Control Systems for Robotics

DASE7503 Robotic Systems Integration

Contents

- Introduction
- On-Off Control
- PID Control
- Velocity Control and Position Control
- Multiple Motors – Driving Straight

Introduction

- Robotic control systems are at the core of modern robotics.
- Enable robotics to perceive their environment, process information, and perform tasks.
- Incorporate logic analysis, data algorithms, and various processing methods to ensure precise and efficient movement.
- Be two common control methods, including **Open-loop control** and **Closed-loop control**.

Introduction

- **Open-loop control**
  - It works without feedback, meaning the system operates based solely on preset parameters.
  - This approach is common in simple applications, like automated conveyor belts or washing machines, where the tasks are repetitive and the environment is stable.
  - However, without feedback, any variation in conditions can lead to inaccuracies.

Introduction

- **Closed-Loop Control**:
  - also known as feedback control, are essential for tasks requiring high precision.
  - By continuously comparing the robot's output with the desired outcome, closed-loop control adjusts performance in real-time, correcting errors.
  - For example, in autonomous vehicles, sensors monitor speed, direction, and external conditions, allowing the system to make immediate adjustments for safety and efficiency.
- In real applications, a motor speed is affected by loading and friction.
- Supplying the same analog voltage (or equivalent: the same PWM signal) to a motor does not guarantee that the motor will run at the same under all circumstances.
- Use motor feedback via encoders for appropriate velocity control of a motor.

On-off Control

- Setting a certain PWM level alone will not help, since the motor's speed also depends on its load.
- Feedback control can solve this issue.
- A desired speed can be specified by the user or the application program, and the current actual speed measured by the shaft encoders.
- **Measurements** and **subsequent actions** can be taken very frequently, for example, from 100 times per second to 20,000 times per second.
- The action taken depends on the control model/strategy.
- The general principle of such model works as follows:
  - If actual speed is lower than desired speed:
    →Increase motor power.
  - If actual speed is higher than desired speed:
    →Decrease motor power.

On-off Control

- In the simplest case, power to the motor is either switched on (when the actual speed is too low) or switched off (when the actual speed is too high). This control law is represented by the formula below, with:

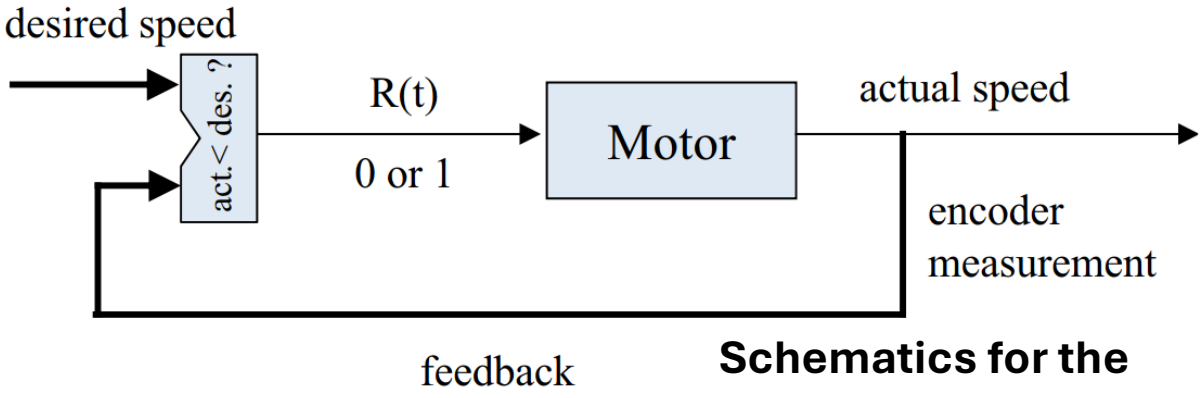
R(t)	motor output function over time t
$v_{act}(t)$	actual measured motor speed at time t
$v_{des}(t)$	desired motor speed at time t

$$R(t) = \begin{cases} 1 & \text{if } v_{act}(t) < v_{des}(t) \\ 0 & \text{else} \end{cases}$$

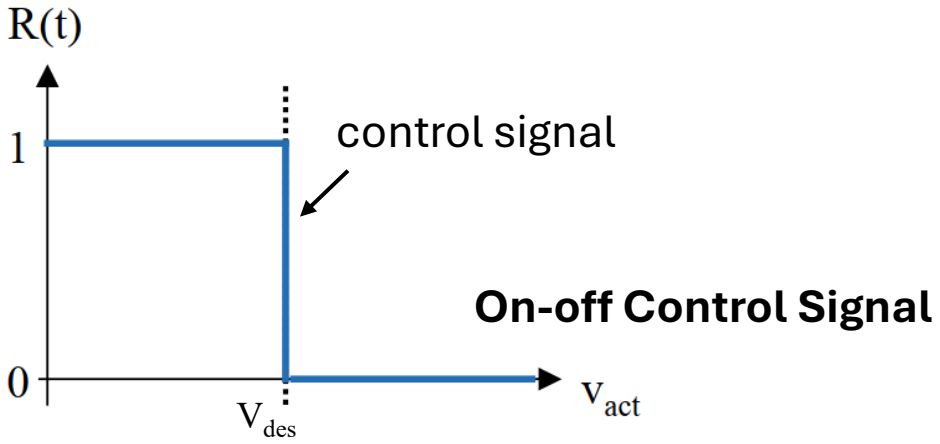
On-off Control

- On-off controller is a simple control strategy where a device or system switches directly between two states—typically "on" (active) and "off" (inactive)—based on certain threshold conditions.
- Known as piecewise constant controller or bang-bang controller.
- The motor output is set to one if the measured velocity is too low, otherwise, it is set to zero.
- This controller only works for a positive value of v_{des} .

$$R(t) = \begin{cases} 1 & \text{if } v_{act}(t) < v_{des}(t) \\ 0 & \text{else} \end{cases}$$



