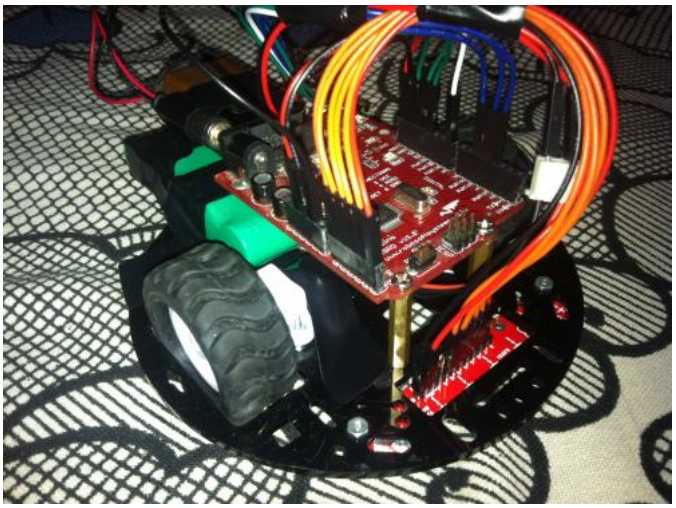
PID Based Line Follower

by Enigmerald Collected by 11 users
Arduino, programming, motors, electronics, wheels, Acrylic Plate, Line Following Sensor, Jumper Wires
Robot

By Enigmerald @ September 23, 2013
Chases down a line pretty fast

- Actuators / output devices: 2 x 30:1 Pololu micro metal motors
- CPU: ATmega168 16MHz Arduino Clone
- Power source: 6.0V Ni-MH for the motors, Duracell 9V for the Arduino
- Programming language: Wiring (Arduino)
- Sensors / input devices: Pololu QTR-8RC Sensor Array
- Target environment: On your kitchen floor or a chart paper with a line on it



PID based Line Following Robot using Arduino

Hello LMRians,

Search

Search

Related blog posts

- PID Tutorials for Line Following

Shout Box

Register and login to chat with other members

User login

Username: *

Password: *

Log in

- Create new account
- Request new password

Log in using OpenID

Recent blog posts

- Arm no.5
- Full Size R2D2 For Sale
- 3d printed lego on my freshly calibrated prusa I3
- Arduino Day 2017 in London
- how to make programs
- Rebates and Chance to Win an Arduino Based Robot Kit on Arduino Day 2017 at RobotShop
- Casual LMR Meeting AFK in Philadelphia (Pennsylvania, USA)
- Second Platform (reserved platform) and Separated Robotic Arm Platform
- Sorry I left for so long...
- Job Offer: RobotShop is looking for a Community Manager

more

Bonus:

- Shops
- Cool Guides
- Other robot-related

Latest weblinks

- Pololu Robotics and Electronics Black Friday/Cyber Monday Sale 2016
- BEAM Robot World
- Project ORRCA
- Spot Mini
- Apple Home Automation
- R2-D2 plans and discussion
- Gloves that translate sign language to speech

This is a PID implemented line follower using an Arduino ATmega168 clone, Pololu QTR-8RC sensor array, a Pololu TB6612FNG Motor Driver, and super fast and zippy pololu micrometal gear motors. (30:1 Gear Ratio, MP version)

PID? Is that the name of a super hero or a mega villain? It's kinda both. If you manage to tune the parameters perfecly, you'll be fascinated by the results that follow, however, at the same time, you're gonna have some sleepless nights if you get it completely wrong. :D

I am no expert here, but what information i have understood about PID, might be helpful for some beginners like me to take your next project up a notch. Basically, when implementing PID, your goal is to achieve a setpoint value. You do that by calculating your error, which is the difference of the setpoint value and your current value. For eg: in this line following robot, our setpoint value is the position of our bot when the middle sensors are exactly above the line it's following. Our current value is obviously the position of our robot at any instant. The error, as mentioned previously is the difference of these two values. Then, by multiplying the error with specific coinstants (Kp, Ki, Kd), the motor speeds can be "predicted" as to smoothly follow a line.

