University of Nottingham
UK | CHINA | MALAYSIA

Department of Electrical & Electronic Engineering

EEEE1002: Session 3 Episode 2
Optical Line Following

Version: January 2022

EEEE1002: Session 03 Episode 02

Optical Line Following

Introduction

Line following is a common challenge in robotics and is the focus of many competitions the world over. There is therefore a multitude of online resources dedicated to line following, however, please bear in mind that when using these resources that they are specific to the vehicle they are based around. The key to success is developing a solution that works for your own robot vehicle, the EEEBot.

Overview

The aim of this session is to modify your car so that it can autonomously follow a line. This involves the design and build of an optical infrared (IR) sensor array suitable for interfacing to the analogue pins of an Arduino microcontroller; consideration of the arrangement of your optical sensors; implementing a simple line following scheme; implementation of a control scheme to control the motors in response to the detected line position to implement an improved line following scheme.

Objectives

In episode 1 you should have:

- Designed a circuit that will provide an **analogue** output from an IR LED emitter/photodiode detector sensor paired
- Constructed and tested the circuit
- Modified the circuit design to provide a **digital** output from the IR sensor
- Construct and tested the comparator circuit and noted the advantages and disadvantages of this over the analogue output version
- Answered the question - could the circuit be used to measure distance?

In this episode, you will use the knowledge gained from Episode 1 to construct an array of analogue optical sensors (up to a maximum of 6). You are free to choose how many sensors to use and how they are distributed in the array based on the information provided in this document and any background research you have carried out. You are therefore required to:

1. **Design your sensor array**, interface the array to an Arduino and demonstrate it working by capturing appropriate data.

2. **Interface your Arduino and sensor array to the ESP32 on the EEEBot** so that the Arduino can send information to the ESP32 to control the motion of the vehicle drive motors as was covered in Session 2 i.e., the ESP32 is to be treated as a slave device and the firmware from Session 2 can be used unmodified for this purpose – see S03 Introduction, Slide 42 for the system overview.

3. **Implement a basic line following scheme** that uses only the centre **two** sensor pairs of your array to follow a black line.

4. **Implement an improved line following scheme.**

    a. Implement a weighted average formula on the Arduino to calculate the position of the line seen by your array of sensors.
    b. Implement PID control on the Arduino to calculate the change in motor speed required to better track the line.

5. Link the weighted average and PID codes so that the car motors respond to the position of the line.

6. Implement a method to individually calibrate each sensor in the array.

7. Demonstrate the line following ability of your vehicle.

Assessment

The assessment will be in the form of an individual lab report covering the work undertaken in session 3 and session 4 and will be worth 20% of the module total. Session 3 contributes 50% of the work required for this report. The final report should be presented in the correct format for a technical document and should include (but not limited to) the following sections that relate to the work undertaken in this session:

1. Optical Sensors Overview.
    a. Use of optical sensors to provide analogue and digital information.
    b. Use of an optical sensor to measure distance.
2. Optical Sensor Array.
    a. System overview.
    b. Optical sensor array design (including schematic) and discussion of the chosen design.
    c. Amplifier circuit design, its operation, and a discussion of testing results.
    d. Discussion of the result of tests carried out on the sensor array.
3. PID Control Scheme
    a. System overview.
    b. Implementation of the weighted average formula.
    c. Implementation of the PID formula.
    d. Code optimisations.
    e. Tuning process.

It is therefore important that you keep a comprehensive log of your work as you progress through the objectives.

Please note:

Poor writing, layout and presentation of the lab report will heavily influence the final grade awarded for the report. Therefore, please reflect on the feedback you will have received on your previous report to inform presentation style.

Note: achievable grade is now linked to the level of technical achievement.

Contents

1. Introduction

In this episode, the EEE-Bot will be modified through the addition of an IR sensor array so that it can follow a black line on a white background. This will be achieved by designing and constructing a sensor array that is connected to the analogue inputs on an Arduino that sends motion control information to the EEEBot ESP32.

One of the initial objectives is to carry out a set of experiments to support an argument for using an array of optical sensors to detect a line using the analogue signals from the sensor system.

A weighted average process is to be applied to the sensor reading to determine the distance of a reference point (typically centre of sensor array) from the black line

A PID control algorithm will be implemented in Arduino code to adjust the motor speed in response to the detected position of the line relative to the vehicle.