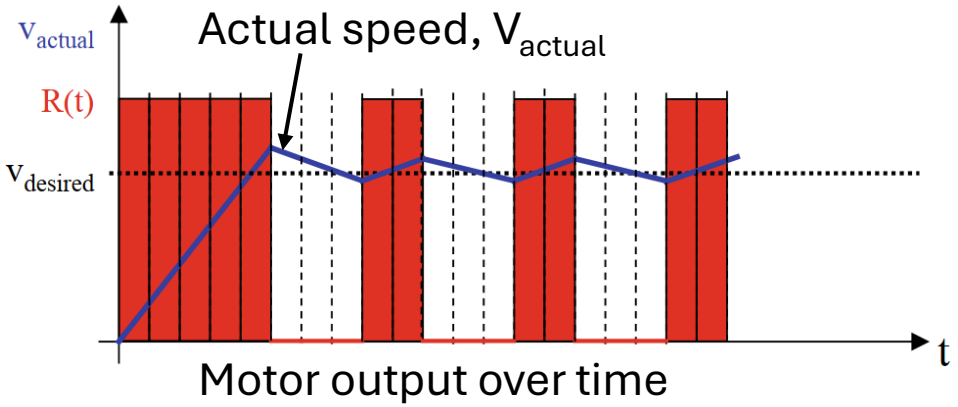
Schematics for the On-off Controller



On-off Control Signal

On-off Control

- The behavior over time of an on-off controller is shown in Figure.
- Assuming the motor is at rest in the beginning and the desired speed is **non- zero**, then the control signal for the motor will be 1.
- This will stay until at some stages the actual speed becomes larger than the desired speed. Then, the motor control signal is changed to 0.
- Over some time, the actual speed will slow down and once it falls below the desired speed, the control signal will again be set to 1.
- Note that the motor control signal is **not instantaneously updated**. Instead, it takes some time to read and process the sensor signal, so **output changes** occur at **fixed time intervals** (e.g., every 10 ms in Figure).
- This delay creates a small **overshooting** or **undershooting** of the actual motor speed and thereby introduces some **hysteresis**.

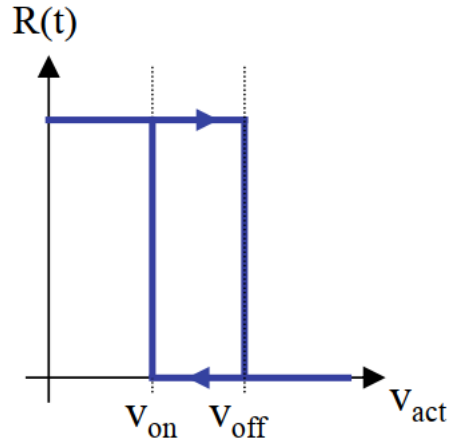


Motor output over time

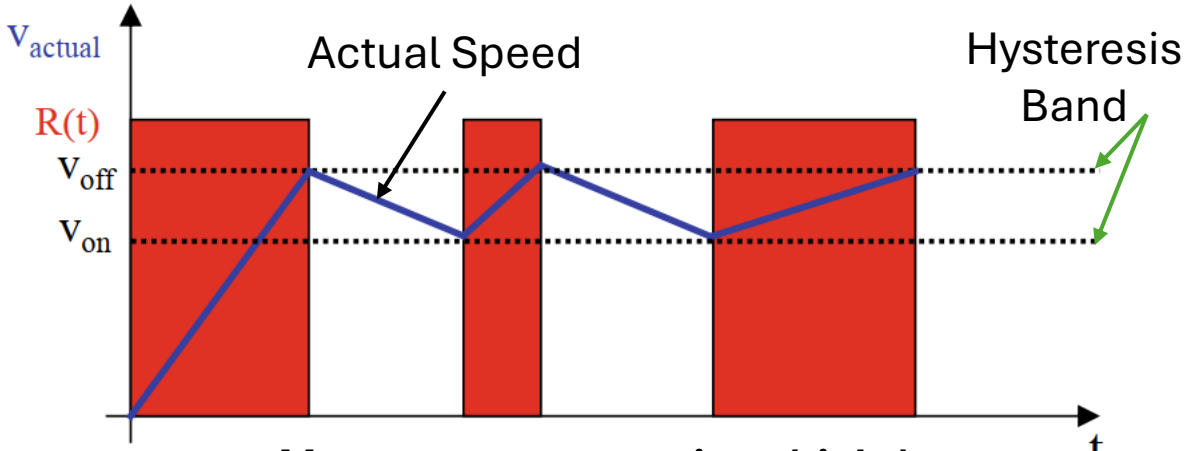
On-off Control

- The on-off controller is the simplest possible method of control.
- Many technical systems use it, not limited to controlling a motor, such as a refrigerator, heater, thermostat, light dimmer, etc.
- Some of these technical systems use a **hysteresis band**, which consists of **two desired values**, one for switching on and one for switching off.
- This **prevents a high switching frequency near the desired value**. The formula for an on-off controller with hysteresis is:

$$R(t+\Delta t) = \begin{cases} 1 & \text{if } v_{act}(t) < v_{on}(t) \\ 0 & \text{if } v_{act}(t) > v_{off}(t) \\ R(t) & \text{else} \end{cases}$$



On-off control signal with hysteresis band

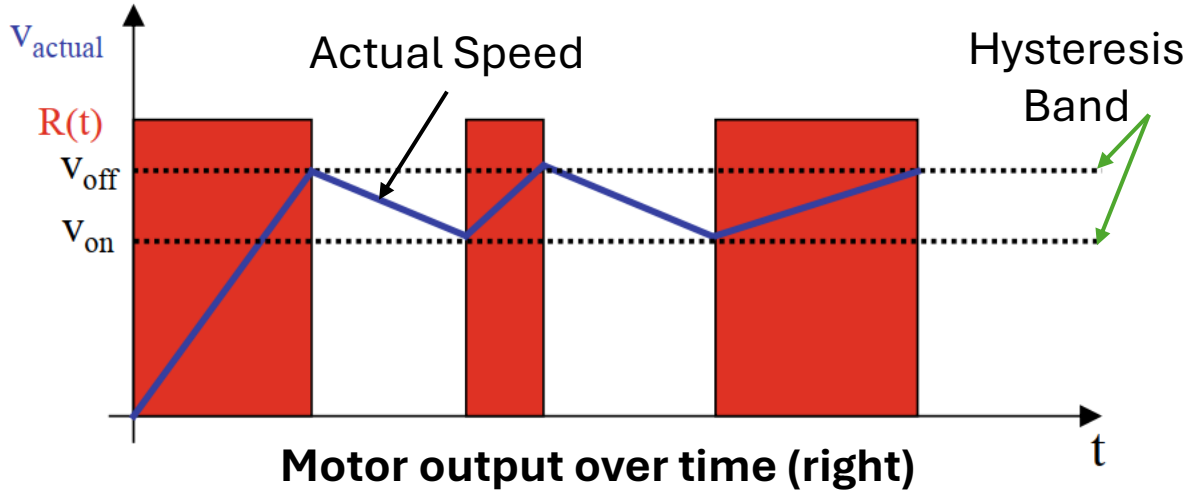


Motor output over time (right)

On-off Control

- Note that this definition is not a function in a mathematical sense, because **the new motor output for an actual speed between the two band values** is equal to the previous motor value.
- This means it can be equal to 1 or 0, depending on its history.

$$R(t+\Delta t) = \begin{cases} 1 & \text{if } v_{act}(t) < v_{on}(t) \\ 0 & \text{if } v_{act}(t) > v_{off}(t) \\ R(t) & \text{else} \end{cases}$$



On-off Control

- For putting this theory into practice and implementing an on-off controller in software, the following steps are processed:
  1. We need a subroutine for calculating the motor signal as defined by the formula.

