rate, or velocity
- Translates high-level objectives into physical quantities
- Balances accuracy with responsiveness

Inner Loop – Fast Dynamics

- Regulates fast-changing variables such as current, pressure, or force
- Operates at high frequency for rapid correction
- Shields outer loops from high-frequency disturbances

Typical Application: Motor Control System

- Position Loop - controls robot arm position
- Speed Loop - maintains target velocity
- Current Loop - regulates motor torque (via current control)

PSD (Proportion, Sum, Difference)- Discrete PID

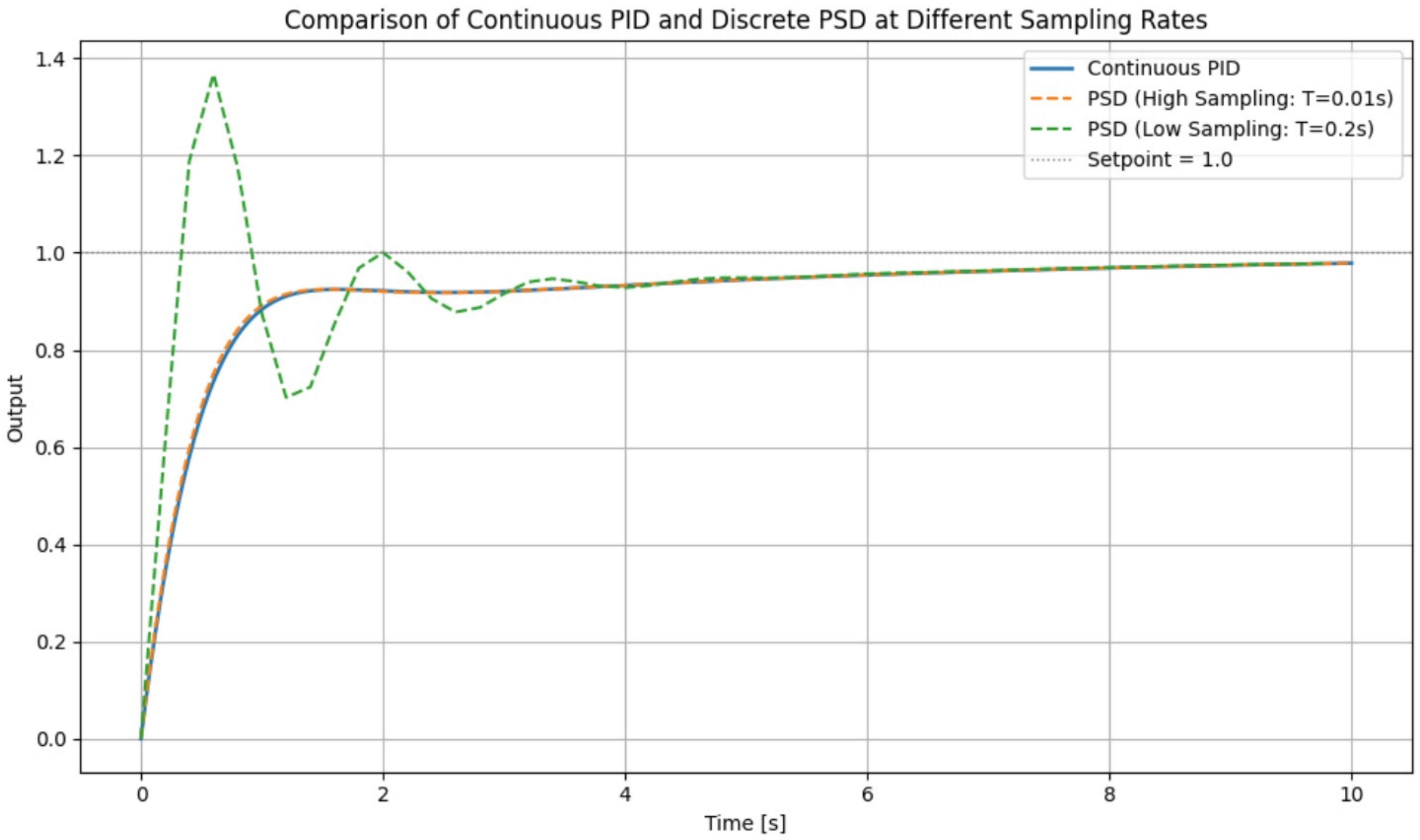
In digital systems, continuous-time PID control must be discretized for implementation on microcontrollers, PLCs, or DSPs.