2. Basic Line Following Scheme

A simple implementation using just two sensors and the principle behind its operation is shown in **Figure 1**.

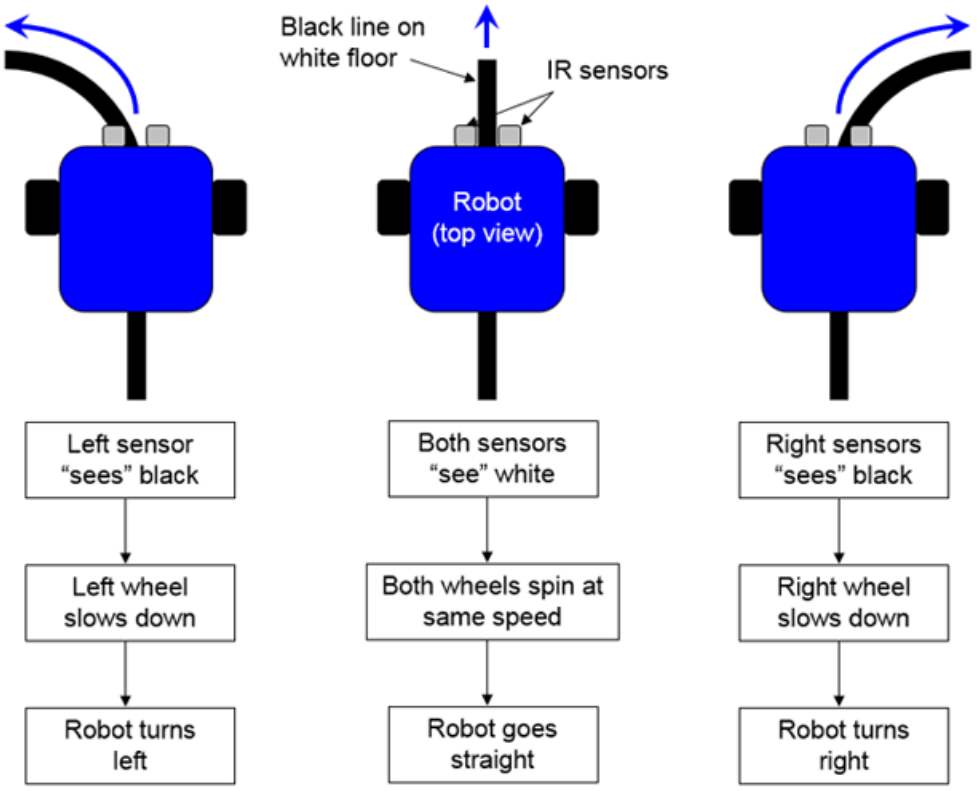

Figure 1 Operation of a two-sensor line following system

For this session's objectives note that:

- An array using up to 6 x IR emitter/detector pairs connected to Arduino analogue pins is to be designed and built
- The analogue pins (A0 to A7) accept an input voltage in the range of 0 to 5v
- The voltage at the pin is converted to a numeric value between 0 and 1023 (10 bit Analogue to Digital Converter as discussed in the module Information and Systems) that the Arduino can work with
- Analogue pins A4 and A5 are to be reserved for I^2C communication

Infrared Sensors for Line Detection

A background on sensors and how they function was covered in episode 1. To detect the position of the line an array of up to 6 sensors will be mounted at the front of the car and **Figure 2** shows an example of a four-sensor array mounted on the front of the EEEBot.

