FACULTY
OF SCIENCE
Masaryk University

Workshop on Line Following Mechanisms, Algorithms & Optimization

A brief introduction to PID controller & Differential Drive

Manash
EEE-2K12, Khulna University of Engineering & Technology

Outline for Section 1 I

Outline for Section 1 II

Outline for Section 1 III

How can we detect a line on a surface?

Intro to Line Detection

These are common methods for line detection:

1. LDR (Light Dependent Resistor) Sensor Array
2. IR (Infra-red) Sensor Array
3. Camera

Which is the best one?

Every method has its own advantages as well as trade-offs. We will be discussing that shortly.

General Purpose Method

IR Sensors

Mostly, IR sensors are widely used in building Line Following Robots. Here are some advantages:

Advantages

1. Ambient light barely affects the performance of an IR
2. Easy to find out if an IR transmitter working or not (By using a Camera)
3. Easy to mount
4. Cheap and available

Trade-offs

It also has some minus points.

Disadvantages

1. Different IR sensors on a same array shows different thresholds
2. Soldering IR may change sensitivity
3. Consumes a lot of power
4. Requires uniform DC supply

LDR vs IR

Let the battle begin

Why choosing LDR over IR for LFR is a **BAD** idea?

Since LDR works as a receiver in a LFR, LED has to play its part as a transmitter.

Diffusion of LED light covers a great area which is sometimes good and sometimes bad. For an LFR, it is quite bad.

Radial diffusion of LED will have an effect on LDR sensors. Not from its corresponding LED, but from other ones, and we have to consider about the unnecessary reflection from other LEDs too.

Which will drive the LFR crazy, literally.

LDR vs IR (continues..)

Who won? You decide!

What if I insist using LDR?

Well, if you are that stubborn, use a partition between each LED-LDR TX-RX pair. But remember, it’s a lot easier to buy IR pairs than making partitions.

When you should choose Camera

Getting hands dirty on Digital Image Processing and A.I.

If you are going to make a LFR based on your AI and DIP skill, then using a camera for detecting line is a very good choice.

You'd find out eventually that processing video and image takes a lot of time and processing power which a microcontroller can not afford. You might go for Raspberry Pi or similar microcomputer boards though.

Camera powered LFR will be really slow, it may get a little bit faster with an improved algorithm but it surely will fail to compete with a PD controller based LFR.
For competition, using a camera is not recommended.

Analog or Digital IR?

Always Analog

Before I mention any IR model, you should always go for Analog IR sensors.

Digital IR only shows 1 or 0, and they are hard to calibrate, on the other hand, Analog IR sensors can give analog reading based on your ADC resolution.
Aside Analog and Digital these IR arrays are great for making LFR.

Recommended IR sensors

Pololu QTR-8A Array

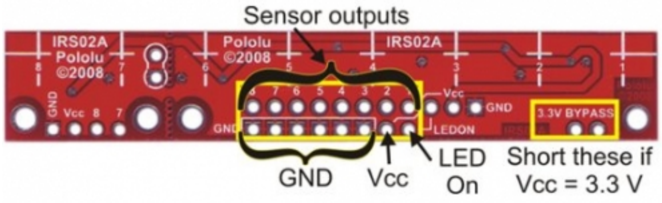
High precision, professional and already packaged as an array. Minus side, header pins have to be soldered by yourself.

TCRT 5000L

TCRT 5000L (L stands for Long leg) comes as a pair of RX-TX, you'd have to make your own array.

Circuit Diagrams

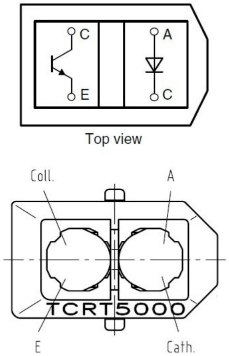
Pololu QTR-8A

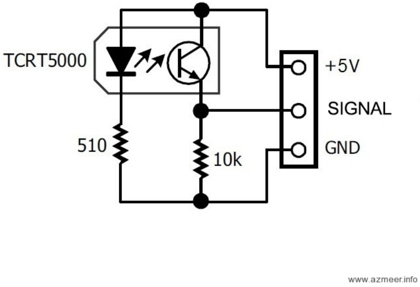


QTR-8x reflectance sensor array with 8x2 connection pins labeled.

