Laboratory 4, Line following

Last update: 25-03-2024 Hash: 5276b08

1 Introduction

The goal of this document is to describe some theoretical aspects and implementation details, regarding the problem of autonomously following a line, by using a robot. The algorithm proposed in this document is a very simple one and presents some noticeable disadvantages; given that the proposed solution is just a suggestion and not a mandatory implementation, you are free to consider a different one if you will.

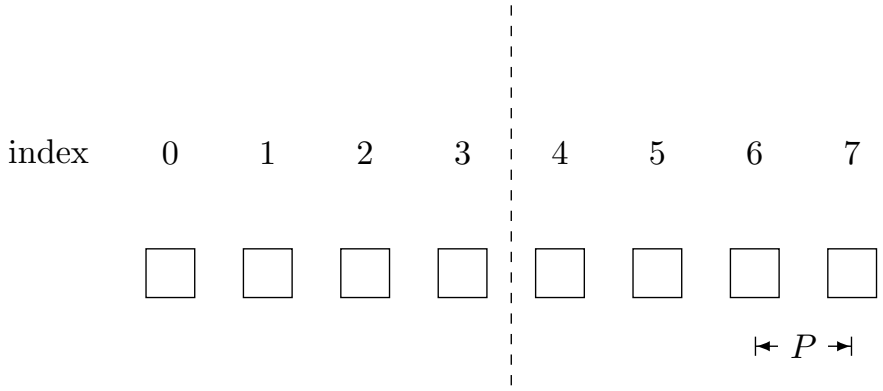
2 Preliminary operations

In the line following problem, the line is represented by a trace, usually drawn on an horizontal surface, characterized by an high contrast when compared with the portion of the surface which is not considered to be part of the line. A dedicated sensor is used to accomplish the goal of detecting the presence of a line with the aforementioned properties. Specifically, the type of sensor that is going to be taken into account in this document is an array of photo-diodes, each paired with a photo-transistor; each pair forms a digital sensor that allows to distinguish between a dark and a light area. In our case the sensor is placed on the front side of the rover.

2.1 Pololu QTR reflectance sensor

Taking the description directly from the datasheet: "The Pololu QTR reflectance sensors can be used to sense how much infrared light is reflected from a nearby surface." and the way it works should be known from the Laboratory 1 experience. Let's remind that the microcontroller can read from the reflectance sensor by interfacing with a port expander; this latter component being connected to the former, using an I2C bus connection. A mechanical property regarding the sensor that is relevant to our consideration is the pitch, which can be defined as the distance between the center of two different sensors. The list below contains some of the characteristic relevant to our considerations:

- Sensor output: digital
- Number of sensors N: 8
- Pitch P: 8mm or 0.008 m



3 Model

Knowing the properties of the sensors, we need to make some preliminary assumption:

- The line is wide enough to activate at least one sensor, almost every time. Its means that almost always during a transition between being a sensor "active" and having one of the adiacent ones "active", both sensor end up active;
- There is always at least one sensor that is not active;
- A line is detected if one or more adjacent sensors are active;

The initial idea is to drive the robot in a way that allow to maintain the line at the center of the array of sensors, i.e. having only sensor 3 and sensor 4 active.

3.1 Error definition

Let's define our goal in terms of line tracking error e_{SL}. Intuitively, the absolute value of the error should increase when the line is drifting from the center, and be equal to 0 when the line is centered.

$$f(x) = \begin{cases} e_{SL} = 0 & \text{when the line is at the center} \\ e_{SL} > 0 & \text{when the line is on the left side of the sensor} \\ e_{SL} < 0 & \text{when the line is on the right side of the sensor} \end{cases}$$

A possible mathematical definition of an error function satisfying the 3 conditions above is presented below. Let's first define the sequence:

$$b_n = \begin{cases} 1 & \text{if sensor } n \text{ is active} \\ 0 & \text{if sensor } n \text{ is not active} \end{cases}$$

representing the set of active sensors, and let ω_n, the distance between the sensor at position n and the center of the sensor array, formally defined as:

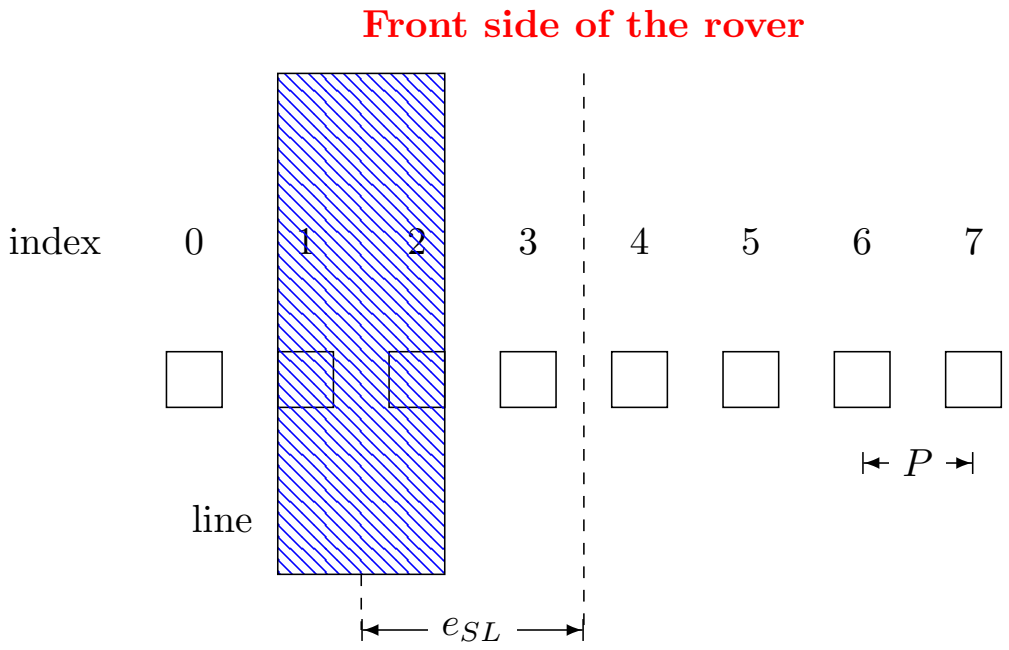
$$\omega_n = \left(\frac{N-1}{2} - n \right) P, \quad n \in 0, \ldots, N-1$$

where

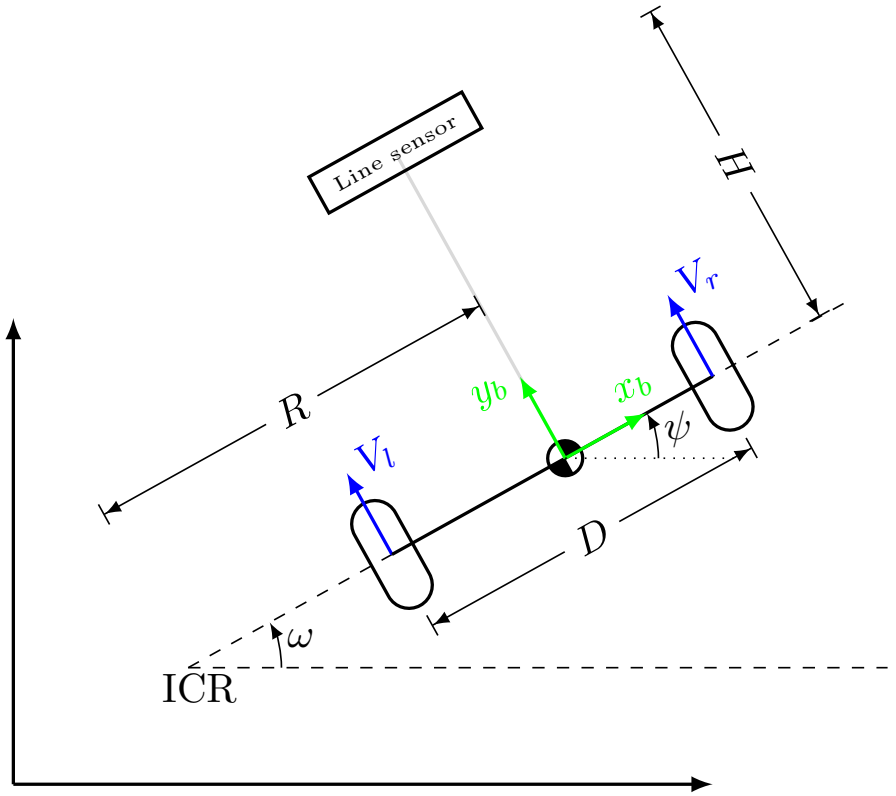
- P is the pitch for the considered line sensor array
- N is the number of sensors behaving to the array

