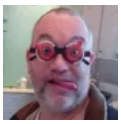
Using motor encoders to control speed

 by OddBot Collected by 38 users
Tip/walkthrough



By OddBot @ October 1, 2013
Provide precision speed control of motors

Attachment	Size
Mini_Driver_Motor_Controller.zip	3.17 KB

This tutorial shows how to connect 2 DC motors with simple encoders to an Arduino for precise speed control and distance measuring. The sample code is written for a DAGU "Mini Driver" and "Simple Motor and Encoders Kit" but will work with any Arduino board with any gearbox / encoder combo.

Recent blog posts
- A brief note from my little sister
- Remote Weather Station
- New Chassis - not the same
- Creating Simple Android Apps with MIT App Inventor 2
- Lobsang- my progress so far.
- First blog entry for PiWars!
- Autonomous navigation Robot
- Modular esp8266 local network
- Spherical Pendulum Robot Ball From Scratch
- LED board (LDR operated)

more

Bonus:
- Basile's 3D Print Club
- Shops
- Cool Guides
- Other robot-related

Latest weblinks
- 2wd with sensor robot simulator base on LUA language :-)
- NEW - JSumo Sumo Robot Projects, Parts, Articles...
- Cooking robot
- Robot parts generator
- My Project Website for Artbyrobot Adam Robot Project
- DFrobot Anniversary
- Getting Started With Basic Circuits Through Minecraft
- THE REVOLUTION IS BEGINNING
- "I've seen enough

When you build a robot using wheels or tracks then you quickly discover that it is impossible to make the robot travel in a straight line. This is because no 2 motors, wheels or gearboxes are identical and brushed DC motors tend to run slightly faster in one direction than another.

You will also discover than your robot will either stall or lurch a bit when trying to perform precise maneuvers at low speeds. This is because the motors need more power to overcome inertia when starting up than they need to maintain a set speed.

The best way to overcome these problems is with encoders. Encoders allow your processor to monitor the actual speed of the wheel and adjust the power to the motor if required to maintain the correct speed. As an added advantage, if you know the circumference of your wheels and resolution of your encoders then you can also measure distance travelled, which is very useful for mapping an environment or navigating by "dead reckoning".

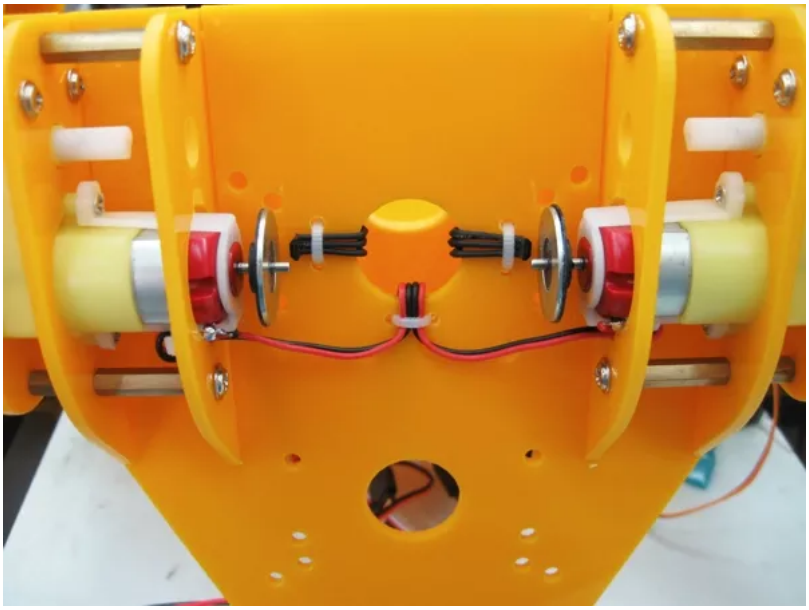
Encoder Types:
Encoders basically come with either 1 or 2 sensors. The encoders that come with 1 sensor can measure speed and distance but cannot determine which direction the motor is spinning. Encoders with 2 sensors (often called quadrature encoders) have 2 sensors that are 90° degrees out of phase. These sensors can determine what direction the motor is spinning as well as measuring speed and distance. In many cases they also offer better resolution.