Circuit Diagrams

TCRT 5000





What is Threshold of an IR?

And how to calculate

Threshold

It is a value which can be used to distinguish between black and white surface.

Calculation Procedure

1. Measure IR sensor value on white surface
2. Measure IR sensor value on black surface
3. The average value will be the threshold

In formula,

$$Threshold = \frac{White+Black}{2}$$

DC Gear Motors

Yellow DC Gear Motor

DC Gear Motors

Micrometal Gear Motor

DC Gear Motors

Micrometal Gear with Pololu Wheel

Motor Drivers

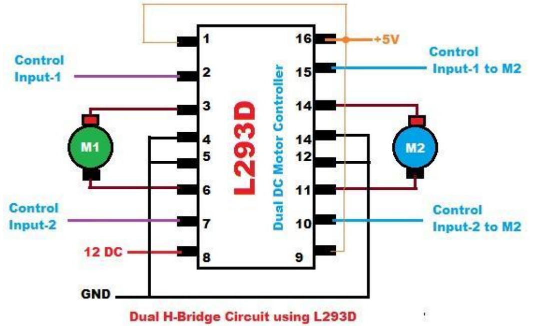
Why do we use Motor Drivers?

1. Motor Driver acts as an intermediate device between a microcontroller, a power supply and the motors.
2. Microcontroller can decide but it can not drive motors because of its very limited power (current and voltage) output.
3. On the other hand, motor driver can provide required current and voltage but can not decide how fast the motor should turn.
4. That’s where PWM comes in, microcontrollers can generate PWM signals which can control speed of motor via motor drivers.

Some Motor Drivers

L298N is better than L293D

L293D Motor Driver

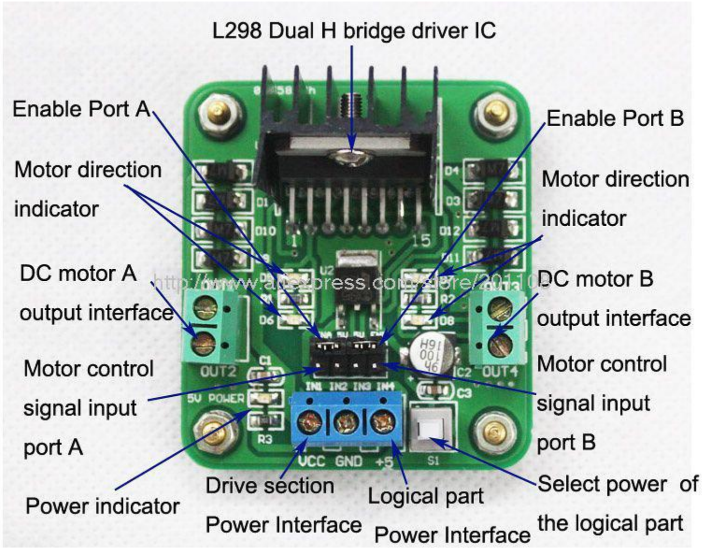


Dual H-Bridge Circuit using L293D

All inputs Low: Motor M1 & M2 = OFF.
Input-1 is High and Input-2 is Low: Motor M1 = Forward Direction.
Input-1 is Low and Input-2 is High: Motor M1 = Backward Direction.
Same Condition to M2.

Some Motor Drivers

L298N Motor Driver Breakout


L298 Dual H bridge driver IC
Enable Port A
Enable Port B
Motor direction indicator
Motor direction indicator
DC motor A output interface
DC motor B output interface
Motor control signal input port A
Motor control signal input port B
Power indicator
Drive section Power Interface
Logical part Power Interface
Select power of the logical part
OUT2
OUT4
IN1 IN2 IN3 IN4
VCC GND +5
5V POWER
http://www.aliexpress.com/store/201108