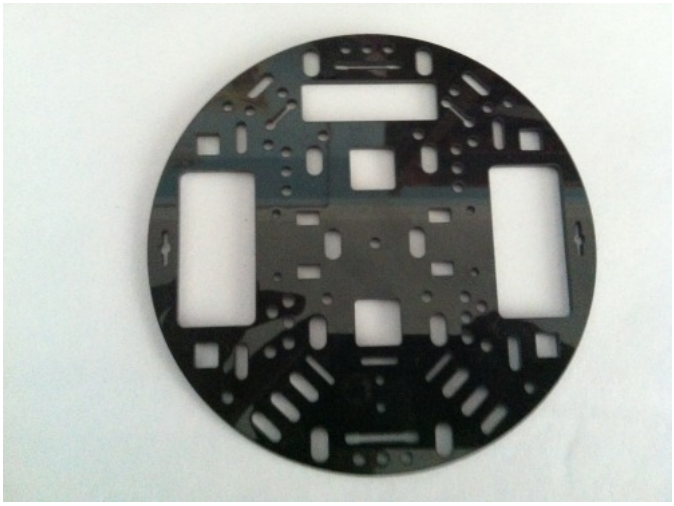
I wrote a small tutorial on implementing PID in a line follower.

In other words, this robot follows a line not only by detecting the turns (yep, that's the basic principle), but also by calculating the error, which is how far it has moved off the track, and then adjusts the motor speeds using the Proportional, Integral and Derivative technique, in short, PID.

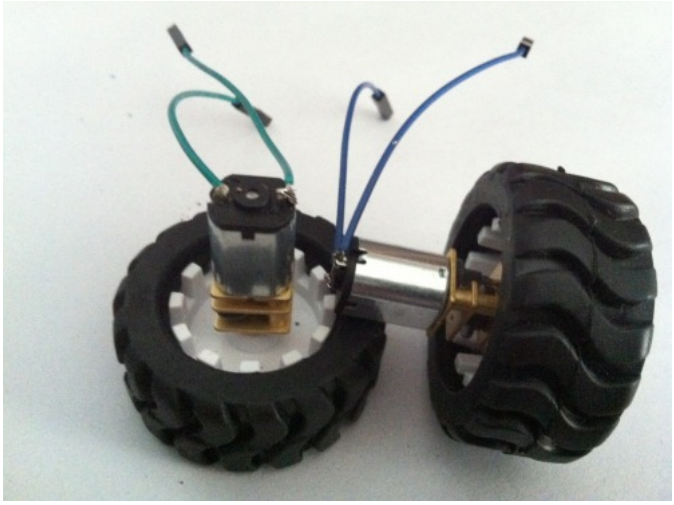
There are many articles and infos across the web about PID, methods of fine-tuning the PID parameters, take a look, and see the results for yourself!!

THE BUILD:

With no 3D printer or CNC machine lurking around my place, and also with an intent of taking part in a national competition, i had to resort to a pre-manufactured simple stable chasis, that could to the job. I used a Pololu circular acrylic plate for the robot body.



Next up, i needed some grippy wheels and some super fast motors to get the most out of my PID routine. Again, i ordered a pair of pololu's popular micrometal gear motors, along with a pair of extended brackets, and a grippy tire set. The motors are rated at 6.0 V and reach speeds of 730 RPM.

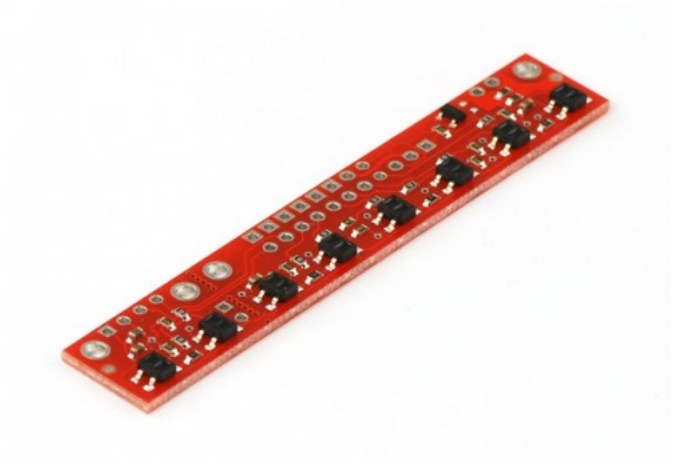


- robot surgeon
- Some LMR bots doing standup and improv comedy...
- Cool use for robotics tech - Artaic