The resulting PSD controller approximates the Proportional (P), Integral (S = Summation), and Derivative (D = Difference) components using sampled data, enabling real-time, software-based control.

$$x[k] = \ K_P * e[k] + K_I * T_s \sum_{i=0}^{k} e[i] + \ K_D * \frac{e[k] - e[k-1]}{T_s}$$

- **Sampling rate** must be high enough to capture system dynamics
- **Numerical integration** (e.g., trapezoidal or rectangular) for the I term
- **Noise filtering** required for D term (e.g., low-pass filter)
- **Anti-windup mechanisms** to prevent integrator overflow

PSD (Proportion, Sum, Difference)- Discrete PID

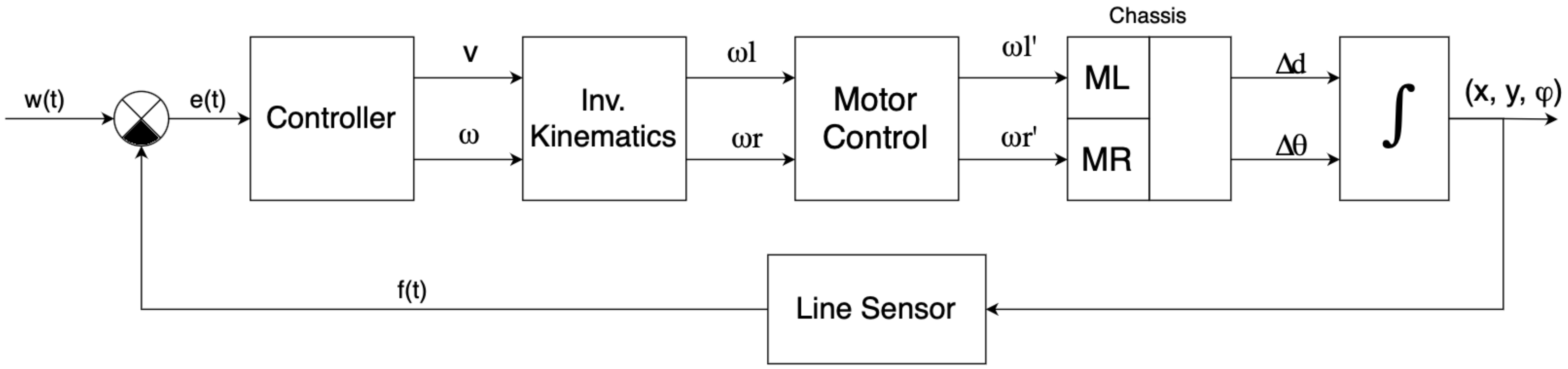


Differential Chassis Control

Reference Trajectory w(t) : Desired path or orientation (e.g., line center).
Line Sensor: Captures environmental feedback $\boldsymbol{f}(\boldsymbol{t})$, typically deviation from the line.
Error Calculation e$(t) = \boldsymbol{w}(\boldsymbol{t}) - \boldsymbol{f}(\boldsymbol{t})$: Determines the control objective deviation.
Controller: Computes desired linear velocity $\boldsymbol{v}$ and angular velocity $\boldsymbol{\omega}$ using PID or similar logic.
Inverse Kinematics: Converts $\boldsymbol{v}$, $\boldsymbol{\omega}$ into left/right wheel velocities $\boldsymbol{\omega_L}, \boldsymbol{\omega_R}$.
Motor Control: Applies actuation commands $\boldsymbol{\omega_L}', \boldsymbol{\omega_R}'$ to motors ML and MR.
Chassis Dynamics: Updates the robot's motion:

- Δd: linear displacement
- $\Delta\theta$: angular displacement

Integrator Block: Accumulates motion to estimate global pose (x, y, ϕ).



Different Controll Loop Approaches

Bang-Bang (Discrete State) Control

- Simple logic using digital sensors (e.g., ”left sensor detects" -> turn left).
- Very easy to implement, no math.
- Jerky motion, poor on sharp turns or noise.

Constant Forward Speed, Regulated Angular Speed

- Linear velocity v is fixed, only angular velocity ω is adjusted based on line-pose error.
- Simple, stable, predictable speed.
- May overshoot on curves or fail at high speed.