     This subroutine has to:

     a. Read encoder data (input)

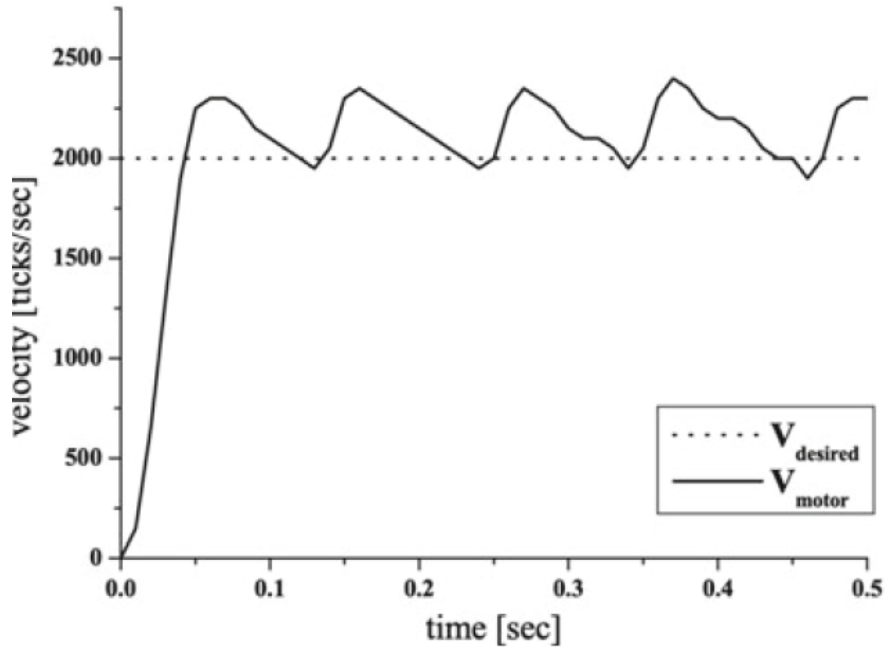
     b. Compute new output value R(t)

     c. Set motor speed (output)
  2. The subroutine has to be called periodically (for example, every 1/100 s).

     Since motor control is a low-level task, we would like this to run in the background and not interfere with any user program.

On-off Control

- **Limitations:**
  - **Chattering:** Rapid switching near the setpoint can cause wear and inefficiency.
  - **Lack of Precision:** It does not provide smooth or proportional control, which can lead to oscillations around the setpoint.
- In summary, on-off control is useful for **simple**, **low-precision** applications in robotics, especially when the system can tolerate **some oscillations** or when simplicity and cost are primary considerations.



Measured step response of an on-off controller

PID Control

- Compared with an On-off controller, a PID controller is a more advanced controller and an industry standard.
- It comprises a **proportional**, an **integral** and a **derivative** control part (**PID = P + I + D**).
- **Proportional (P):**
  - Reacts proportionally to the current error (difference between desired and actual value).
  - Provides immediate correction, reducing steady-state error but may cause overshoot.
- **Integral (I):**
  - Reacts based on the accumulation of past errors.
  - Eliminates steady-state error and improves long-term accuracy.
  - Can lead to slower response or oscillations if not tuned properly.
- **Derivative (D):**
  - Reacts to the rate of change of the error.
  - Provides predictive damping, reducing overshoot and oscillations.
  - Enhances system stability and response smoothness.
- PID control is a versatile and essential method in robotics for achieving precise, stable, and responsive control of various robotic subsystems.

PID Control – Proportional Controller

- For many control applications, the abrupt motor control change between fully on and off does not result in a smooth control behavior.
- This issue can be solved by using a linear or **proportional** term instead.
- The formula for the proportional controller (P controller) is:

$$R(t) = K_P \cdot (v_{des}(t) - v_{act}(t))$$

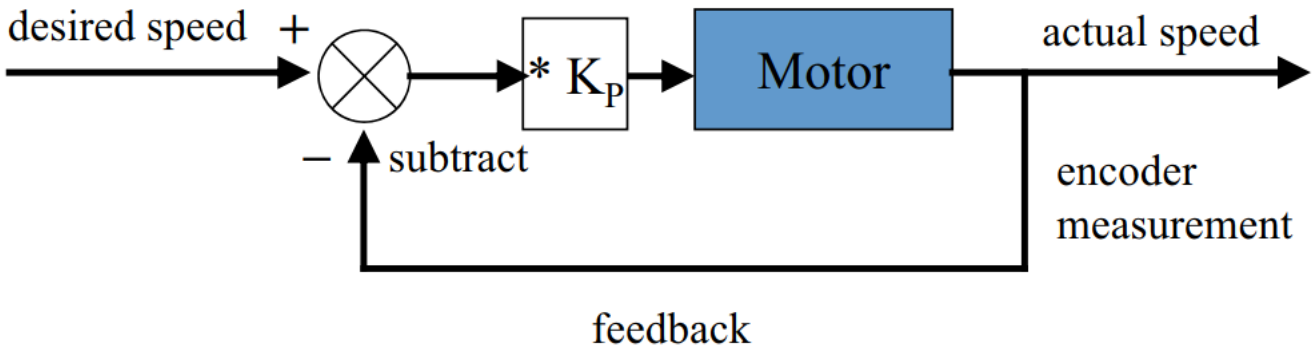
- The difference between the desired and actual speed

  [$\mathbf{V_{des}(t) - V_{act}(t)}$] is called the **error function**.

PID Control – Proportional Controller

- Varying the controller gain K_P will change the controller behavior.
- The higher a value for K_P is chosen, the faster the controller responds; however, a too high value will lead to an undesirable **oscillating** system.
- Therefore, it is important to choose a value for K_P that guarantees a fast response, but does not lead the control system to overshoot too much or even oscillate.