Navigation

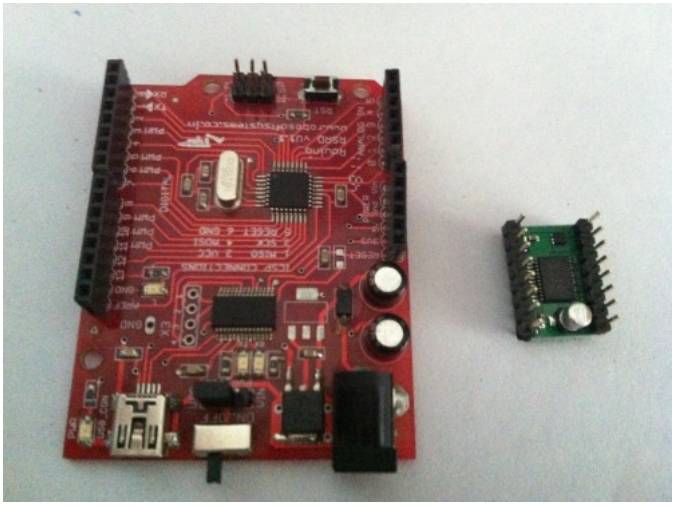
- LMR on Google+
- LMR on Facebook
- LMR on Twitter
- Creative Commons License Idea
- LMR Scrapbook
- User list
- Let's Make Robots World Maker Faire 2016
- Unread posts
- RobotShop Rover Development
- RSS feeds
- Spam Control

For the line tracking part, i used the Pololu QTR-8RC array. It not only detects whether the line is bright or dark, but it intelligently is capable to determine the line position. I am pretty much impressed by this sensor, and with the arduino libraries, it is a nice choice for building a speedy line follower.



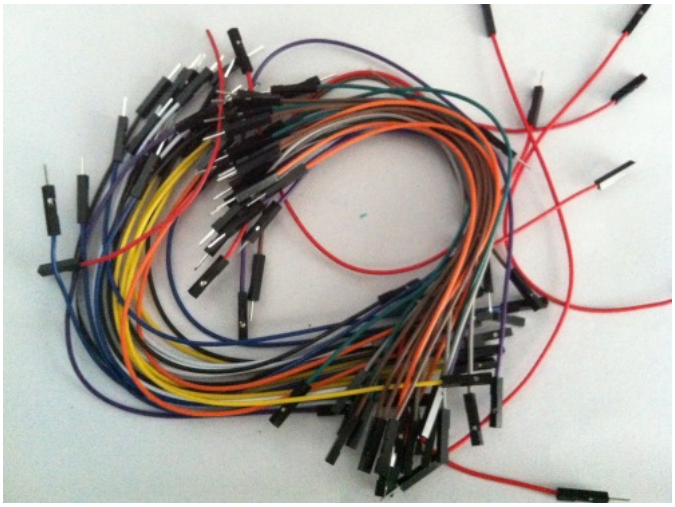
Then, i needed a powerful microcontroller to make the the error calculations fast as possible. Although not that powerful, i used a 16MHz ATmega168 Arduino clone, and it did not disappoint.

To handle turns and change directions in a snap, i needed a motor driver IC that was capable of effectively braking the motors when the PWM signal went low. For this i used the Pololu TB6612FNG Motor Driver. I wrote a little tutorial on controlling the direction and speed of two bi-directional DC motors using this motor driver here.



For powering up my system, i used a Duracell 9V to supply a stable 9V to my Arduino, and as a temporary choice, i used a 6.0 V Ni-MH battery pack, which i soon plan on replacing with a lightweight Li-Po battery.

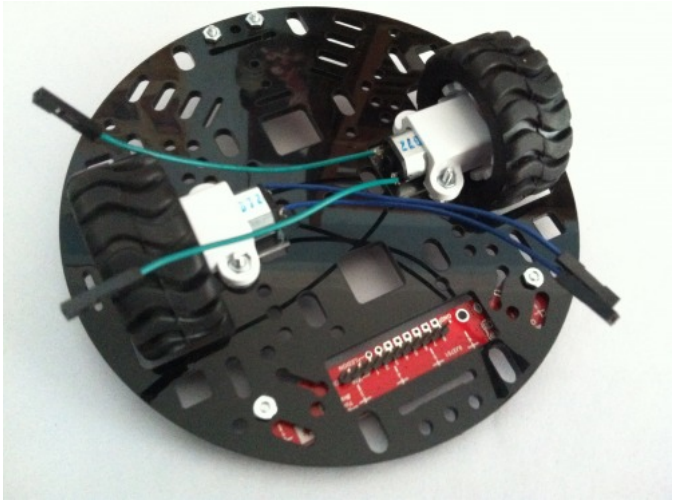
And then, i used some jumper wires for connecting respective pins across the robot.