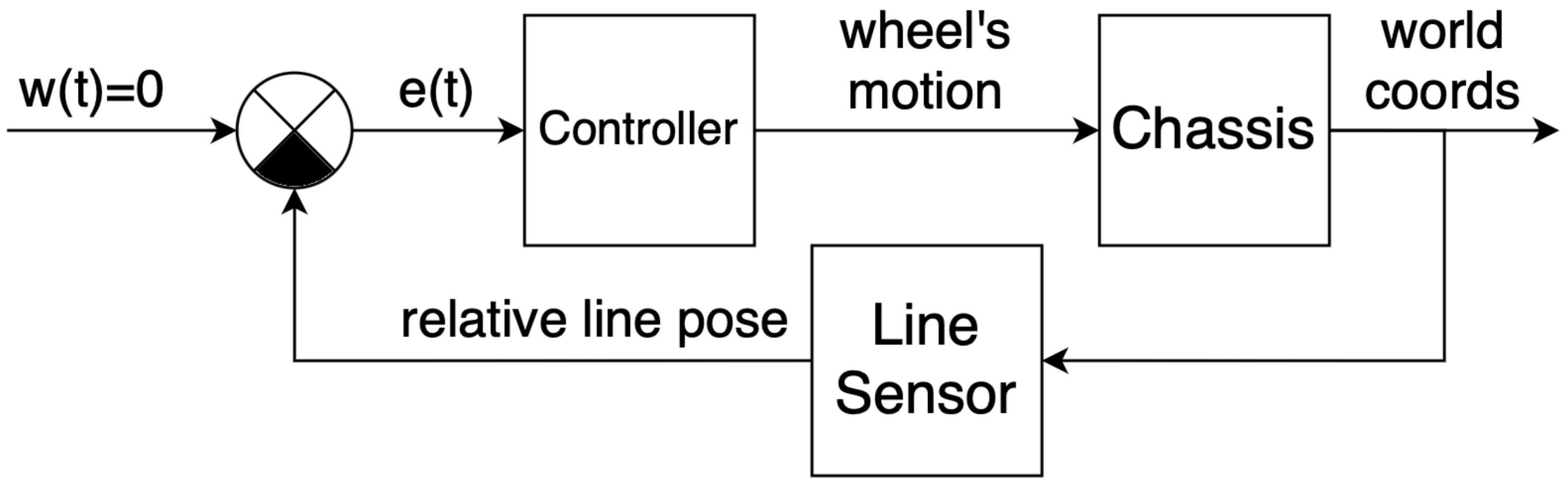
Controlled Forward and Angular Speed

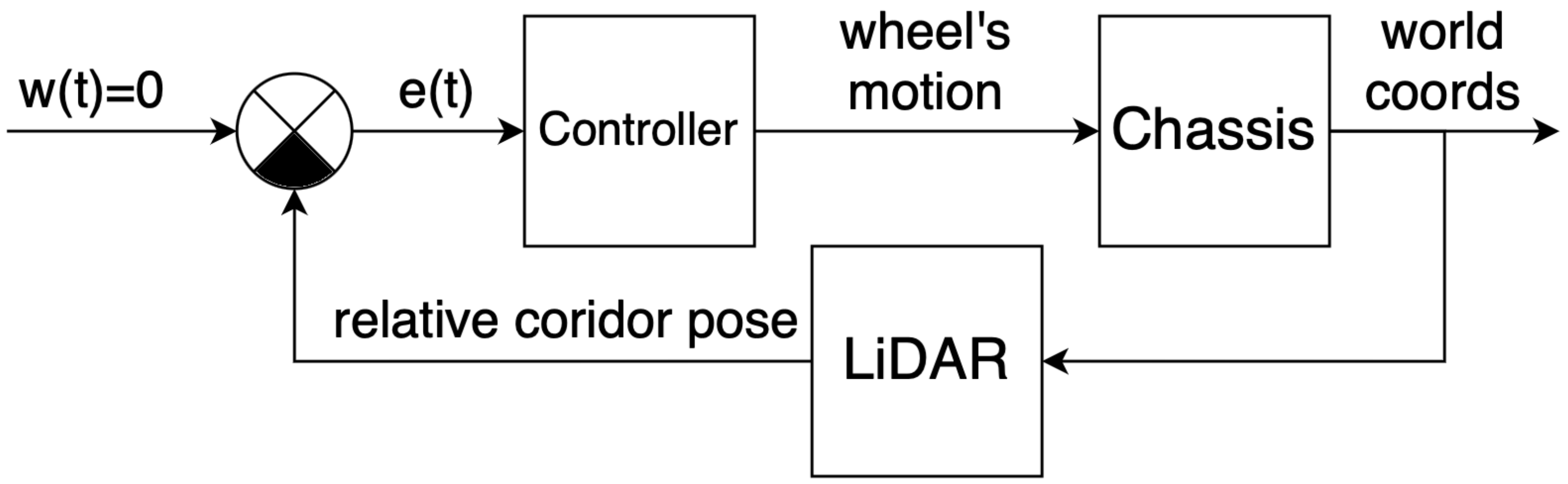
- Both v and ω are dynamically adjusted based on error and/or curvature of the line.
- Better handling of curves, adaptable speed.
- Requires more tuning and sensor feedback.