Typically most encoders in hobby robots use optical sensors however there are other types. In this tutorial I am using hall-effect sensors that detect a magnetic field but the type of sensor used should not matter for this tutorial. The only thing that matters is that the outputs of the sensors swing between digital high and digital low as the wheel turns.

Encoder Resolution:
How precisely you can control your motors depends largely on the resolution of your encoders. Higher resolution generally gives better control although it also depends a bit on the speed of your processor and how you write your code. The encoder's resolution is how many state changes the encoder produces per revolution of the wheel. A state change is when the sensors output changes from low to high or from high to low.

The resolution depends on where the encoder is within the gearbox as well as the number of sensors used and how they are triggered. In my example I only have 1 sensor on each motor. The sensor is triggered by a magnetic disc that has 4 north poles and 4 south poles. This means that for every revolution of the magnetic disc my sensor output will change state 8 times.

Because my encoder disc is on the motor shaft, my encoder resolution needs to be multiplied by the gear ratio of the gearbox. The gearbox has a ratio of 118.5:1 so my final resolution is 8 x 118.5 = 948 state changes per revolution of the wheel. My wheels have a diameter of 65mm so the distance travelled in 1 revolution is 204.2mm. This means I can measure distances as small as 0.2154mm.



The gearboxes I'm using also allow me to put the encoder disc directly on the output shaft. This would then give me an encoder resolution of just 8 state changes per revolution of the wheel and the minimum distance I could measure would be 25.5mm.

Controlling Speed Without Encoders:
When a robot does not have encoders and wants to drive in a straight line the software simply gives both motors the same PWM (**P**ulse **W**idth **M**odulation) and assumes they are both running at the same speed. More advanced programs may multiply the PWM values by a correction factor previously determined from a calibration test but this is still not 100% accurate. With encoders the software specifies the speed the motors should run at and then uses the encoders to ensure the motors maintain that speed.

Using Encoders With Interrupts:
Connect your encoders to the external interrupt pins (usually D2 & D3) on most Arduino controllers. Check here if you're not sure: http://arduino.cc/en/Reference/AttachInterrupt. We use the external interrupts because they are the most efficient and accurate method of responding to the encoder sensors. My hall-effect sensors have an open drain output so I need to enable the internal pullup resistors on these pins. D2 & D3 are inputs by default. When I write a digital 1 to these pins while they are set to input it enables the internal pullup resistors.

Although we could control the speed by counting the number of encoder state changes per second this would mean that the software could be very slow to respond to changes in desired speed. We could count pulses over a shorter time such as 1mS for a faster response but this would require high-resolution encoders and can still loose precision at low speeds.

In this tutorial we will measure the time between state changes in microseconds to determine the speed. This allows the software to

(micro)-Hentai to know where this is going."
- Face Recognition Module

Navigation
- LMR on Google+
- LMR on Facebook
- LMR on Flicker
- LMR on Twitter
- LMR Scrapbook
- User list
- Unread posts
- RSS feeds
- Spam Control

respond much quicker as it can measure the wheel speed after 2 state changes. For this method we need to know what is the maximum number of state changes your encoder will generate at full speed. From this we can calculate the time in µS between pulses that want to achieve a set speed.

Encoder ISR's:
When an interrupt occurs the processor stops what it is doing and jumps to an ISR (**I**nterrupt **S**ervice **R**outine). The ISR should be as small as possible to minimize any chance of it interfering with other interrupts. Some functions such as Servo control can be affected if the ISR is too big.