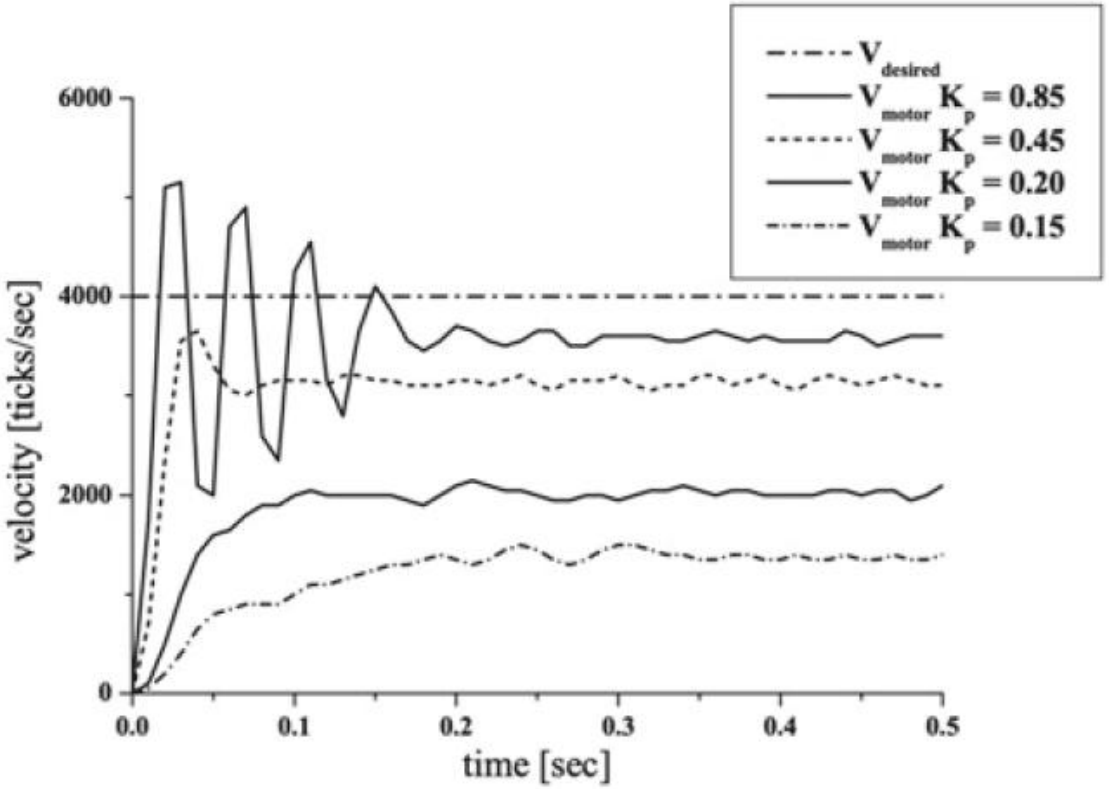
$$R(t) = K_P \cdot (v_{des}(t) - v_{act}(t))$$



Schematics for the Proportional controller

PID Control – Proportional Controller

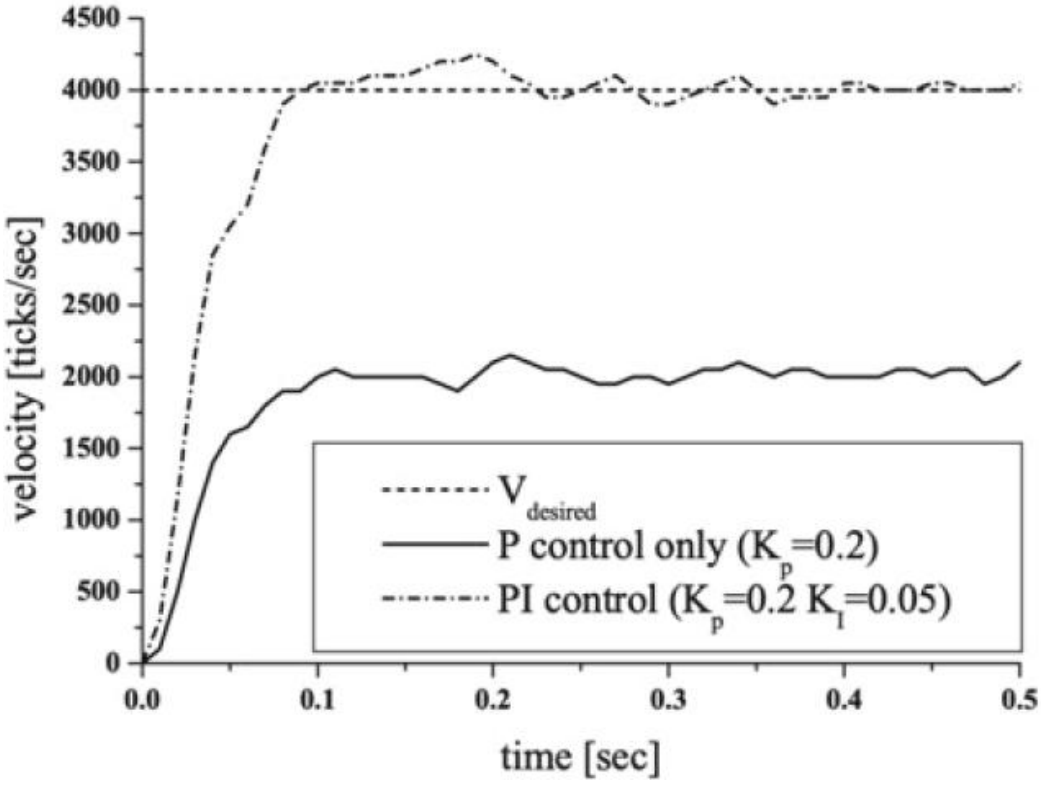
- The P controller's equilibrium state is not at the desired velocity.
- If the desired speed is reached exactly, the motor output is reduced to zero, as defined in the P controller formula shown above.
- Therefore, each P controller will keep a certain steady-state error from the desired velocity, depending on the controller gain K_P.
- As can be seen in Figure, **the higher the gain K_P, the lower the steady-state error**. However, the system starts to oscillate if the selected gain is too high.



Step response for proportional controller

PID Control – Integral Controller

- Unlike the P controller, the I controller (integral controller) is rarely used alone, but mostly in combination with the **P** or **PD** controller.
- The idea for the I controller is to reduce **the steady-state error of the P controller**. With an additional integral term, this steady-state error can be reduced to zero, as seen in the measurements in Figure.
- The equilibrium state is reached somewhat later than with a pure P controller, but the steady-state error has been eliminated



Step response for integral controller

PID Control – Integral Controller

- When using e(t) as the error function, the formula for the PI controller is:

$$R(t) = K_P \cdot [e(t) + 1/T_I \cdot \int_0^t e(t)dt]$$

- To rewrite this formula by substituting $Q_I = K_P/T_I$, so we receive independent additive terms for P and I:

$$R(t) = K_P \cdot e(t) + Q_I \cdot \int_0^t e(t)dt$$

- The way of implementing the I controller part is to transform the integration into ***a sum of a fixed number*** (for example, 10) of **previous error values**. These 10 values would then have to be stored in an array and added for every iteration.