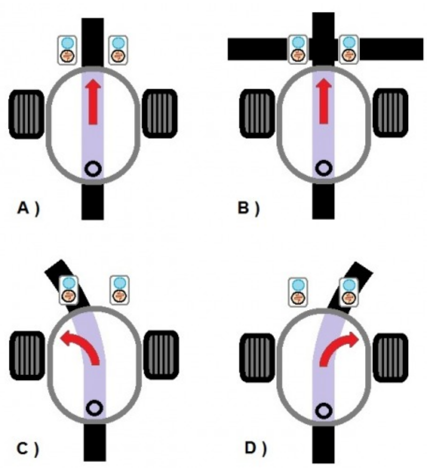
Outline for Section 2 I

Outline for Section 2 II

Outline for Section 2 III

Line Following Algorithms

2 Wheel Drive Locomotion : Differential Drive



Applying Differential Drive

Intro to Differential Drive

Procedure

- Take reading from IR arrays, if front IRs are on the line then full speed ahead
- If robot is slightly moved LEFT from the line, meaning MOST IR is on RIGHT SIDE , then go slight right
- If robot is slightly moved RIGHT from the line, meaning MOST IR is on LEFT SIDE, then go slight left

Improving Differential Drive Algorithm

By Matching Pattern

We can improve basic differential drive algorithm in numerous ways.

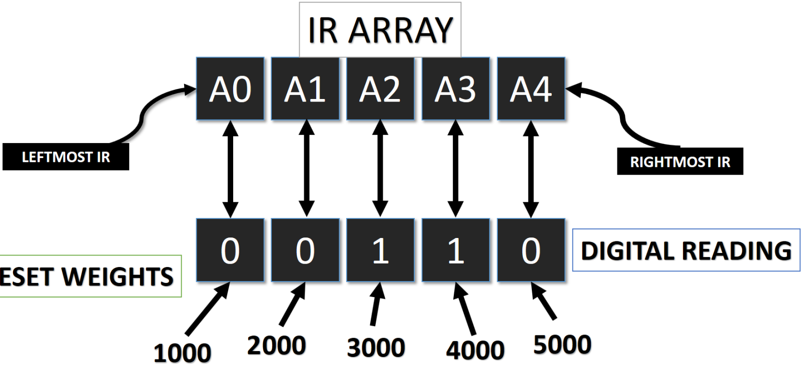
Matching pattern could be one of those.

Following Line by matching pattern

- Take note of some patterns encountered by your robot in binary form in an array [1 means on line, 0 means out of line]
- Experiment with the patterns and find out the required speed to keep the robot on track
- Write some if-else statement for checking those patterns and if a pattern is matched then the robot should use the corresponding speed

Further Improvement

'`readLine`' and '`getPosition`' Method



$$\boldsymbol{Total\ Current\ Weight} = \mathbf{0 \times 1000 + 0 \times 2000 + 1 \times 3000 + 1 \times 4000 + 0 \times 5000 = 7000}$$

$$\text{POSITION} = \frac{Total\ Current\ Weight}{Number\ of\ Active\ Sensors} = \frac{7000}{2} = 3500$$

Implementing ‘`readLine`’ Method

1. Create a global array ‘`digitalReading`’ with the capacity equal to the number of IR sensors used building the robot
2. Create a global integer ‘`activeSensors`’ and set it to 0
3. Create ‘readLine’ function with void input parameter and return type and
   - 3.1 At the beginning set ‘`activeSensors = 0`’
   - 3.2 Use a ‘`for loop`’ to read all the sensors and check the reading with predefined threshold if the ir is on the line or not
   - 3.3 If checked IR is on line then put 1 in the corresponding index of ‘`digitalReading`’ array and increase ‘`activeSensors`’ by 1 else just put 0 (No change in activeSensors)

Implementing ‘`getPosition`’ Method

1. Create ‘`getPosition`’ method with void input parameter and integer as return parameter
2. Call ‘`readLine`’ function
3. Create a ‘`totalWeight`’ variable and set it to 0
4. Create a ‘`for loop`’ with the range of number of total sensor and within the loop
    - 4.1 Update the ‘`totalWeight`’ variable with ’`(i + 1) * 1000 * digitalReading[i]`’ where i is the initialized variable of the ‘`for loop`’
5. return ‘`totalWeight / activeSensors`’

Implementation of ‘readLine’ I

```
#define numberOfSensors 5
#define THRESHOLD 500

int digitalReading[numberOfSensors];
int sensors[] = {0, 1, 2, 3, 4};
int activeSensors = 0;

void readLine(void)
{
  activeSensors = 0;
  for (int i = 0; i < numberOfSensors; i++)
  {
    if (analogRead(sensors[i]) > THRESHOLD)
    {
      activeSensors++;
      digitalReading[i] = 1;
```