In my sample code the ISR only measures the time in µS since it was last called, increments the counter used for measuring distance and then sets a flag that tells the code to call the MotorControl() function where the rest of the calculations are done. It will not matter if the rest of the calculations take a little longer to complete because the time between state changes was accurately measured in the ISR.

In reality the timing accuracy of the micros() function has a maximum resolution of 4µS so the time could be out by as much as 3µS but this will be close enough for our purposes. Momentum limits how quickly the motor can change speed and averages out these small inaccuracies.

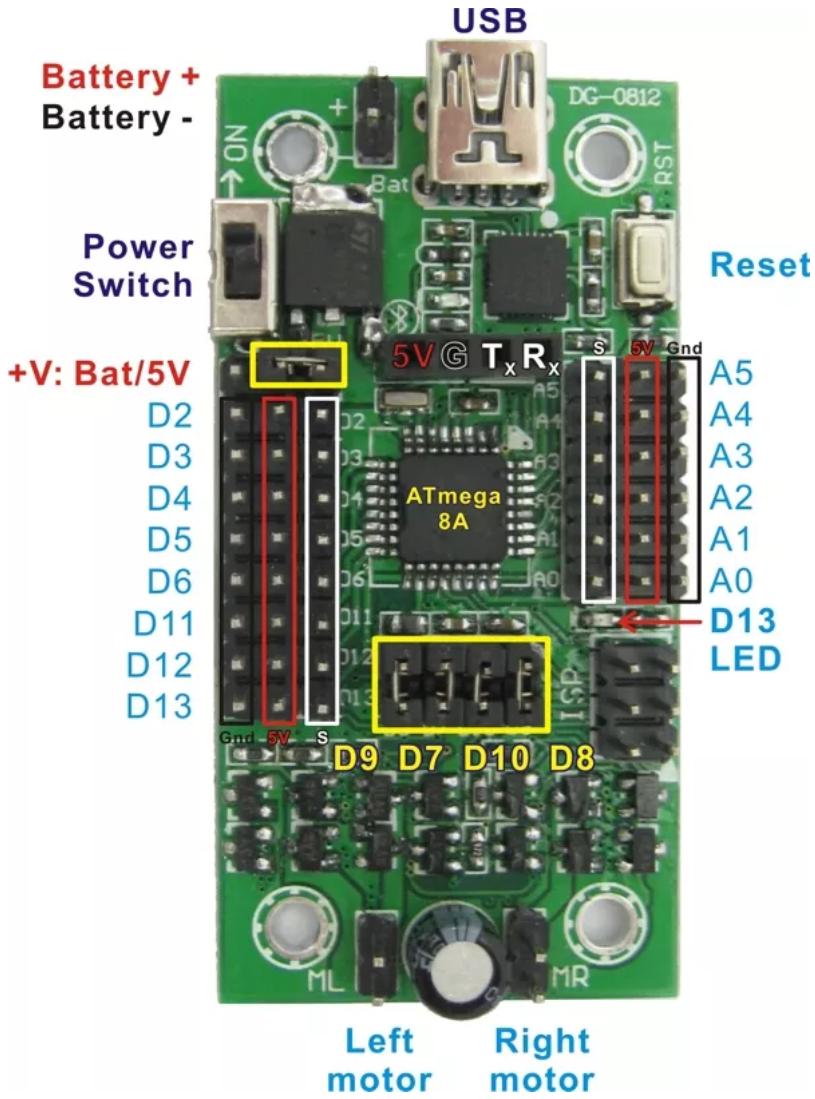
Maximum State Changes:
In my example, I measured the time it took for my wheel to complete 1 revolution (no load) with a 6V supply and determined the speed to be approximately 135rpm. My gearbox has a ratio of 118.5:1 so this means my motor speed is 15997.5rpm, which is about right for this type of motor running at 6V.

So round it up to 16000rpm and my motor shaft spins 16000 / 60 = 266.66 revolutions per second. My encoder gives 8 state changes per revolution so my maximum possible number of state changes per second is 266.66 x 8 = 2133.33. As this was with no load on the motor I'm going to round it down to 1900 which is about 89%.