PID Control – Integral Controller

- The proper way of implementing a PI controller starts with **discretization**, replacing **the integral with a sum**, using the trapezoidal rule:

$$R_n = K_P \cdot e_n + Q_I \cdot t_{delta} \cdot \sum_{i=1}^{n} \frac{e_i + e_{i-1}}{2}$$

- Now we can get rid of the sum by using the preceding output value R_{n-1}:

$$R_n - R_{n-1} = K_P \cdot (e_n - e_{n-1}) + Q_I \cdot t_{delta} \cdot (e_n + e_{n-1})/2$$

- Therefore (substituting K_I for $Q_I \cdot t_{delta}$):

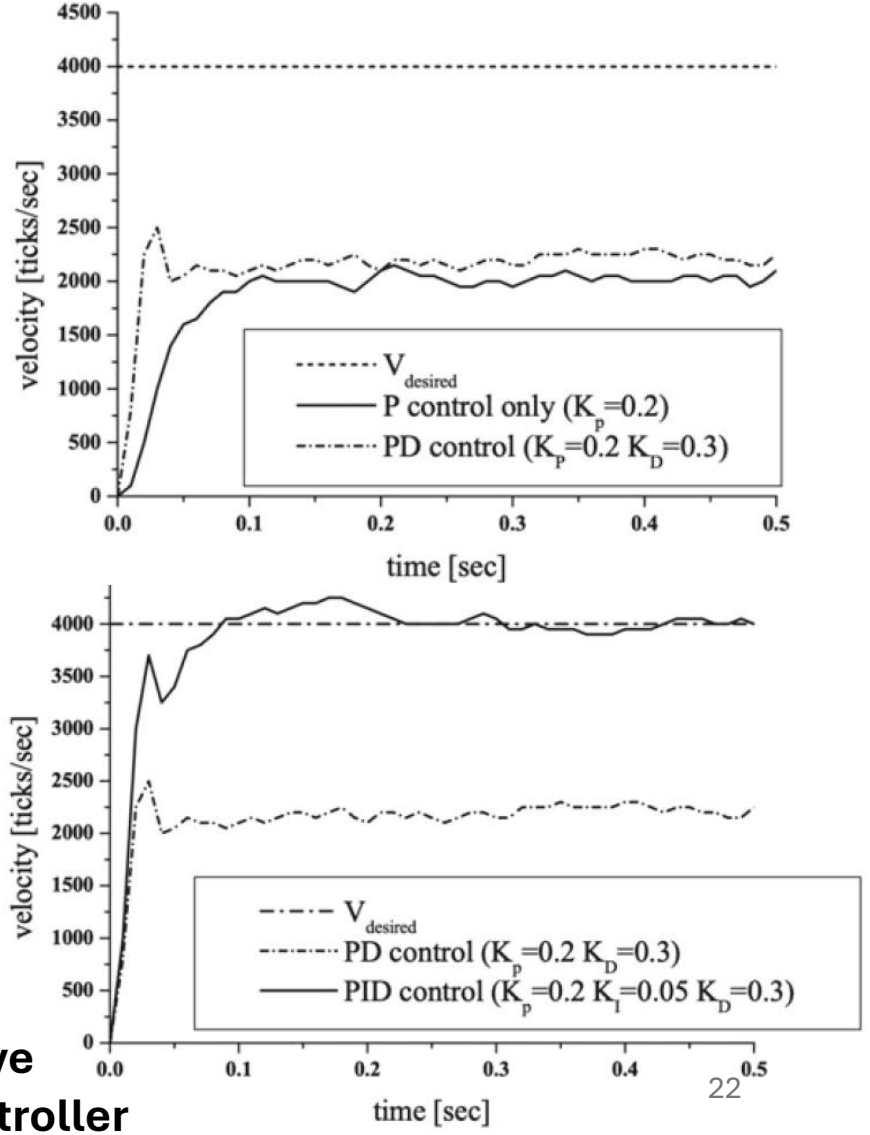
$$R_n = R_{n-1} + K_P \cdot (e_n - e_{n-1}) + K_I \cdot (e_n + e_{n-1})/2$$

PID Control – Integral Controller

- So, we only need to store **the previous control value** and the **previous error value** to calculate the PI output in a much simpler formula.
- Here it is important to limit the controller output to the **correct value range** and also store the limited value for the subsequent iteration.
- Otherwise, if a desired speed value cannot be reached, both **controller output** and **error values** can become arbitrarily large and invalidate the whole control process.

PID Control – Derivative Controller

- Similar to the I controller, the D controller (derivative controller) is rarely used by itself, but mostly in combination with the **P** or **PI** controller.
- The idea for adding a derivative term is to speed up the P controller's response to a change of input.
- The right Figure shows the measured differences of a step response between the P and PD controller (top), and the PD and PID controller (bottom).
- The PD controller reaches equilibrium faster than the P controller, but still has a steady-state error.
- The full PID controller combines the advantages of PI and PD. It has a fast response and suffers no steady-state error.



Step response for derivative controller and full PID controller

PID Control – Derivative Controller

- When using e(t) as the error function, the formula for a combined PD controller is:

$$R(t) = K_P \cdot [e(t) + T_D \cdot de(t)/dt]$$

- The formula for the full PID controller is:

$$R(t) = K_P \cdot [e(t) + 1/T_I \cdot \int_0^t e(t)dt + T_D \cdot de(t)/dt]$$

- Again, we rewrite this by substituting T_D and T_I, so we receive independent additive terms for P, I and D. This is important, in order to experimentally adjust the relative gains of the three terms.

$$R(t) = K_P \cdot e(t) + Q_I \cdot \int_0^t e(t)dt + Q_D \cdot de(t)/dt$$

PID Control – Derivative Controller

- Using the same discretization as for the PI controller, we will get:

$$R_n = K_P \cdot e_n + Q_I \cdot t_{delta} \cdot \sum_{i=1}^{n} \frac{e_i + e_{i-1}}{2} + Q_D / t_{delta} \cdot (e_n - e_{n-1})$$

- Again, using the difference between subsequent controller outputs, this results in:

$$\begin{aligned} R_n - R_{n-1} = {} & K_P \cdot (e_n - e_{n-1}) + Q_I \cdot t_{delta} \cdot (e_n + e_{n-1})/2 \\ & + Q_D / t_{delta} \cdot (e_n - 2 \cdot e_{n-1} + e_{n-2}) \end{aligned}$$

- Finally (substituting K_I for $Q_I \cdot t_{delta}$ and K_D for Q_D/t_{delta}):

$$\begin{aligned} R_n = {} & R_{n-1} + K_P \cdot (e_n - e_{n-1}) + K_I \cdot (e_n + e_{n-1})/2 \\ & + K_D \cdot (e_n - 2 \cdot e_{n-1} + e_{n-2}) \end{aligned}$$