Implementation of ‘readLine’ II

```
    }
    else digitalReading[i] = 0;
  }
}
```

Implementation of ‘getPosition’ I

```
int getPosition(void)
{
  readLine();
  int totalWeight = 0;
  for (int i = 0; i < numberOfSensors; i++)
  {
    totalWeight += (i + 1) * 1000 * digitalReading[i];
  }
  return totalWeight / activeSensors;
}
```

Wrapping it all up I

Why and How?

How ‘getPosition’ will help?

1. Now we do not have to consider all possible patterns, just the positional value will be enough
2. We can find out the possible positional values and examine which PWM combinations are required to keep the robot on track
3. ‘getPosition’ will come in handy implementing the PID controller

Wrapping it all up II

Why and How?

Let us consider an example:

```
void differentialDrive(void)
{
  int position = getPosition();
  if (position >= 3500 && position < 4500) forward();
  else if (position >= 2500 && position < 3500) slightLeft();
  else if (position >= 4500 && position < 5000) slightRight();
  else stop();
}
```

Introduction to PID Controller I

Open Loop vs Closed Loop

Open Loop Control System (Differential Drive)

An open-loop controller, also called a non-feedback controller, is a type of controller that computes its input into a system using only the current state and its model of the system.

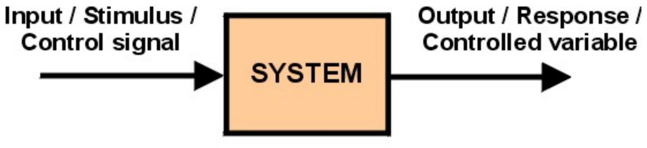

Input / Stimulus / Control signal
SYSTEM
Output / Response / Controlled variable

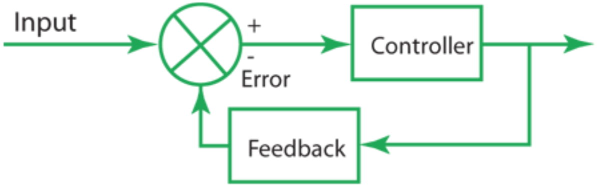
Fig.14 Open loop control system

Closed Loop Control System (PID Controller)

Introduction to PID Controller II

Open Loop vs Closed Loop

A Closed-loop Control System, also known as a feedback control system is a control system which uses the concept of an open loop system as its forward path but has one or more feedback loops (hence its name) or paths between its output and its input.



Key-Concepts I

Set-point, error, overshoot, settling time, oscillation

Set-point

Set point is the desired value, if we want to keep our line follower robot at ‘3500’ positional value then the Set point will be 3500.

Error

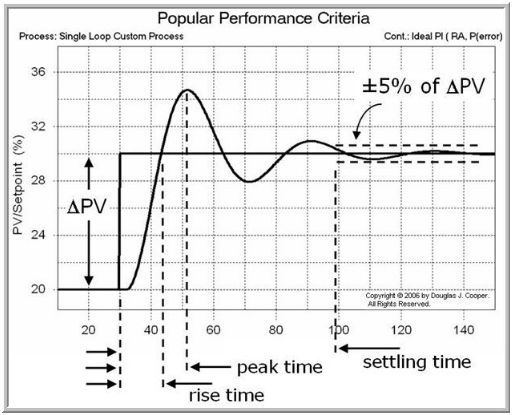
Difference between Set-point and current value. If set-point is 3500 and current value is 4000 then the error is -500.

Settling Time

Key-Concepts II

Set-point, error, overshoot, settling time, oscillation

Settling time is the time required for an output to reach and remain within a given error band following some input.



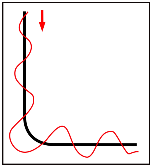
Key-Concepts III

Set-point, error, overshoot, settling time, oscillation

Oscillation

Oscillation is the continuous deviation from set-point within a period of time.

Oscillation in a line follower robot



Tuning

Tuning is the method of acquiring the value of PID constants at which PID controller works as expected.