Model-Based Control (Trajectory Tracking)

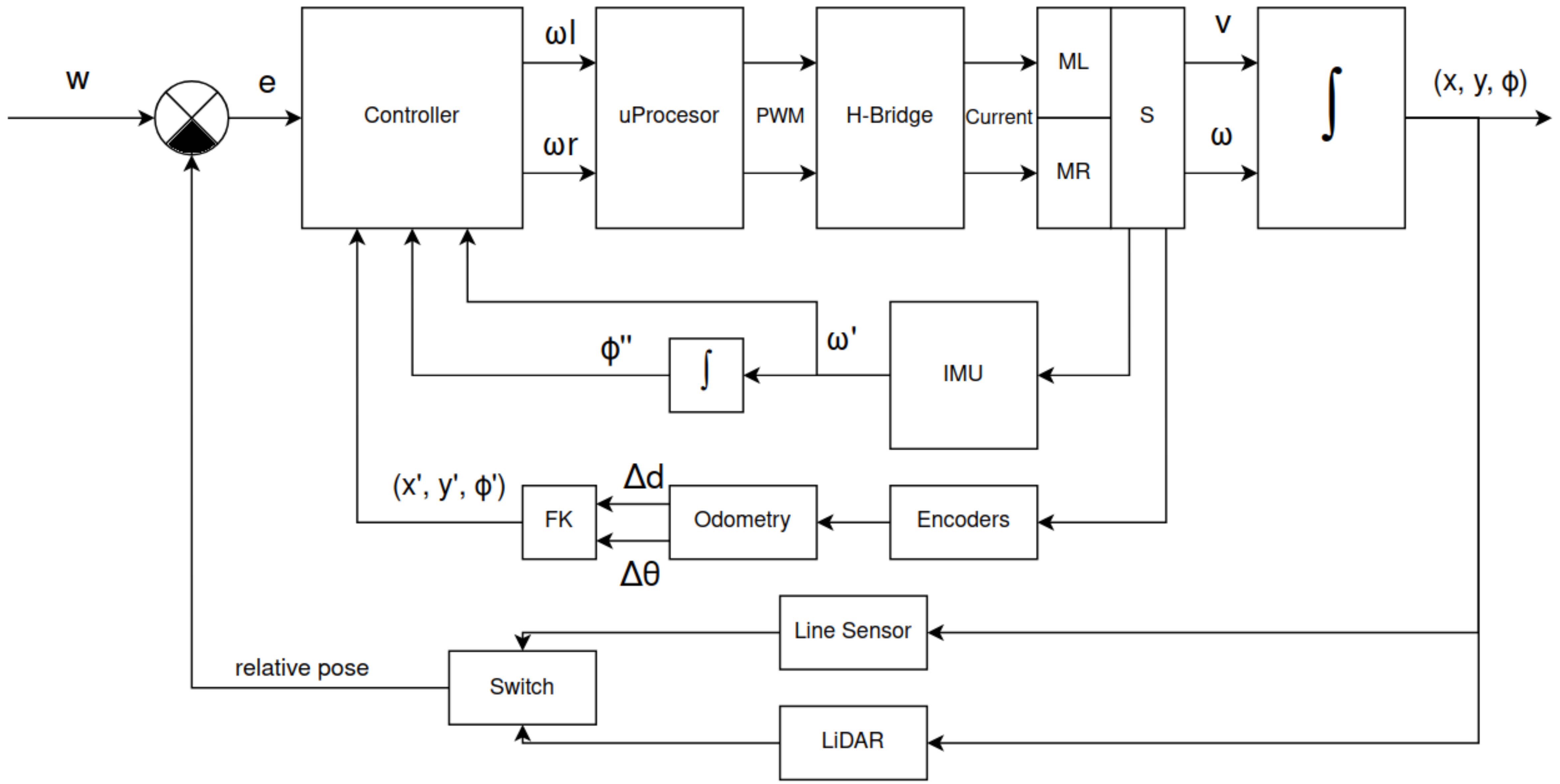
- Combine line tracking with **trajectory planning**, pose estimation, or kinematic models.
- Suitable for complex environments.
- Requires global localization and more computation.