And finally some black electrical tape to make sure everything is in its place. Loose wires spell trouble!! Have been knocked out a couple of times from line following competitions, you don't want to repeat that mistake!! :D

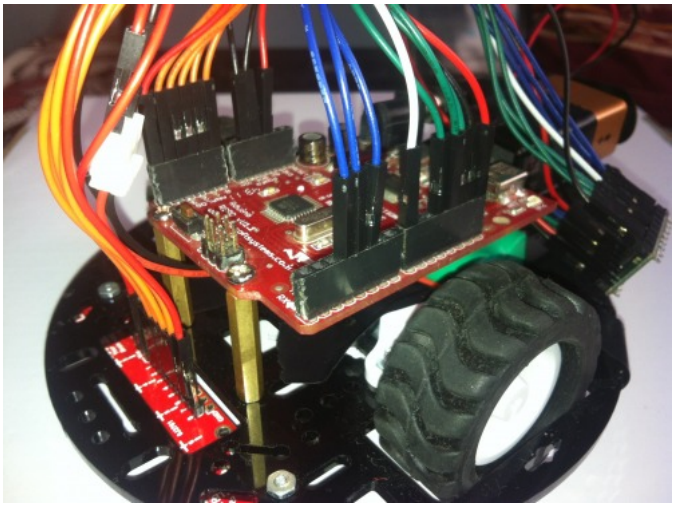
Assembling the chasis is as easy as it gets. The sensors rest at a perfect height from the floor and the motors and wheels are a tight fit. I ordered the parts individually, but more or less, it is pretty much a line following kit. Of course, there are several other mounting options and expansion slots on the acrylic plate, you can hack it to make it something more than just a line follower.

This is what it looks like with the sensors, motors, and wheels mounted on to the plate.

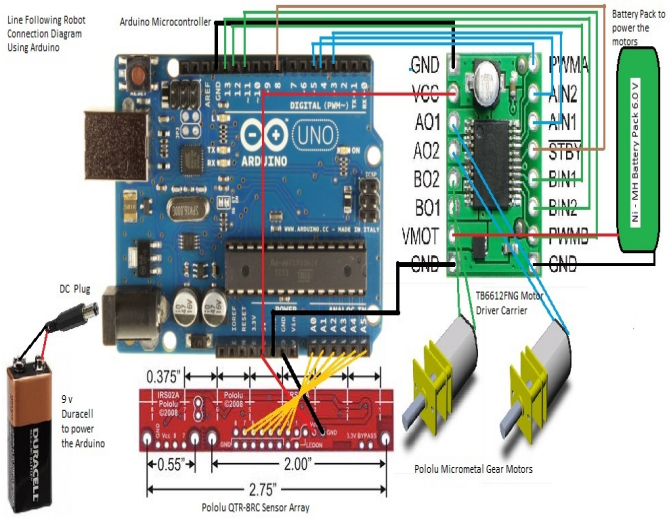


Then i simply mounted the motor driver and the Arduino. (big deal!!!)

And finally added the battery pack at the back, made the connections accordingly. This is what it looked like at last.



And here is a diagramatic representation of the connections to be made, if you're having some trouble connecting. Click on it for a better view.



And then, the reason why it's working, "**DA CODE**" !!! :D

PID Based Line Follower Arduino Code

UPDATE: 1/6/2014:

Thanks to Ladvien, one of the few LMR geniuses around here, I came to know about "Source Code Beautifier". I have removed the boring download link to the code and instead, have pasted the code right here, with some colours and added beauty!!

```
#include <QTRSensors.h>

#define Kp 0 // experiment to determine this, start by something small that just makes your bot follow the line at a slow speed
#define Kd 0 // experiment to determine this, slowly increase the speeds and adjust this value. ( Note: Kp < Kd)
#define rightMaxSpeed 200 // max speed of the robot
#define leftMaxSpeed 200 // max speed of the robot
#define rightBaseSpeed 150 // this is the speed at which the motors should spin when the robot is perfectly on the line
#define leftBaseSpeed 150  // this is the speed at which the motors should spin when the robot is perfectly on the line
#define NUM_SENSORS  6     // number of sensors used
#define TIMEOUT       2500  // waits for 2500 us for sensor outputs to go low
#define EMITTER_PIN   2     // emitter is controlled by digital pin 2

#define rightMotor1 3
#define rightMotor2 4
#define rightMotorPWM 5
#define leftMotor1 12
#define leftMotor2 13
#define leftMotorPWM 11
#define motorPower 8

QTRSensorsRC qtrrc((unsigned char[]) {  14, 15, 16, 17, 18, 19} ,NUM_SENSORS, TIMEOUT, EMITTER_PIN); // sensor connected through analog pins

unsigned int sensorValues[NUM_SENSORS];

void setup()
{
  pinMode(rightMotor1, OUTPUT);
  pinMode(rightMotor2, OUTPUT);
  pinMode(rightMotorPWM, OUTPUT);
  pinMode(leftMotor1, OUTPUT);
  pinMode(leftMotor2, OUTPUT);
  pinMode(leftMotorPWM, OUTPUT);
  pinMode(motorPower, OUTPUT);

  int i;
for (int i = 0; i < 100; i++) // calibrate for sometime by sliding the sensors across the line, or you may use auto-calibration instead

  /* comment this part out for automatic calibration
  if ( i  < 25 || i >= 75 ) // turn to the left and right to expose the sensors to the brightest and darkest readings that may be encountere
     turn_right();
   else
     turn_left(); */
   qtrrc.calibrate();
   delay(20);
wait();
delay(2000); // wait for 2s to position the bot before entering the main loop

    /* comment out for serial printing

    Serial.begin(9600);
    for (int i = 0; i < NUM_SENSORS; i++)
    {
      Serial.print(qtrrc.calibratedMinimumOn[i]);
      Serial.print(' ');
    }
```

```
  Serial.println();

  for (int i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(qtrrc.calibratedMaximumOn[i]);
    Serial.print(' ');
  }
  Serial.println();
  Serial.println();
  */
}

int lastError = 0;

void loop()
{
  unsigned int sensors[6];
  int position = qtrrc.readLine(sensors); // get calibrated readings along with the line position, refer to the QTR Sensors Arduino Library
  int error = position - 2500;

  int motorSpeed = Kp * error + Kd * (error - lastError);
  lastError = error;

  int rightMotorSpeed = rightBaseSpeed + motorSpeed;
  int leftMotorSpeed = leftBaseSpeed - motorSpeed;

    if (rightMotorSpeed > rightMaxSpeed ) rightMotorSpeed = rightMaxSpeed; // prevent the motor from going beyond max speed
  if (leftMotorSpeed > leftMaxSpeed ) leftMotorSpeed = leftMaxSpeed; // prevent the motor from going beyond max speed
  if (rightMotorSpeed < 0) rightMotorSpeed = 0; // keep the motor speed positive
  if (leftMotorSpeed < 0) leftMotorSpeed = 0; // keep the motor speed positive

   {
  digitalWrite(motorPower, HIGH); // move forward with appropriate speeds
  digitalWrite(rightMotor1, HIGH);
  digitalWrite(rightMotor2, LOW);
  analogWrite(rightMotorPWM, rightMotorSpeed);
  digitalWrite(motorPower, HIGH);
  digitalWrite(leftMotor1, HIGH);
  digitalWrite(leftMotor2, LOW);
  analogWrite(leftMotorPWM, leftMotorSpeed);
}
}

void wait(){
    digitalWrite(motorPower, LOW);
}
```

Feel free to roll over the code, and make appropriate changes as per your robot's specifications. An instant transmission to your microcontroller using the magic "copy paste" might not work ;-) . I hope you'll get a fair bit of an idea how line following works using PID. :-)

Although i have mentioned PID, i have just used the Kp and Kd constants in the code. (So, it's pretty much a PD based line follower). When it comes to line following, it is not mandatory to include Ki in the equation, but you can try and see how it goes

You may notice, the Kp, and Kd values are set to 0 in the code. Now to find these constants it's up to you, what's the fun in not finding it? experiment, and gud luck. Note that the Kd value should be much more bigger than the Kp value.