PID Control – PID Parameter Tunning

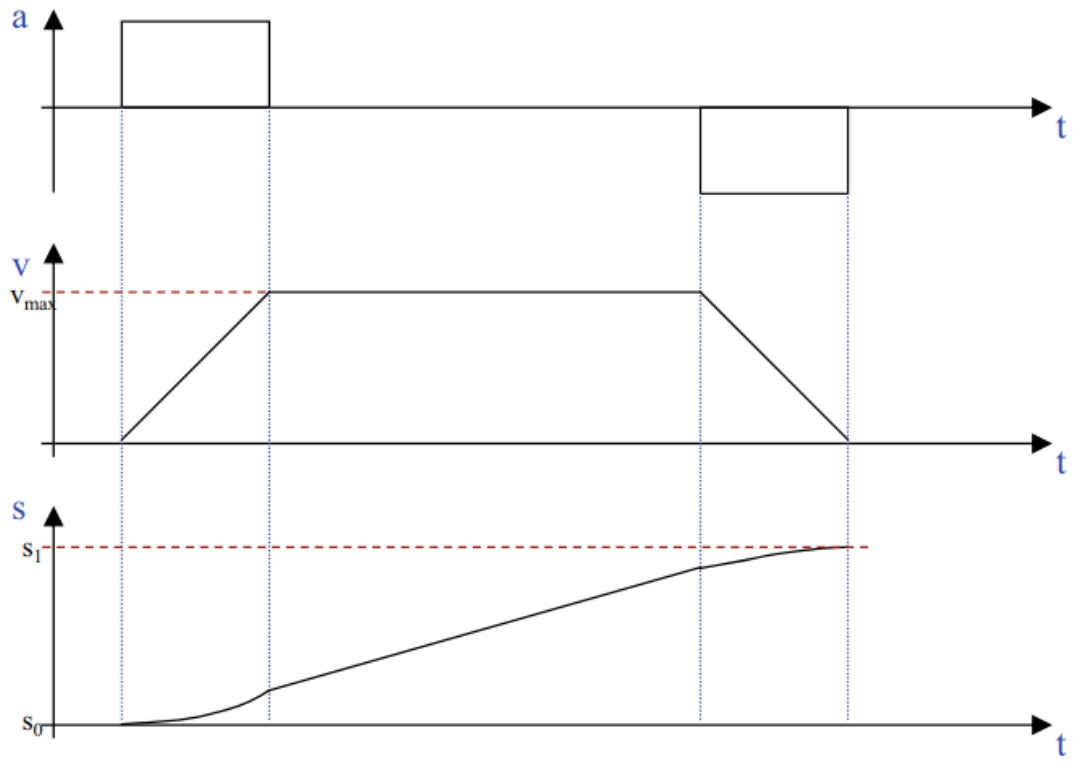
- The tuning of the three PID parameters K_P, K_I, and K_D is an important issue.
- The following guidelines can be used for experimentally finding suitable values:
  1. Select a typical operating setting for the desired speed, turn off integral and derivative parts, then increase K_P to the maximum or until oscillation occurs.
  2. If system oscillates, divide K_P by 2.
  3. Increase K_D and observe the behavior when increasing/decreasing the desired speed by about 5%. Choose a value of K_D which gives a damped response.
  4. Slowly increase K_I until oscillation starts. Then divide K_I by 2 or 3.
  5. Check whether overall controller performance is satisfactorily under typical system conditions.

Velocity Control and Position Control

- So far, a single motor at a certain speed can be controlled, but we are not yet able to drive a motor at a given speed for a number of revolutions and then come to a stop at exactly the right motor position.
- The former, maintaining a certain speed, is called **velocity control**, while the latter, reaching a specified position, is called **position control**.
- Position control requires **an additional controller** on top of the previously discussed velocity controller.
- The position controller sets the desired velocities in all driving phases, especially during the **acceleration** and **deceleration** phases (starting and stopping).

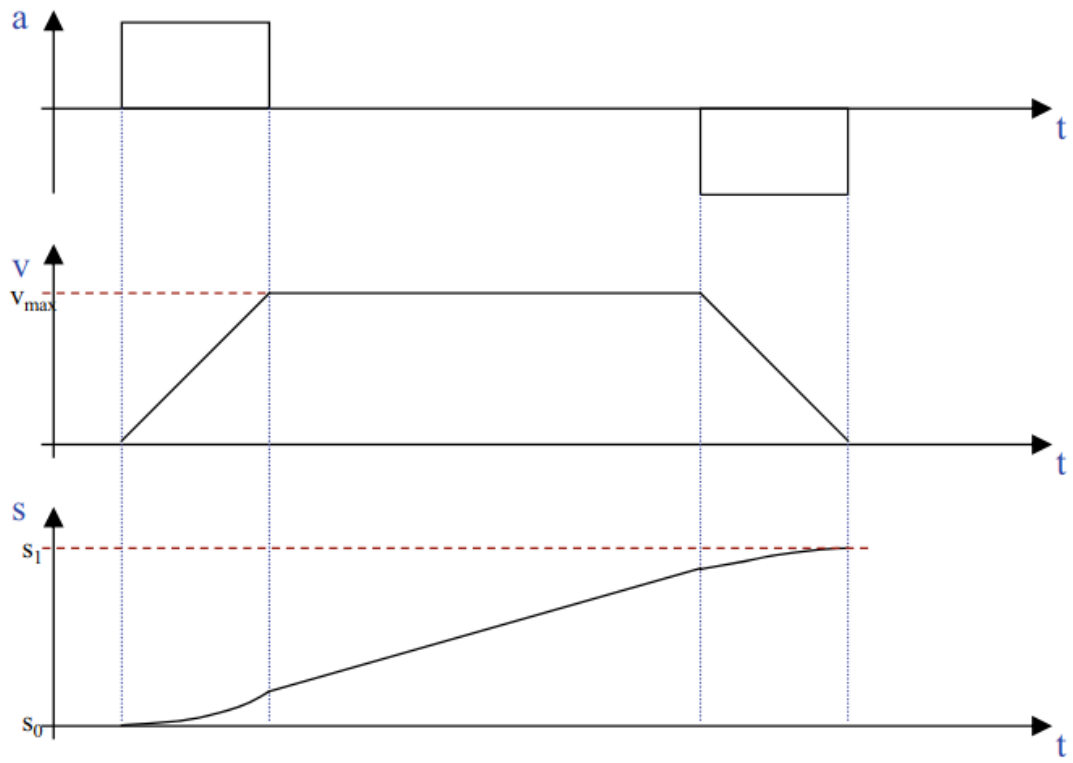
Velocity Control and Position Control

- Let us assume a single motor is driving a robot vehicle that is initially at rest and which we would like to stop at a specified position.
- The right Figure demonstrates the speed ramp for the starting phase, constant speed phase and stopping phase.
- When ignoring friction, we only need to apply a certain force (here constant) during the starting phase, which will translate into an acceleration of the vehicle.
- The constant acceleration will linearly increase the vehicle's speed v (integral of acceleration a) from 0 to the desired value v_{max}, while the vehicle's position s (integral of v) will increase quadratically.



