LINE FOLLOWER WITH PID CONTROL

Measurements' lab

Team members:

- سلمى علاء محمد سعيد 18010801
- عز الدين السيد محمد السيد 18011052
- عبد المنعم بهاء الدين عبد المنعم السيد الديب 18011045
- محمد رشاد نصر المزين 18011462
- عثمان على عثمان طقيشم 18011049
- محمد رشدى محمد على 18011463
- سعيد مجدى شحاته عبدالواحد 18010789
- محمد احمد سعد محرم 18011352

Team number: 22

Contents

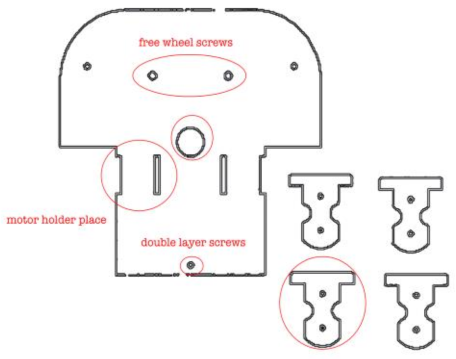
Introduction

The purpose of this project is to implement a line follower robot with PID control, The car is designed to move automatically and follow a path (black line on white surface) using a group of infra-red sensors to identify the black line. The task was completed by measuring analog readings from the IR sensors then using those readings to control 2 DC motors speed using PID control.

Components:

- Arduino uno
- 4X TCCRT5000 line sensors
- 2X DC yellow motors
- 3X 4V rechargeable batteries (connected in series)
- L298 Motor driver
- Body
- Breadboard and wiring

Design:

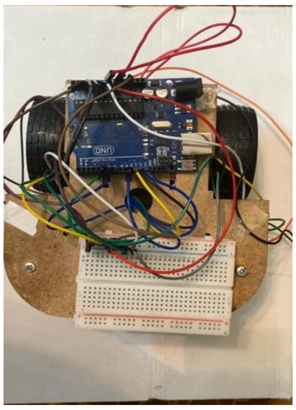


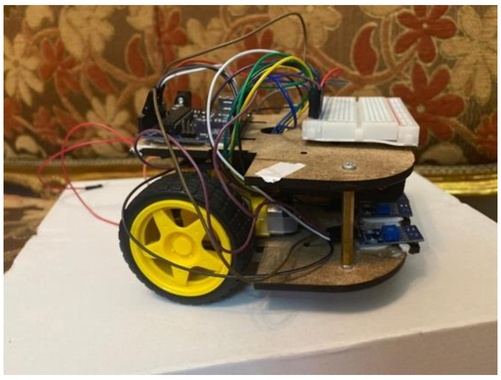
Robot body designed by AutoCad software

Material: wood with 4mm thickness

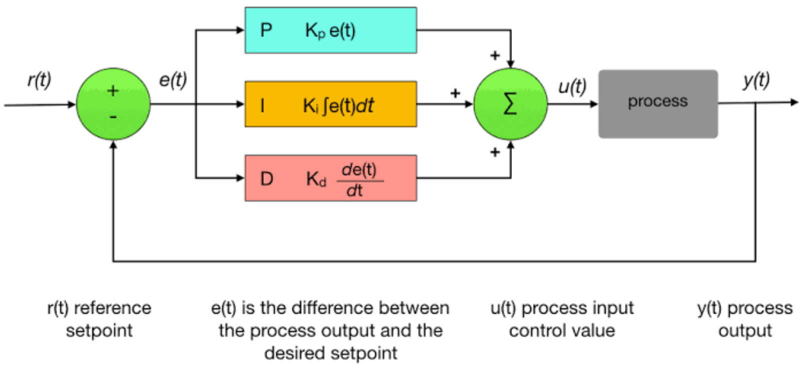
Dimensions: 14 x 14.5 cm

Motor holders are designed to fit the yellow motor used in this robot





PID controller



PID stands for Proportional Integral Derivative controller it's a control closed loop with feedback which continuously calculates error ($e_r(t)$) as a difference between predefined set-point and actual measured value and generate correction for this error in terms of its three parts P, I and D.

PID is widely used in control systems because of its fast, continuous, and accurate response to the error

P – Proportional: directly proportional correction to the error that means if error is very high in positive direction the proportional correction will be very high in the same direction. by increasing the P term, the system may overshoot and oscillate more.

I – Integral: It is used to decrease the steady state error, as the term I is integration of the actual value of the error with respect to time, so by using the PI controller, we can eliminate the steady state error by using a considerable value of integral, as for errors even small errors the integral response will be high, and the term will continue to change till the error becomes zero.

D-Derivative: It is used to predict the future behaviour of the error, however we can’t use the D controller alone as if the error is constant, the controller will behave like there is a zero error, so we use PD controller, this controller can be used to improve the transient response of the system.