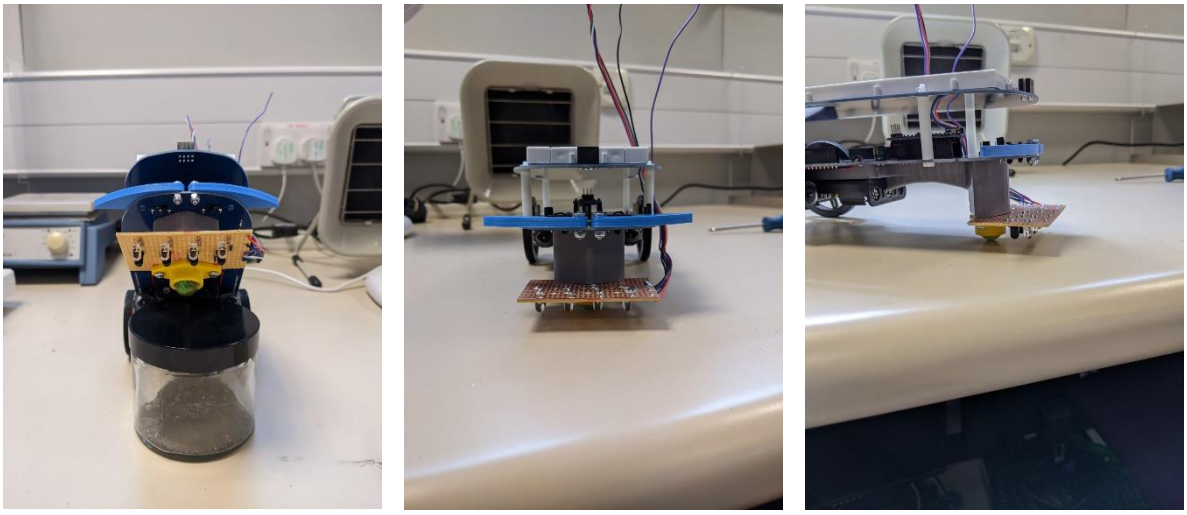

Figure 2 Example of a four-sensor IR reflectance array mounted on the EEE-Bot

The line following subsystem will require the design and construction of an amplifier circuit for your IR sensors, as was covered in episode 1 for a single sensor. This will take the very small current produced by the photodiode and convert it into an analogue voltage that the Arduino can read. A suitable circuit for this is the **trans-impedance amplifier**, the background theory of operation of which circuit is covered in the module Information & Systems (EEEE1004). The circuit schematic of a trans-impedance amplifier is shown in **Figure 3**.

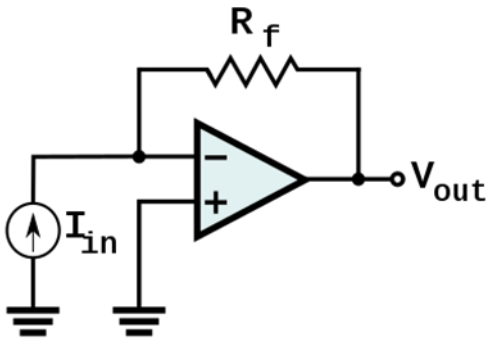

Figure 3 Basic trans-impedance amplifier circuit

You will be provided with MCP6292 dual op-amp ICs, each package containing two operational amplifiers (op-amps), therefore up to three ICs will be required for the completion of this session. The amplifier circuits are to be finally built on the EEE-Bot's expansion board.

In episode 1 you should have built a single complete sensor circuit (including the amplifier) on a breadboard and used this to assess the operation of an optical sensor circuit. It was also required that you implement a comparator circuit to provide a digital output and you will need to discuss why this may not be the most suitable solution for line following.

Note: The Arduino has sufficient analogue inputs to accept signals from up to six sensors, as A4 and A5 are used for I^2C communications which may be required later.

Once you have proven and analysed the single sensor circuit in episode 1 only then design and build your chosen sensor array. Once built and tested and interfaced to the EEEBot, use only the middle two sensors to implement a basic line following robot.

3. Improved Line Following Scheme

To remain centred over the line the EEE-Bot needs to know whether to turn left or right as it deviates from the line, therefore the position of the line under the sensor array must be known. This is achieved by using a number of sensors and implementing a weighted average formula that takes the sensor values, their positions along the array, and produces the position of the line relative to a fixed point of **reference** on the EEE-Bot. The following introduces the concept using an array with eight sensors as an example. The same approach can be applied to an array with any number of sensors.

The Weighted Average Formula

The weighted average formula is given in equation (1) and **Figure 4** shows how the parameters relate to a physical sensor array of eight sensors. It is important to note that this assumes the sensors give a high output when exposed to the black line.

$$X_{pk} = \frac{\sum_{i=0}^{7} X_i S_i}{\sum_{i=0}^{7} S_i} \quad (1)$$

Given the amplitude (intensity) of the signal from each sensor, S_i and the sensor's distance from a fixed reference point, X_i , the weighted average formula will give the distance of the peak amplitude from the fixed point of reference i.e. the position of the black line. In the example here the fixed point of reference is the leftmost edge of the vehicle that the sensor is attached to.