You may need to experiment with this figure a bit. If it is higher than the slowest motor can achieve under load then the robot may still not achieve a straight line at full speed. If you set this value lower then you guarantee both motors can achieve this speed, even under load but you may limit your maximum speed. I suggest starting at around 90% of maximum speed.

This value is defined in my sample code as the constant "maxspeed" but in my sample code I use a variable called "bestspeed" that has been adjusted to allow for changes in battery voltage. If you're using a battery where the voltage drops as it discharges (e.g. Alkaline or LiPo) then you may want to use a voltage monitor to tweak this value as the battery voltage drops. I prefer NiMh batteries because their voltage remains constant for about 90% of their discharge cycle.

Correcting for Battery Voltage:
In my example I am using a DAGU Mini Driver, which is the equivalent to an Arduino NG or older w/ ATmega8. This low cost controller has a dual "H" bridge rated at 2.5A per motor built in and a battery monitor on A7. My sample code checks the battery voltage every 5 seconds and adjust the variable "bestspeed" so that as the battery voltage drops the maximum possible speed is also reduced.



As power is equal to voltage x current and the current is proportional to the voltage this suggest that when the battery voltage is halved, the maximum current draw should also be halve and therefore the power would be ½ x ½ = ¼. This formula is not perfect as it does not allow for gearbox friction and the motor is not a pure resistor but it should be accurate enough for this application.

Now my bestspeed = maxspeed x (batvolt / maxvolt)²

This should ensure that as the battery voltage drops the best speed should remain obtainable allowing the robot to maintain a straight line.

Converting State Changes to Microseconds:
So now we have a value for maximum state changes per second and we have the number of microseconds between state changes. We need to put this information to work. The reason we bothered with the maximum number of state changes per second is it gives us a linear value to calculate our power from. The time between state changes in µS however is a logarithmic scale as you can see in the table below.

Power		SC/sec		µS/SC
100%	=	1900	=	526µS
90%	=	1710	=	585µS
80%	=	1520	=	658µS
70%	=	1330	=	751µS
60%	=	1140	=	877µS
50%	=	950	=	1053µS
40%	=	760	=	1316µS
30%	=	570	=	1754µS
20%	=	380	=	2631µS
10%	=	190	=	5263µS

In my sample code the desired speed is expressed as a value from -100% to +100% and the desired number of state changes is a fraction of the maxspeed value. Negative values indicate reverse direction.

State changes per second = abs(desired speed) / 100 * bestspeed.

Therefore µS between state changes = 1 / (abs(desired speed) / 100 * bestspeed) * 1000000

As the Arduino IDE processes the formula strictly from left to right a more accurate result is given by re-arranging the formula to: 1000000/abs(desired speed)*bestspeed/100.

Correcting the Speed:
Now that we know how many µS we should be getting between state changes if the speed was correct we can now compare this to the

actual time between state changes.

To ensure smooth motor control my sample code calls the MotorControl() function once every mS. When the motor is stopped the encoders do not generate any information to work from so my code jumpstarts the motor speed a small amount if the encoder has not turned after 20mS. If the motor is not supposed to turn then the control code will quickly reduce the power back to 0 again.

Using the Sample Code:
The sample code is pretty straightforward. You can add your own code for sensors etc. very easily. The main loop has a timer that calls the motor control routine once every mS. You can change this but it will affect how quickly the motors respond to change.

The ISR's and the motor control code are in their own separate tabs to keep the code neat and easy to read.

If you are using different geared motors or different encoders then you will need to change the maxspeed constant. You will also need to change the maxvolt constant to suit your battery or if you do not have a battery monitor, just eliminate the 3 lines of code in the main loop where the battery voltage is read and a new bestspeed value is calculated.