Velocity Control and Position Control

- When the force (acceleration) stops, the vehicle's velocity will remain constant, assuming there is no friction, and its position will increase linearly.
- During the stopping phase (deceleration, braking), a negative force (negative acceleration) is applied to the vehicle. Its speed will be linearly reduced to zero (or may even become negative if applied too long—the vehicle would then be driving backward).
- The vehicle's position will increase slowly, following the square root function.



Velocity Control and Position Control

- The tricky bit now is to control the amount of acceleration in such a way that the vehicle:

  i) Comes to rest

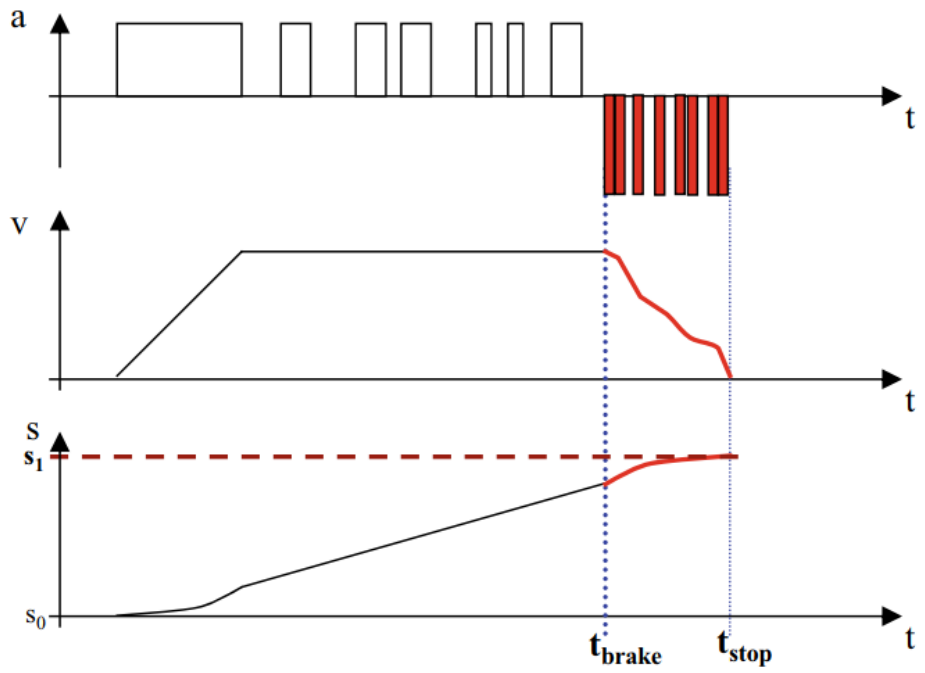
  (not moving slowly forward to backward)

  ii) Comes to rest at exactly the specified position

  (for example, we want the vehicle to drive exactly 1 m and stop within ±1 cm).

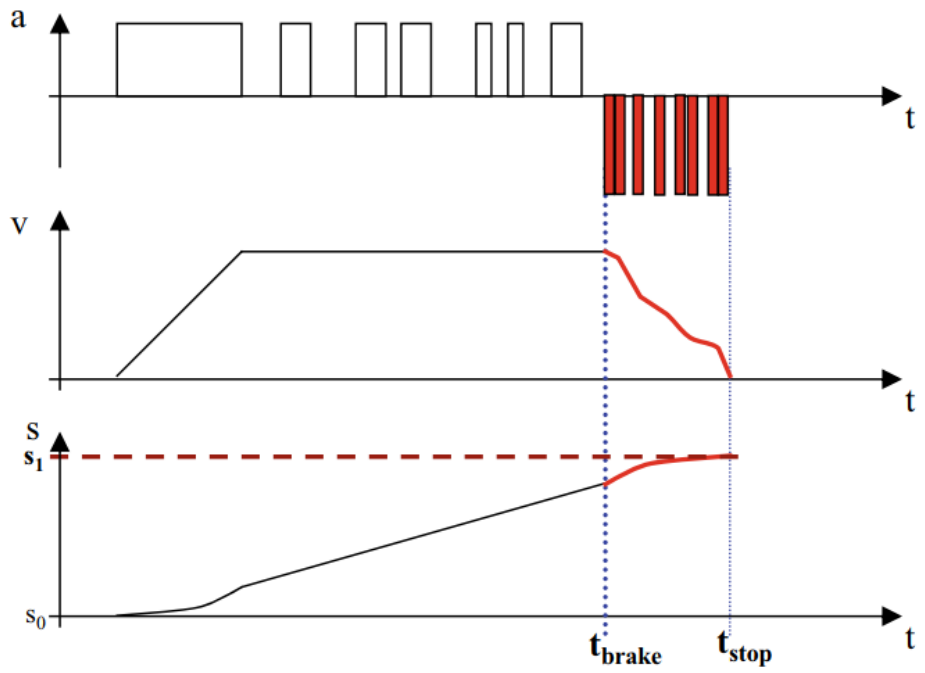
Velocity Control and Position Control

- The right Figure shows a way of achieving this by controlling (continuously updating) the braking acceleration applied.
- This control procedure has to take into account not only the current speed as a feedback value, but also the current position, since previous speed changes or inaccuracies may have already had an effect on the vehicle's position.



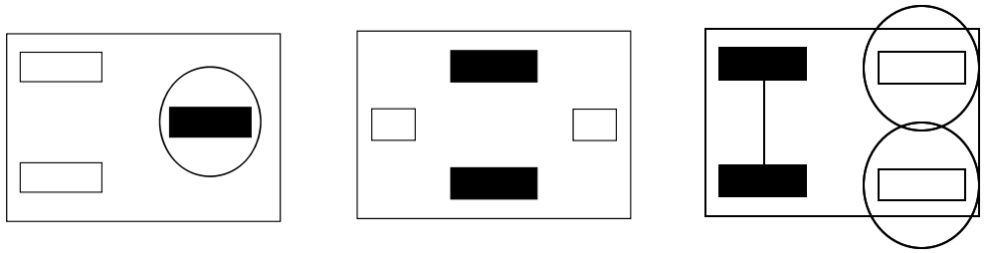
Velocity Control and Position Control

- In a real application scenario, there are of course roll resistance and air resistance, so the control algorithm also has to apply acceleration forces during the constant speed phase to maintain the desired speed, as shown in Figure.
- Braking forces using friction brakes can only reduce a vehicle's speed to zero, not below.
- As each vehicle has a certain inertia, the control forces have to be calibrated to be able to move the vehicle accurately at low speeds before it stalls.
- If our vehicle comes to a rest, e.g., 1 cm too short or too far from the desired position, then it may not be possible to shift it by only 1 cm.