PID can be implemented in software and digital systems using discretisation by taking samples at different times with equal delay between each time called sampling time and always keep current sample and previous sample to use them in integral and derivative parts then we can implement the equation easily.

Mathematical (academic) equation of PID controller in S-domain:

$$U_{PID} = K_c \left(1 + \frac{1}{T_i} * \frac{1}{s} + T_d * s \right) * error$$

Where:

$$K_p = K_c \qquad K_i = \frac{K_C}{T_i} \qquad K_d = K_C * T_d$$

Implemented algorithm

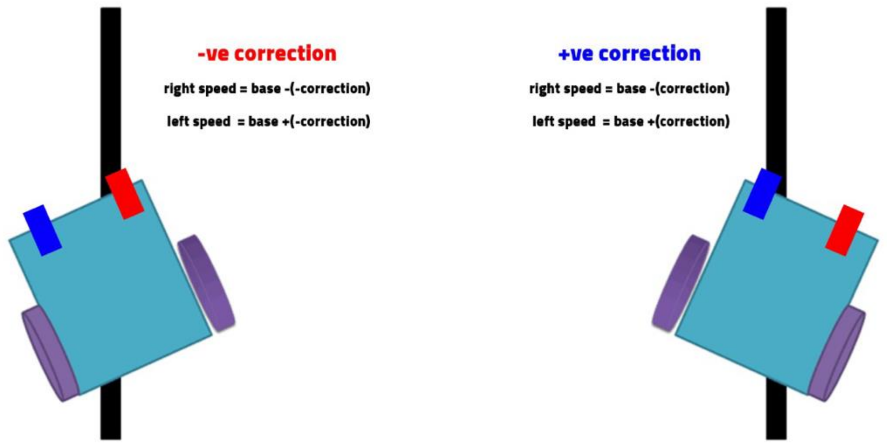
Used functions:

- PID variables calculations: we implemented this function as explained in the tutorial, we initialized the set point in the void setup function using the sensors reading at the beginning of the track, as the used pins of the sensors are analog, we used the weighted average method to get a signed value for the position (input to the PID controller) that will determine the direction in which the robot is shifted from the line. Also, the weighted value is important in the correction value as one of the two side sensors will have high values in case of errors because they have higher weight than center sensors so it will give higher correction value.
  Every time the sampling time (calculated by the millis function) is reached, the function will be called.

Since the set point is already defined, error = set point – position, then all PID parameters will be calculated. After that we used the equation to get the correction value (output of PID controller).

- Set motors: in this function we take the correction value calculated in PID function and add it to base speeds of the two motors. But we add it with different signs:

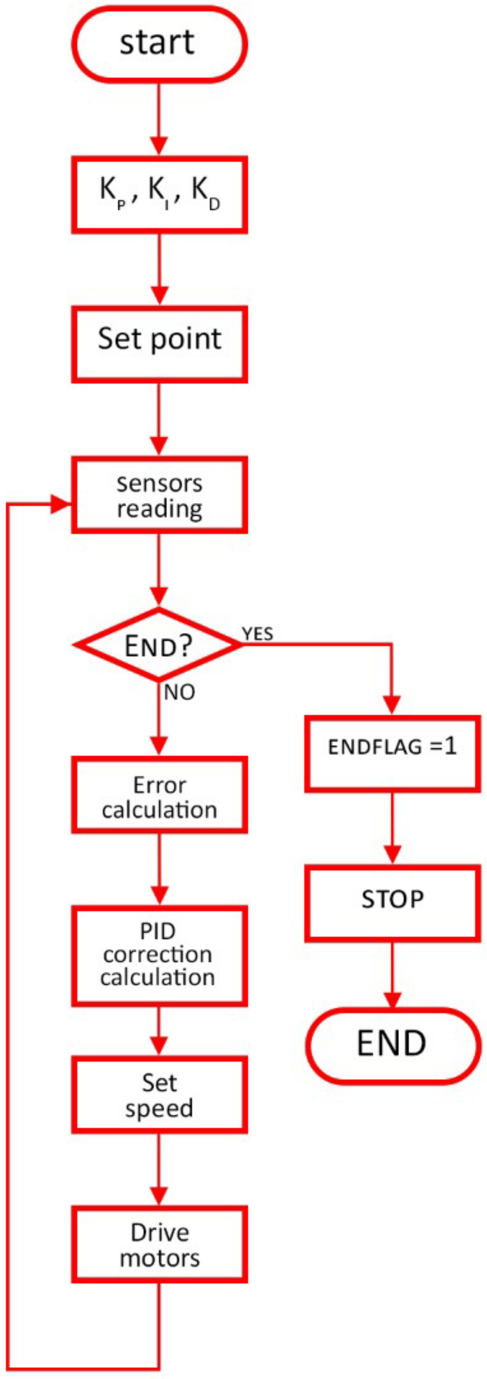
  * -ve sign for the right speed ——> because right sensors' values are negatively weighted so their correction value will be negative so the right motor's speed will increase
  * +ve sign for the left speed ——> because left sensors' values are positively weighted so their correction value will be positive so the left motor's speed will increase


-ve correction
right speed = base -(-correction)
left speed = base +(-correction)
+ve correction
right speed = base -(correction)
left speed = base +(correction)

2nd step is to constrain speeds after correction between 0 and 250 to make them suitable for PWM signal values if speed is higher than 250 the function will set it to 250 and if it is lower than zero the function will make it zero. then we pass both two speeds to motor drive function.