Line Following vs Corridor Following



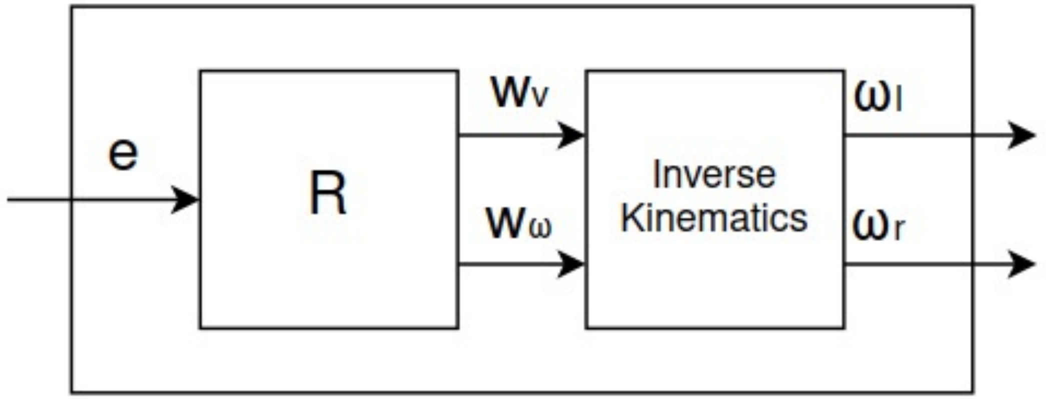


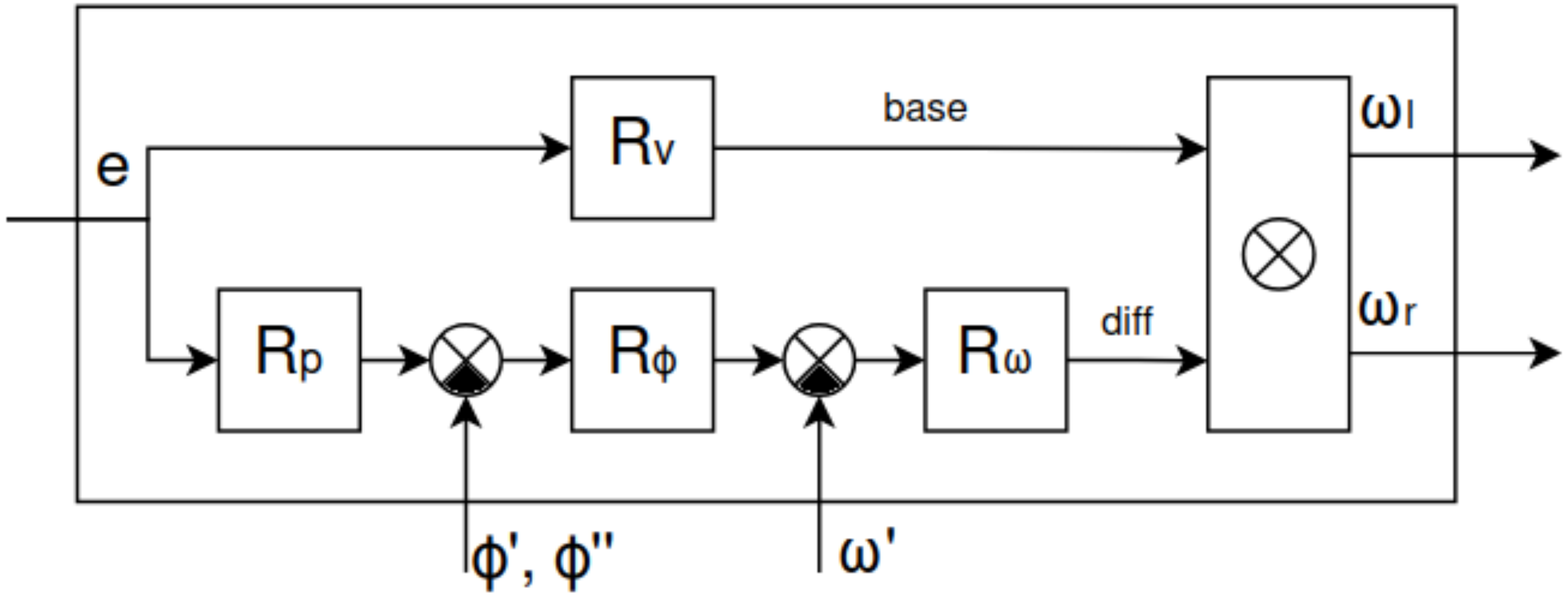
Big Picture


w
e
Controller
ωl
ωr
uProcesor
PWM
H-Bridge
Current
ML
MR
S
v
ω
∫
(x, y, ϕ)
ϕ''
∫
ω'
IMU
(x', y', ϕ')
FK
Δd
Δθ
Odometry
Encoders
Line Sensor
relative pose
Switch
LiDAR