Note: though the point of reference in this example was chosen to be the left-most edge of the vehicle you are free to choose a similar point of reference or any other of your choosing e.g the centre point of the array.

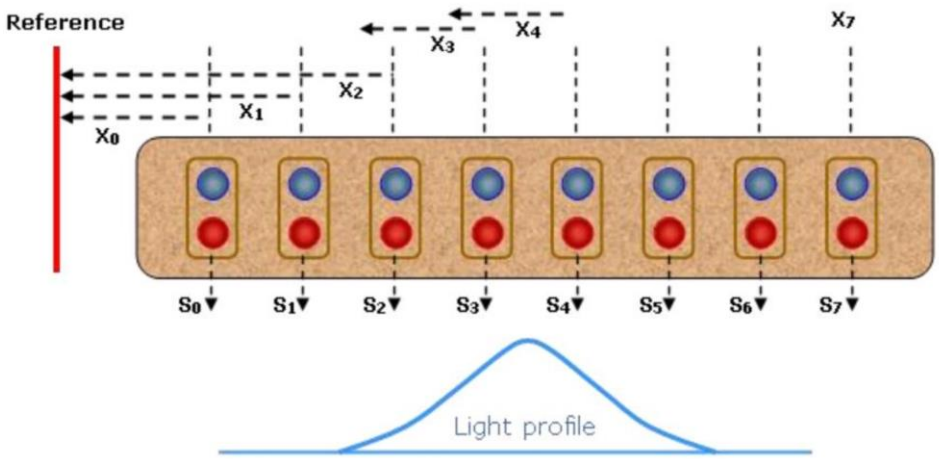

Figure 4 Eight sensor array with parameters for the weighted average indicated. X_i is the distance from a reference point and S_i is the sensor output intensity assuming the sensor array is centred over a black line

Once the position of the line being followed is known, a correction can be applied to the wheels to steer the vehicle so that the line always falls under the centre of the vehicle. If this is simply done by applying a left and right turn it can lead to the vehicle's movements becoming erratic as it tries to find the line. This can be improved by implementing a common method of control known as Proportional Integral Derivative control, or PID.

Proportional Integral Derivative Control (PID)

Note: The theory behind PID control will be covered in specific control modules later in the course. The formula for PID control is provided in this section, however, it will be up to you to implement it in code to run on the Arduino.

PID is a control algorithm. It takes the **error** between a chosen **setpoint** and the **current measurement point** and produces an output to quickly correct this error. For a line following car, the setpoint will be the fixed distance from the centre of the car to the reference point used in the weighted average algorithm. The current measurement comes from the weighted average of the sensor signals. The error, which is used as the input to a PID control algorithm is then given by (2):

$$error = setpoint - weightedAverage \tag{2}$$

The PID algorithm consists of three parts, each of which uses the error differently to affect the output. The proportional part uses the instantaneous error, the integral part uses the integration of the error, and the differential part uses the differentiation of the error. Each part has a constant that is set to tune the response of the control. The PID formula is given in (3)

$$u = K_p e + \int_0^t K_i e \, dt + \frac{d}{dt} K_d e \tag{3}$$

If the formula is calculated repeatedly at a fixed time period, then it simplifies to:

$$u = K_p e_n + K_i \sum_0^n e_n + K_d (e_n - e_{(n-1)}) \tag{4}$$

where e_n is the current error value. The resulting output variable, u, is then used to control the speed of the motors to keep the vehicle centred on the line. When u is zero the car will be centred over the line and the vehicle should move straight by keeping the left-hand and right-hand motors at the same speed. When u is non-zero the car will be travelling either to the left or the right of the line and therefore the speed of the motors need to be altered to steer the car back over the line.