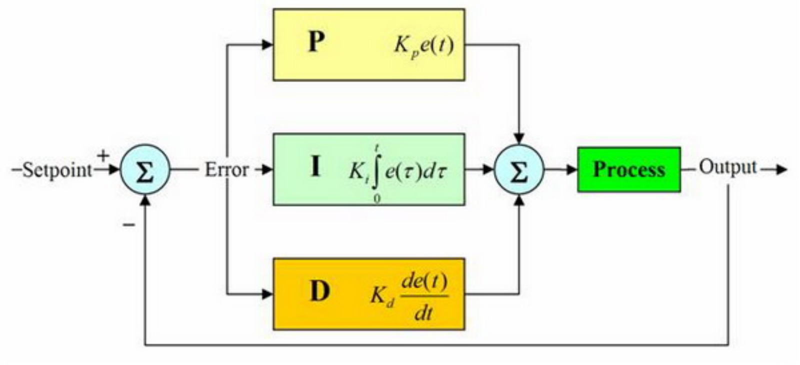
PID Formula

& Controller Diagram

$$PID = K_p e + K_i \int e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

Where, K_p, K_i and K_d are positive.

Controller Diagram



What will PID calculate for LFR?

Perfectly tuned PID controller will calculate how much speed you need to reduce from motors (sign will determine from which motor) to keep the robot on track. Which we have been calculating from our common sense until now.

I/O of PID

Input: Current Positional Value

Output: Calculated speed which will be reduced from the motor (Sign will determine from which motor)

Implementing PID Controller I

Algorithm

1. First of all, you have to determine set point and put the value on a global variable
2. Create these global variables `Kp`, `Ki`, `Kd`, `error`, `totalError`, `power`, `previousError` and set it all to `0`
3. Create ‘`PID`’ function and within the function:
   - Create a local variable ‘`currentPosition`’ and store value of current position by calling ‘`getPosition`’ function
   - Store ‘`error`’ to ‘previousError’
   - Subtract ‘`setPoint`’ from ‘`currentPosition`’ and store it to ‘`error`’
   - Update ‘`totalError`’ by error

Implementing PID Controller II

Algorithm

- Apply PID formula for ‘`power`’, $power = Kp * error + kd * (error - previousError) + (ki * totalError)$
- If the calculated ‘`power`’ is greater than the maximum value of speed (here 255) then set ‘`power`’ equal to the maximum value (power = 255)
- If the calculated ‘`power`’ value is less than the minimum value of speed (here -255) then set ‘`power`’ equal to the minimum value (power = -255)
- If power is negative then subtract speed from left motor (depends) else subtract speed from right motor
- Apply current reduced speed to follow the line

Implementation of PID Controller I

Example Code

```
int maxSpeed = 255;
float error = 0;
float previousError = 0;
float power = 0;
int totalError = 0;

int leftSpeed = 0;
int rightSpeed = 0;

int setPoint = 3500; //Let’s assume set point is 3500

float kp = 0;
float ki = 0;
float kd = 0;
```

Implementation of PID Controller II

Example Code

```
void PIDLineFollow(void)
{
  int currentPosition = getPosition();
  previousError = error;
  error = currentPosition - setPoint;
  totalError += error;
  power = (kp * error) + (kd * (error - previousError)) + (ki *
        totalError);
  if (power > maxSpeed) power = maxSpeed;
  else if (power < -1 * maxSpeed) power = -1 * maxSpeed;

  if (power < 0)
  {
    leftSpeed  = maxSpeed - abs((int)power);
```

Implementation of PID Controller III

Example Code

```
      rightSpeed = maxSpeed;
    }
    else
    {
      leftSpeed = maxSpeed;
      rightSpeed = maxSpeed - int(power);
    }
    forward(leftSpeed, rightSpeed);
  }
```

Tuning I

Tuning of K_p, K_i, K_d

Once you have PID running in your robot, you will probably notice that it still doesn't follow the line properly. It may even perform worse than it did with just proportional! The reason behind this is you haven't tuned the PID routine yet.

PID requires the Kp, Ki and Kd factors to be set to match your robot's characteristics and these values will vary considerably from robot to robot. Unfortunately, there is no easy way to tune PID.

It requires manual trial and error until you get the desired behaviour. There are some basic guidelines that will help reduce the tuning effort.