- Motor drive: in this function we get right and left speeds as arguments then we pass them to analogWrite function to generate PWM signal at PWM pins of the two motors (except the case of end point where all sensors read black so the sensor sum will be higher than 1500 then we raise end flag to end motion and stop robot directly in void loop function). Then we set direction pins to be forward as we don't have backward motion all the time direction of each motor is either forward or just stop.

- End: this function is used to stop the motors; it is called in the void loop when end flag == 1.
- Note: the code is attached with comments explaining the full code.



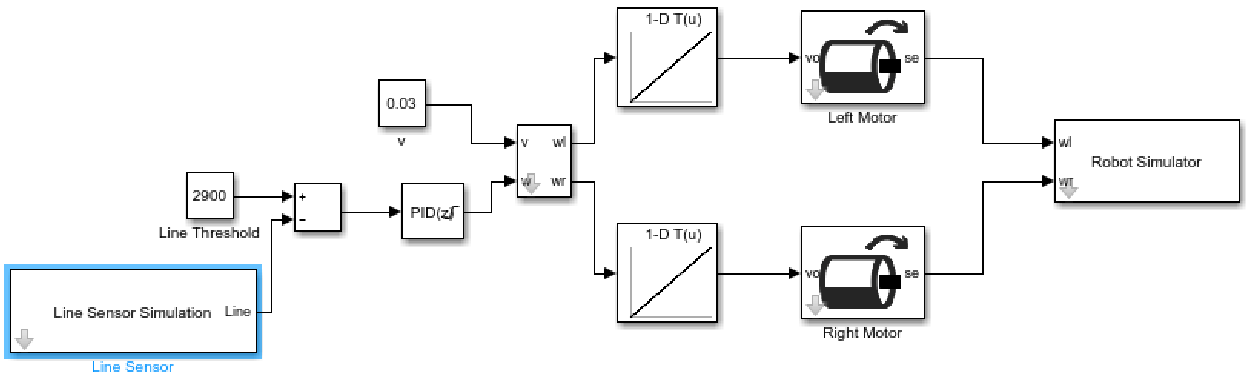
Tuning

In tuning phase, we tried to use Z-N method (S-shape) but it was really difficult to calculate the variables of the transfer function (delay time, time constant, and system gain). So, we tried to tune the P controller manually by try and error and monitoring the behavior using serial monitor and then we used the attached table and the common practice steps to complete the tuning process.

Response	Rise Time	Overshoot	Settling Time	S-S Error
K_P	Decrease	Increase	NT	Decrease
K_I	Decrease	Increase	Increase	Eliminate
K_D	NT	Decrease	Decrease	NT

Simulation

We managed to simulate the response of the robot using mobile robotics training library in Simulink, we also used PID control in the simulation. MATLAB, Simulink files and a short video of the simulation are uploaded.

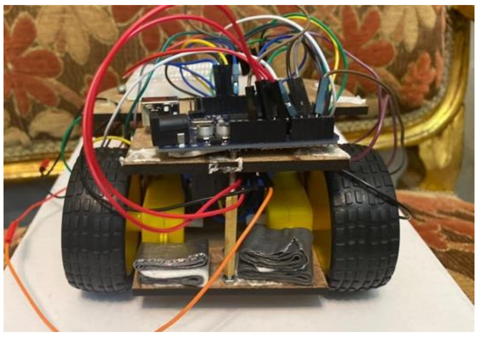


Problems & solutions

We faced problems at different phases of project, and we solved them:

Design wasn't stable because components were concentrated on the front part, so wheels weren't fully touching ground

—> we added weights to the back of the robot for balancing



Response of adding integral constant wasn't clear at the beginning and it was making robot move randomly

—> we changed tuning method and used MATLAB simulation to predict response

Proposed improvement

for better response and easy control:

- QTR reflectance sensor array should be used, it will give more accurate values and will be easier in coding than separate IR sensors.
- PID Arduino library should be used.
- 5 IR sensors should be used to increase the covered area by the sensors hence the range of sensors readings.
- Arduino uno and nano can perform this task with high efficiency, so they are quite good for that task.

Links

- Drive link : [Drive]
- Github link : [Github]