This can be achieved by choosing a base speed that results in the car travelling in a straight line and subsequently modifying this base speed with the output from the PID algorithm as shown in (5)

$$\begin{aligned} leftMotorSpeed &= baseSpeed + u \\ rightMotorSpeed &= baseSpeed - u \end{aligned} \tag{5}$$

Once the code has been implemented the PID controller will need to be tuned. This involves setting the K_p, K_i, and K_d constants so that the car smoothly follows the line. You should start with just K_p then once the response is reasonable adjust the other two.

- K_p, the proportional constant directly multiplies the error and should therefore set a reasonable motor speed change given the error magnitude.
- K_i, the integral component multiplies a running sum of the error and therefore its effect increases the longer the robot is not on track.
- K_d, the derivative component multiplies the change in error therefore its effect depends on how quickly the line is changing compared to the course of the robot.

Refer to EEEE1002_S03_Introduction for further info.

4. Calibration of the Sensor Array

During the testing of your sensors, you will likely have noticed that each one has a slightly different response. If this is not accounted for it will affect the performance of your line following solution.

A calibration scheme is needed to account for the variation in the sensors, ensuring they give a uniform result. One approach is outlined in **Figure 5** obtains the values from each sensor as it sees black and then white, these values (which may be different for each sensor) are then **mapped** onto a common range.

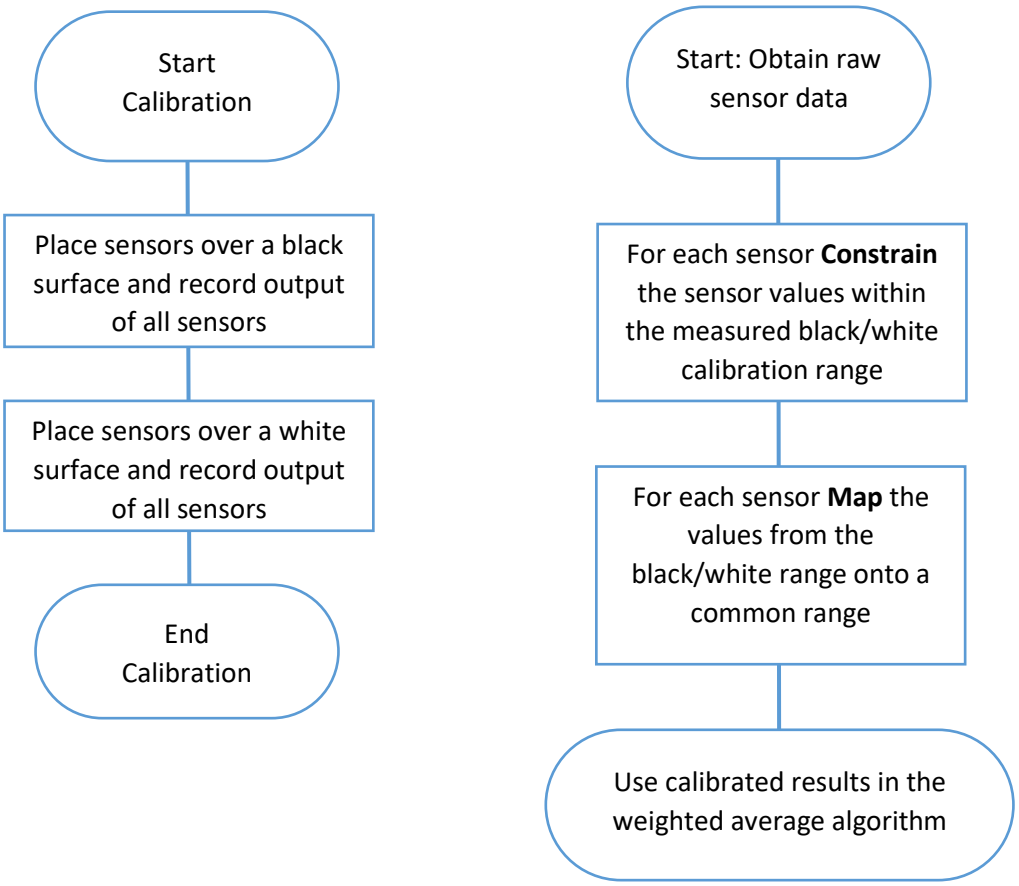

Figure 5 Acquisition of sensor calibration values and achieving calibrated values by mapping the acquired sensor values onto a common range. Note that Constrain and Map are functions within the Arduino programming language.