Multiple Motors – Driving Straight

- So far, we have only looked at a single isolated motor with velocity control and briefly at position control.
- The way that a robot vehicle is constructed, however, shows us that a single motor is not enough, as shown in Figure.
- All these robot constructions require **two motors**, with the functions of **driving** and **steering** either separated or dependent on each other.
- In the design on the left and the design on the right, the driving and steering functions are separated.
- It is therefore quite easy to drive in a straight line (simply keep the steering fixed at the angle representing straight) or drive in a circle (keep the steering fixed at the appropriate angle).
- The situation is quite different in the differential steering design shown in the middle of Figure, which is a very popular design for small mobile robots.

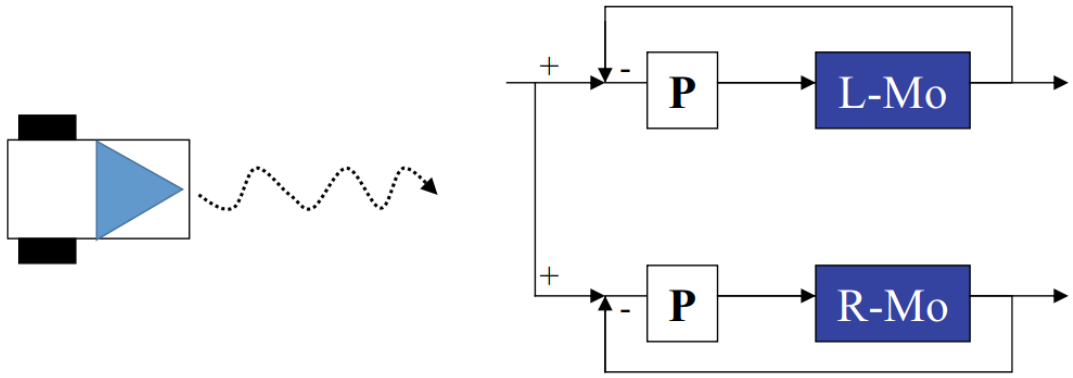


Multiple Motors – Driving Straight

- One has to constantly monitor and update both motor speeds in order to drive straight.
- Driving in a circle can be achieved by adding a constant offset to one of the motors.
- Therefore, a synchronization of the two motor speeds is required.

Multiple Motors – Driving Straight

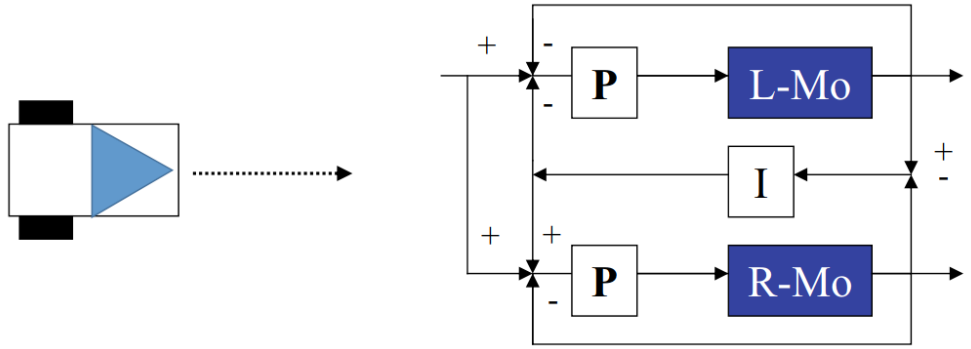
- The right Figure shows the first try for a differential drive vehicle to drive in a straight line.
- There are two separate control loops for the left and the right motor, each involving feedback control via a **P controller**.
- The desired forward speed is supplied to both controllers concurrently.
- Unfortunately, this design will **not produce a nice straight line of driving**.
- Although each motor is controlled in itself, there is no control link for any slight speed discrepancies between the two motors.
- Such a setup will most likely result in the robot driving in a wiggly line, but not straight.



Driving straight—first try

Multiple Motors – Driving Straight

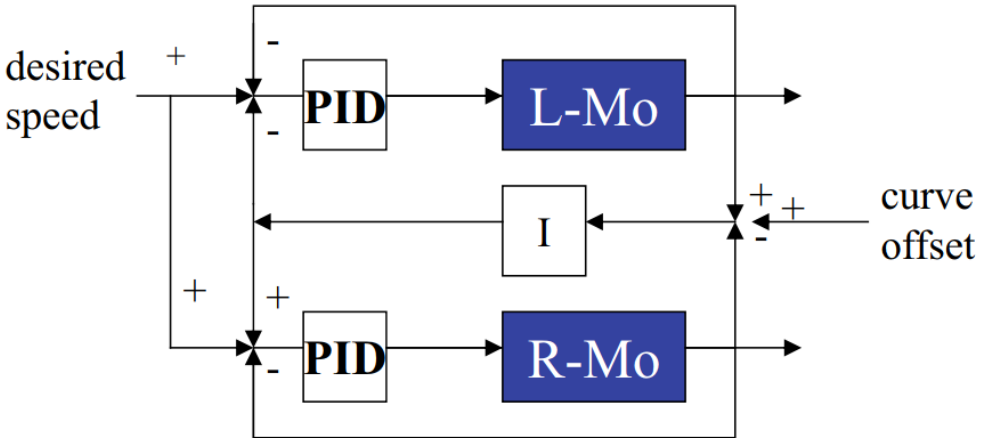
- An improvement of this control structure is shown in the right Figure as a second try.
- We now also calculate the difference in motor movements (position, not speed) and feed this back to both **P controllers** via an additional **I controller**.
- The I controller integrates (or rather sums up) the differences in position, which will subsequently be eliminated by the two P controllers.
- Note the different signs for the input of this additional value, which is the opposite sign of the corresponding I controller input. Also, this principle relies on the fact that the whole control circuit is very fast compared to the actual motor or vehicle speed.
- Otherwise, the vehicle might end up in a trajectory parallel to the desired straight line.



Driving straight—second try

Multiple Motors – Driving Straight

- We added another user input with the **curve offset**.
- With a curve offset of zero, the system behaves exactly like the previous one for driving straight.
- A **positive curve offset**, however, will let the robot drive in a counter-clockwise circle (a negative offset will let it drive in a clockwise circle).
- The radius can be calculated from the curve offset value.



Driving straight or in curves