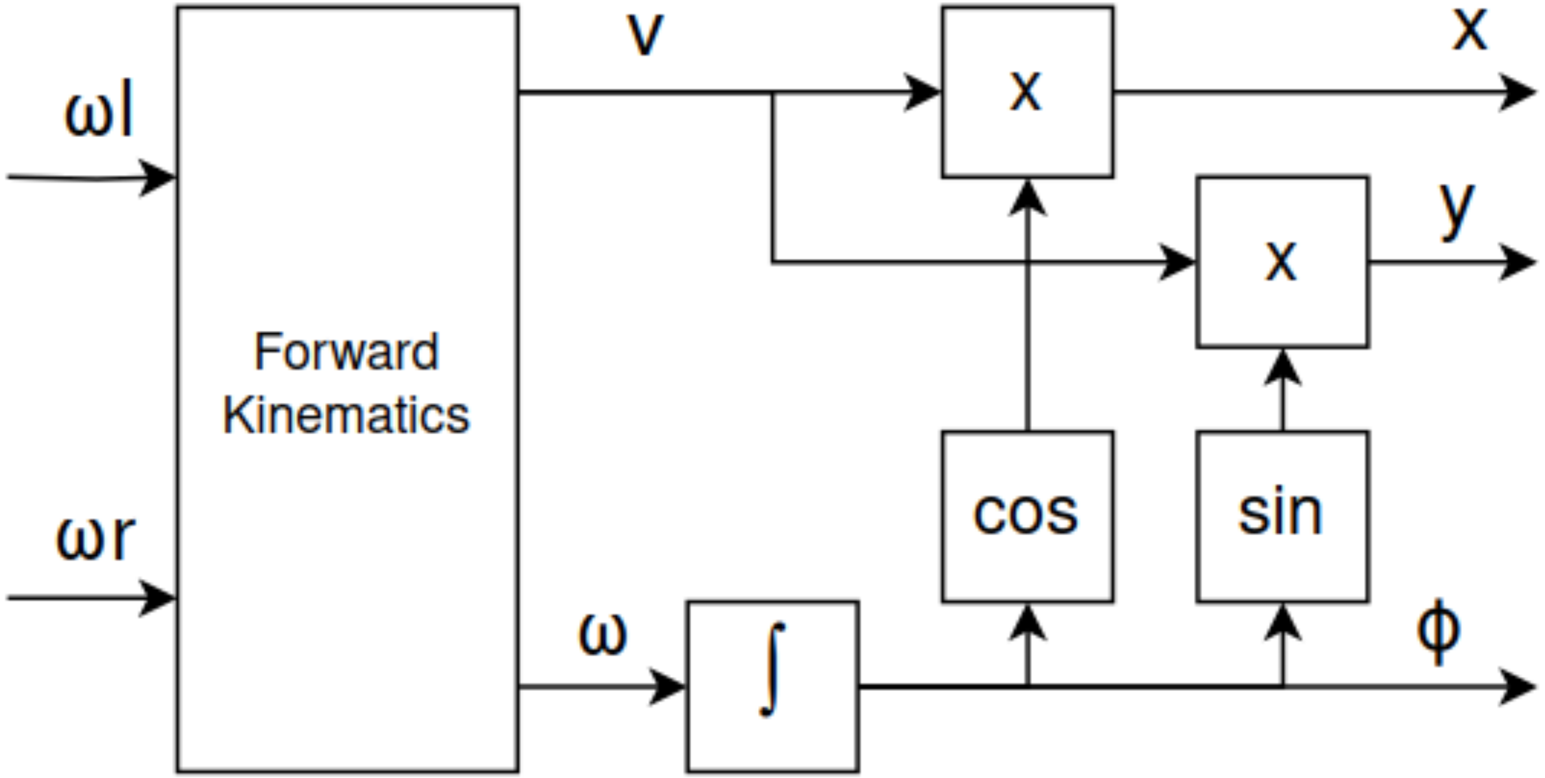
Big Picture – Controller Implementation

Possible Controller Implementation





Robot Model



Handling Noisy Sensor Data

In real-world systems, sensor feedback is often corrupted by **electrical noise**, **quantization error**, or **mechanical vibrations**.
This noise can destabilize control loops — especially when using **derivative action**, which amplifies high-frequency variations.

Derivative term magnifies noise -> jittery or erratic control

Low-Pass Filtering

- Apply a digital filter to smooth sensor readings
- Common: **Moving Average**, **Exponential Smoothing**, or **Butterworth filters**
- Place filter before D term to reduce amplification of high-frequency noise

Sensor Fusion & Averaging

- Combine multiple readings or sensor types to reduce random error
- Median filtering removes outliers effectively

Sample Rate Optimization

- Tune your controller's **sampling frequency** to balance responsiveness and noise sensitivity

Quantization & Deadband

- Implement a **deadband** to ignore small fluctuations
- Useful when dealing with digital sensors or low-resolution ADCs

Physical Contrains & Practical Tips

Tune the K_P , K_I , and K_D gains to achieve a control response that is **fast**, **stable**, and **accurate** — without overshoot, oscillations, or long settling times.

Tunning Algorithm

Start with P-only Control

- Set $K_I = 0,\ K_D = 0$
- Increase K_P until system responds quickly but starts to oscillate
- Back off slightly for stability