Tuning II

Tuning of K_p, K_i, K_d

- Start with Kp, Ki and Kd equalling 0 and work with Kp first. Try setting Kp to a value of 1 and observe the robot. The goal is to get the robot to follow the line even if it is very wobbly. If the robot overshoots and loses the line, reduce the Kp value. If the robot cannot navigate a turn or seems sluggish, increase the Kp value.
- Once the robot is able to somewhat follow the line, assign a value of 1 to Kd (skip Ki for the moment). Try increasing this value until you see lesser amount of wobbling.

Tuning III

Tuning of K_p, K_i, K_d

- Once the robot is fairly stable at following the line, assign a value of 0.5 to 1.0 to Ki. If the Ki value is too high, the robot will jerk left and right quickly. If it is too low, you won't see any perceivable difference. Since Integral is cumulative, the Ki value has a significant impact. You may end up adjusting it by .01 increments.
- Once the robot is following the line with good accuracy, you can increase the speed and see if it still is able to follow the line. Speed affects the PID controller and will require retuning as the speed changes.

Tuning IV

Tuning of K_p, K_i, K_d

Tuning Hack

Kp, Ki and Kd may need to be changed frequently until you find a perfect value. Changing value within the code and again uploading it for testing is not always the best solution. Instead you can either use 3 buttons for these three constants or three potentiometers with ADC channel. With each push of a button, a small fractional value will be added to the current value. Same theory can be applied to the potentiometers.

A Simple Code Snippet Explaining Tuning Hack

Tuning V

Tuning of K_p, K_i, K_d

```
#define input(pin) (pinMode(pin, INPUT_PULLUP))
#define input_pot(pin) (pinMode(pin, INPUT))

#define INCR_SIZE 0.01

float Kp = 0;
float Kd = 0;
float Ki = 0;

int buttons[] = {2, 3, 4}; //For Kp, Ki and Kd
int potentiometers[] = {A3, A4, A5}; //Kp, Ki and Kd

void increasePIDByButton(void)
{
```

Tuning VI

Tuning of K_p, K_i, K_d

```
if (digitalRead(buttons[0]) == LOW)
{
   Kp += INCR_SIZE;
   delay(250);
}

if (digitalRead(buttons[1]) == LOW)
{
   Ki += INCR_SIZE;
   delay(250);
}

if (digitalRead(buttons[2]) == LOW)
{
   Kd += INCR_SIZE;
```

Tuning VII

Tuning of K_p, K_i, K_d

```
      delay(250);
    }
}

void increasePIDByPot(void)
{
  Kp = analogRead(potentiometers[0]);
  Ki = analogRead(potentiometers[1]);
  Kd = analogRead(potentiometers[2]);
}


void setup()
{
  for (int i = 0; i < 3; i++)
```

Tuning VIII

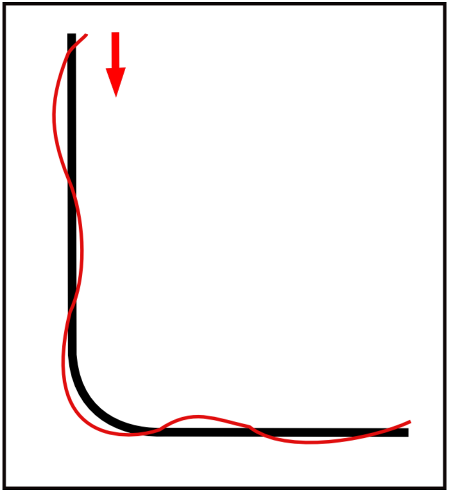
Tuning of K_p, K_i, K_d

```
    {
      input(buttons[i]);
      input_pot(potentiometers[i]);
    }
  }

  void loop()
  {
    increasePIDByButton();
    increasePIDByPot();
    PIDLineFollow();
  }
```

Oscillation after fine tuning!

Smooth!



PID Controller in other robots

Engineering Marvels with PID at heart

1. Self-Balancing Bot complete project from Cornell University : `https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2015/dc686_nn233_hz263/final_project_webpage_v2/dc686_nn233_hz263/`
2. DIY Stewart platform with 6 DOF: `http://bit.ly/1rvYUhP`
3. PID Controlled QuadCopter: `https://github.com/vjaunet/QUADCOPTER_V2`

Applying PID in Wall/Cave Following Robots

DIY

A wall/cave follower robot can be made easily with PID algorithm. Try to make one!