The error can be finally defined as:

$$e_{SL} = \frac{\sum_{n=0}^{N-1} b_n \omega_n}{\sum_{n=0}^{N-1} b_n}$$



3.2 Kinematic model



Assuming pure roll condition for the wheels (in other words, assuming a non slipping condition), the following relations hold:

$$V_r = r \cdot \omega_r$$
$$V_l = r \cdot \omega_l$$

where:

- ω_r and ω_l are the angular speed of the right and left wheel, respectively, measured in [RAD/s]
- r is the radius of the wheel, measured in [m]

The ground contact speed of the left wheel v_L and the right wheel v_R causes the vehicle to rotate at an angular velocity ω. This rotation is around the Instantaneous Center Of Rotation (ICR), which is situated at a distance R from the vehicle's center. The distance between the wheels, or the track, is denoted as D. The following equations hold:

$$\begin{aligned} V_r &= \omega(R + D/2) \\ V_l &= \omega(R - D/2) \end{aligned} \tag{1}$$

By subtracting these two equations, we get:

$$V_r - V_l = \omega D \rightarrow \omega = \frac{V_r - V_l}{D}$$

It's important to note that the angle ψ, which is the angle between the global reference axis and the robot reference axis, is equal to the angle between the x-axis and the line connecting the ICR with the vehicle's center. Therefore, the yaw rate $\dot{\psi}$ is equal to ω.
Hence, the robot's yaw rate $\dot{\psi}$ can be expressed as:

$$\dot{\psi} = \frac{V_r - V_l}{D}$$

Referring to Eq. 1, we can define $V = \omega R$ which is the speed of the rover. Solving the two equations for V we get:

$$V = \left(\frac{V_r + V_l}{2}\right)$$

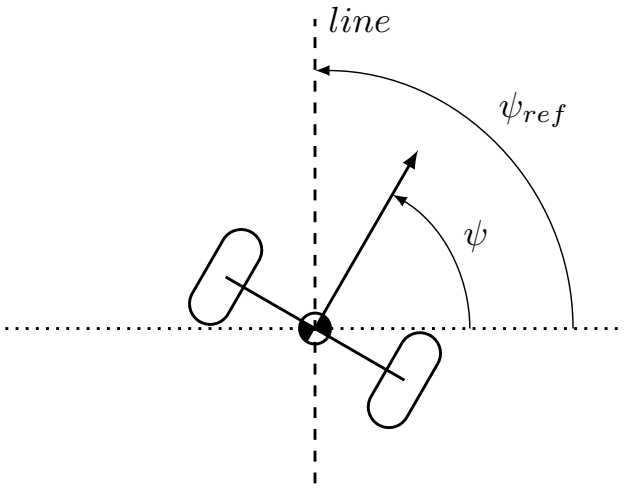
Notice that the speed of the rover is equal to the average longitudinal speed of the wheels.
From the relationships described above, we can derive the formulas which take into account both the longitudinal speed of the wheels and the yaw rate:

$$\begin{cases} V_r = V + \dot{\psi}\frac{D}{2} \\ V_l = V - \dot{\psi}\frac{D}{2} \end{cases}$$

4 Turtlebot mechanical parameters

- Wheel distance $D = 0.165$ m
- Distance between the line sensor and the vehicle's center $H = 0.085$ m
- Wheel radius $r = 0.034$ m

4.1 Yaw error



At this point it is possible to define the yaw error as:

$$\psi_{err} = \psi_{ref} - \psi$$

Considering the previously given definition of e_{SL}, it is possible to determine a relationship between ψ_{err} and the line following error:

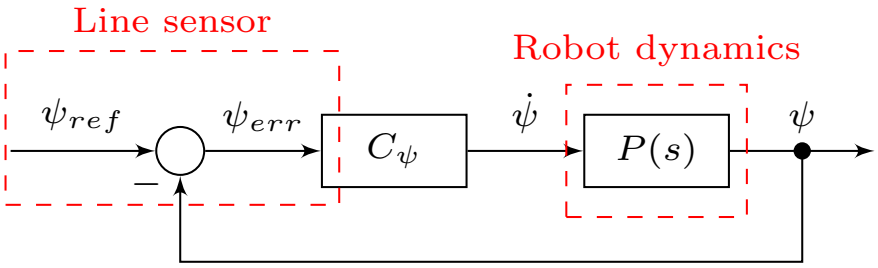
$$\psi_{err} = \arctan\left(\frac{e_{SL}}{H}\right)$$

Knowing that $\arctan(x) \approx x$ when x has values close to 0, and assuming this condition holds in our specific case, we can state that:

$$\psi_{err} \approx \frac{e_{SL}}{H}$$

Notice that in order to compensate a positive error, the controller, with the assumption that has been made, needs to produce a positive output signal and vice-versa.