Add Integral (I) Term

- Slowly increase KIKI to eliminate **steady-state error**
- Watch for **overshoot** or **sustained oscillations**

Add Derivative (D) Term (Optional)

- Increase KDKD to reduce **overshoot** and **dampen oscillations**
- Be cautious: **D amplifies noise** — consider filtering the signal

Manual Tuning is effective for most systems; start with Ziegler–Nichols only for experimentation

- Tune for your goal: **speed**, **stability**, or **energy efficiency**
- Always filter noisy signals before applying D term
- Implement **anti-windup** to limit integrator output under saturation
- Check your **sampling rate**: too low = poor response, too high = computational noise

Other Resources

Understanding PID Control by **Matlab**
https://www.youtube.com/watch?v=wkfEZmsQqiA&list=PLn8PRpmsu08pQBgjxYFXSsODEF3Jqmm-y

DC motor PID speed control by **Curio Res**
https://www.youtube.com/watch?v=HRaZLCBFVDE

Low Pass Filter Design & Implementation by **Curio Res**
https://www.youtube.com/watch?v=HJ-C4Incgpw

PID Contorel - A Brief Introduction by **Brian Douglas**
https://www.youtube.com/watch?v=UR0hOmjaHp0&t=14s

PID Controller Implementation in Software by **Phil's Lab**
https://www.youtube.com/watch?v=zOByx3Izf5U&t=372s

Adam Ligocki

ligocki@vutbr.cz

https://www.linkedin.com/in/adamligocki/

https://github.com/adamek727

Robotics and AI Research Group

https://github.com/Robotics-BUT

Brno University of Technology
Faculty of Electrical Engineering and Communication
Department of Control and Instrumentation