Outline for Section 3 I

Outline for Section 3 II

Outline for Section 3 III

Arduino Hack: Compiling code in Optimization Level 2 I

-O2 is recommended

Without any optimization option, the compiler’s goal is to reduce the cost of compilation and to make debugging produce the expected results.

- -O0 - No optimization, but has the fastest compilation time
- -O1 - Moderate optimization
- -O2 - Full Optimization; generates highly optimized code with slowest compilation time
- -O3 - Full Optimization as ‘-O2’ also uses more aggressive automatic inlining of subprograms.
- -Os (default) - Optimized Space Usage

How to change the Optimization Level I

Works with the latest Arduino IDE

- Open Arduino IDE
- Go to ‘File > Preferences’
- Click on `C:/Users/YourName/AppData/Local/Arduino15/preferences.txt`
- Close the IDE and go to `packages > arduino > hardware > avr > 1.6.X`
- Open `platform.txt`
- Replace all -Os with -O2
- Save the file and close it
- Done!

Where you should use PID and where you should not

PID’ing the right way!

PID Controller works unbelievably well in smooth turns and twisted lines. But it won’t work at hard/ sharp angular turns.

To overcome this problem you may need to use your own method. In general you can follow this guideline.

- Create more functions such as `turnLeft()`, `turnRight()` for detecting those turns and taking appropriate action
- Call those functions in the `loop()` function
- Why are we calling the turn functions in the loop? Because we don’t know when the robot should take turns, it is a good idea to check for turns as the bot keeps going

Implementation of turn taking functions I

- It is always a good idea to have extra IR sensors at the edge of the array which have not been used in PID controller.
- Then you can detect turns by checking the edge IR sensors only

```
void turnLeft(void)
{
    bool takeLeftTurn = false;
  if (analogRead(leftEdgeSensor) > THRESHOLD)
  {
    takeLeftTurn = true;
  }

  if (takeLeftTurn)
```

Implementation of turn taking functions II

```
    {
      while(1)
      {
        forward(0 , rightSpeed); //Robot will turn left
        if ( getPosition() > 0 ) //On White Surface getPosition()
             will return -1 (Because number of sensor will be 0
            and something / 0 results in -1 in C++)
        {
          takeLeftTurn = false;
          break;
        }

      }
    }
  }
```

It is just a basic implementation. You may need to change the code.

Building your own Arduino Library

Work Smart

If you constantly need one piece of code over and over again, it is better to make it permanent by turning it into a library. Library can be header only or header with a source file. Either will do.

Arduino has its own tutorial for making library. You can find it here: `https://www.arduino.cc/en/Hacking/LibraryTutorial`

Note

Some extra tips

- PD Algorithm is recommended for line following robots
- It is better to write a library rather writing same code over and over again
- Try to use OOP Paradigm for building library

Home Work

Do it for yourself!

1. Write a function for Surface Calibration
2. Write a Motor Control Library
3. Write a PID Library
4. Using the above libraries write your own PID Line Follower program

Which topics were not discussed!

But quite important

1. Building a line follower from ground up
2. Making a Maze Solving Robot using Left Hand Rule / Right Hand Rule
3. What to do if one track contains both white line on black surface and black line on white surface

Additional Tools

Line follower simulator

About

The applet simulates a line following robot. It demonstrates how changes in robot's geometry and adjusting PID constants affect the line following performance.

A model of differential driven wheeled robot is implemented. The width of the line sensor, the line sensor position and the wheel gauge (distance between the two wheels) can be adjusted.

Link: `http://www.ostan.cz/LineFollowerSimulator/`

Some of my projects

https://github.com/manashmndl

- PID Line Follower :
  `https://github.com/Electroscholars/LineFollower`
- Cave Follower :
  `https://github.com/manashmndl/CaveFollowerRobot`
- PID Line Follower With Tracer : `https://github.com/manashmndl/LineFollowerWithTracer`
- Artificial Neural Network Driven Line Follower :
  `https://github.com/manashmndl/PDANNLineFollower`

Any Questions?