5 Yaw controller



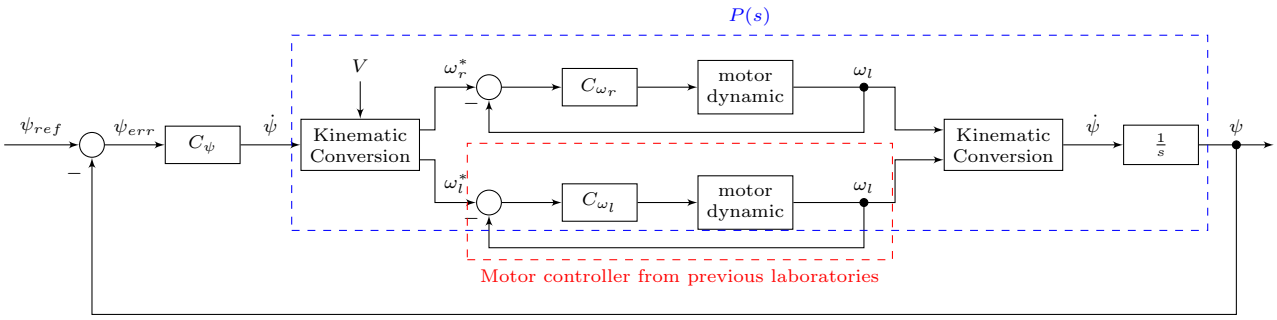
From the considerations made until now, we can model our controller such that:

- the input of the controller is the yaw angle error ψ_{err}
- the output of the controller is the yaw rate, $\dot{\psi}$

Having each wheel feedback controlled in its speed by PID controllers we can approximate our plant $P(s)$ as an integrator, it follows that

$$P(s) = \frac{1}{s} \tag{2}$$

Our goal is to regulate the yaw angle to the desired constant set point, with zero steady-state tracking error. Given that we are under the assumption for our system to behave like an integrator, it is possible to conclude that is sufficient, in order to reach our goal, to use a P-controller. Notice, however, that to uniquely determine the desired wheel speeds to pass to the motor control PIDs, it is not sufficient to provide $\dot{\psi}$: the longitudinal speed of the rover V is also required and it is up to you how to tune it.



6 Exercises

6.1 Exercise 1

Write a program for the Turtlebot, able to follow a line.
As an initial approach, keep things simple!! Do NOT implement the yaw controller proposed above. Try to keep the robot centered in the line. According to the error make the bot turn by increasing the rotation speed of one wheel and decreasing the other one. You can test the TBot on the test circuit.

6.1.1 Implementation details:

The general architecture of the software implementing the controller is similar to the one used for the motor control. The control routine is executed periodically from inside the callback of a properly configured timer.

You have to use the code you wrote for the previous labs. Is suggested to start from the code for the motor control and add all the necessary functions/initializations for the i2c.

If you don't have a working code from the previous labs, we will provide a base project. ASK FOR IT. However, we will take this into account in the final evaluation.

```c
void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef * htim )
{
    /* Heading angle (yaw ) control routine */
    if(htim -> Instance == TIM6 )
    {
    /* 1. SX1509 (1) - Get Bank B inputs ( Line Sensor ) */
    /* 2. Evaluate line sensor error */
    /* 3. Calculate which wheel should brake or accelerate */
    /* 4. Calculate the wheel target speed */
    /* 5. Use the computed data and to control the angular speed of the motor accordingly */
    }
}
```

6.2 Bonus

1. Implement the yaw controller. You can use the method proposed in the previous section or any other method you think would be suitable. Any innovative contribution is welcome.

2. Implement an "adaptive control" strategy to achieve the best lap time on the test circuit. For example, the TBot can be programmed to increase speed on straight sections of the track and slow down when approaching a corner.

3. Improve the control strategy to enable the TBot to navigate a more challenging circuit that includes obstacles, missing sections, noise, and other impairments.

4. Whatever you want. Innovative contributions and ideas are welcome.