I intend to upgrade this bot, and soon grow it on my own custom chasis, maybe further fine-tune the PID constants, and of course, increase the speed. Probably also make use of encoders, just to let the robot "foresee" the turns and let it know what the speeds at various points should be. I'll also study some more about PID, and hopefully post some more helpful info on it. Till then, that's all guys!! Thanks for reading :-)

Ashim

Comment viewing options

Threaded list - expanded ▼ | Date - oldest first ▼ | 10 comments per page ▼ | Save settings

Select your preferred way to display the comments and click "Save settings" to activate your changes.

By Drakuni @ Mon, 2013-09-23 17:22

nice robot, but pid?

Hello Enigmerald, i have read trough your post and your code, and one thing confuses me:

in the setup you write:
#define Kp 0 // experiment to determine this
#define Kd 0 // experiment to determine this

and your calculation later is:
int motorSpeed = Kp * error + Kd * (error - lastError);

which is like: motorspeed=0*error+0*(error-lastError) that just queals out to 0...

so do you change the kp and kd value manualy somewhere else, or is the kode just a model?

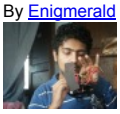
btw nice robot :D

By the_p_engineer @ Wed, 2013-09-25 09:10

Code no longer available

It seems that the ino file you uploaded has been removed. Please check....

By Enigmerald @ Wed, 2013-09-25 10:22



i just shortened the "wait

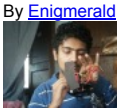
i just shortened the "wait function " in the code , hope the link's working now

By SimpleBotics @ Wed, 2013-09-25 11:21

Nice robot! It's inspired me

Nice robot! It's inspired me to build something like this. Where did you get those stand offs. I suppose the thread is 6/32?

By Enigmerald @ Wed, 2013-09-25 14:39



Thanks Simple ! I got those

Thanks Simple !
I got those standoffs from Dagu, they fit the M3 screw. :)

Sorry, I don't have the exact measures of the threads.

By spdboyz @ Wed, 2013-12-11 02:44

rightMotor1, rightMotor2 & leftMotor1, leftMotor2

What is the use of the following constant variable?

#define rightMotor1 5

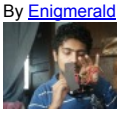
//#define rightMotor2 4

#define leftMotor1 6

//#define leftMotor2 13

Aren't there just 2 motors? What is the 1&2 mean?

By Enigmerald @ Thu, 2013-12-19 12:29



Hello there spdboyz,Yes

Hello there spdboyz,

Yes there are just two motors, but these are dc motors, not servos (which can be programmed directly) , so we need a motor driver IC to control the speed and direction. Motor driver ICs use something called H-Bridges. Google "H-Bridge" for more info on this. I am sure you will find plenty of information.

As for the code, it mentions:

#define rightMotor1 5

#define rightMotor2 4

#define leftMotor1 6

#define leftMotor2 13

The 1s and 2s, simply put, denote the pins of the motor as each motor has two terminals. We control the direction and speed of these motors by supplying a logic voltage from the microcontroller to the motor driver IC. Thus, the 1s and 2s are control lines to which logic voltage is supplied, not to power up the motors, but just to determine the speed and direction.

As you may notice, there are 4 outcomes.

if 1 is high and 2 is low, your motor rotates clockwise.

if 1 is low and 2 is high, it's the reverse, and you motor rotates anti-clockwise.

if both the pins are low, then nothing happens, your motor doesn't rotate.

and if both are high, well, i haven't tried this, but people say that your motor will get toasted. :-D

And programming these motors depends upon the pinouts of the specific motor driver ICs you are using. For example: the L293D and the TB6612FNG have slightly different pinouts. So it all depends on the motor driver you are using.

Google and LMR are your best friends, so good luck.

Ashim

By VerDe @ Sun, 2013-12-22 03:07

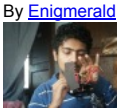
Even if im still a begginer

maybe i can help your question "spdboyz"

HERE YOU GO!!!

https://fbcdn-sphotos-b-a.akamaihd.net/hphotos-ak-ash3/p480x480/1521416_626562530733370_246119875_n.jpg

By Enigmerald @ Tue, 2014-04-15 12:54



thanks Verde. But maybe you

thanks Verde. But maybe you could try posting the whole photo in the comment instead of just the link?

Cheers!

By petrisor23 @ Tue, 2014-04-15 10:26

#define motorPower 8this is

#define motorPower 8

this is the STBY pin from driver?

1 2 3 next › last »