Note: the sample code was written in Arduino 1.04 IDE. I cannot guarantee it will work with other versions of the IDE because the Arduino IDE is not always backward compatible.

Good luck and enjoy!

Comment viewing options

Threaded list - expanded ▼ | Date - newest first ▼ | 10 comments per page ▼ | Save settings
Select your preferred way to display the comments and click "Save settings" to activate your changes.

By Thorn438 @ Tue, 2015-03-31 02:12

What chassis are you using?

Login or register to post comments

OddBot, thanks for the fantastic tutorial! Can you tell me what chassis you are using?

By xlozamor @ Wed, 2014-04-23 02:49

Dagu Rover 5 Encoders

Login or register to post comments

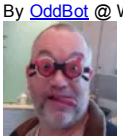
Hello OddBot,

I recently purchased a Dagu Rover 5 along with the 4 Motor Controller and Arduino Uno. I am powering the motors with six AA NiMH batteries, and the arduion Uno with a 9V battery. I have been trying to use your code to get the rover to go straight, but I must be doing something wrong. This is my first time using the Uno, but I have had some experience with coding and ISR.

Since I am using the NiMH batteries, I do not have to monitor the battery voltage, correct? Also when I was trying to determine my value for maxspeed this is what I did. I first estimated that I have about 89 revolutions per minute. (Observed the wheels for a certain amount of time to get this value) I then multiplied this by my gearbox ratio, 86.8:1, giving me 7725 rpm (rotations per minute) which is 129 rps (rotations per second). I then multiplied 129rps by 1000/3 state changes/ wheel rotations. And finally multiplied by .89 to account for load. My final value was 3800. Did I derive my value incorrectly? I have tried changing this value, but have still not been able to get my rover going straight. Any ideas?

I have also tried to just monitor the lcount and rcount variables. I kept my rpwm constant, and tried to change lpwm depending on the difference between lcount and rcount. Either increasing or decreasing lpwm. This has also not gotten my rover going straight.

I would appreciate any help/advice you have, and am willing to post my code if you are you interested. Thank you for your help.

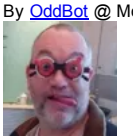
> By OddBot @ Wed, 2014-04-23 15:38
>
> 
>
> **This post was some of my**
>
> Login or register to post comments
>
> This post was some of my earlier experimentation with encoders. I have since written another tutorial with code specific to the Rover 5 chassis. http://letsmakerobots.com/node/39098
>
> If you have a look at my robot OB1, the latest version of the code converts a Mini Driver into an I2C motor driver that is quite accurate. http://letsmakerobots.com/node/39492
>
> > By Frankie @ Sun, 2015-03-29 19:29

hello

Login or register to post comments

i dont see the code for this tutorial sir,good read tho

By OddBot @ Mon, 2015-03-30 10:26



See "Attached files" just

Login or register to post comments

See "Attached files" just under the first photo. This is an old entry. See the later version here: http://letsmakerobots.com/node/39098

The result of my experimenting can be found here: http://letsmakerobots.com/content/smart-motor-driver-reads-encoders-control-speed-and-measure-distance

By Emarte @ Wed, 2014-04-23 04:58

Maybe you need to calibrate your robot

Login or register to post comments

Dear xlozamor, in practice robots need to be calibrated in order for you to get them to go straight, what seems to be a simple task.

This is due to small diference in tires size, chasis, motor etc. Any difference will make your robot not to go the way you want.

Have a look at this http://www.dprg.org/articles/2009-02a/

The results are amazing!!

Good Luck

Edwin

By kas @ Mon, 2014-02-24 20:59

Fluctuating rpulse

Login or register to post comments

Hi OddBot,

I just got a 2 motors/2 encoders Rover5, together with the dedicated 4 channel motor driver
Both are nice pieces of equipment, at a very acceptable price

I tried your code and it works well with no PID control
When I try to print rpulse, I get _very_ fluctuating values, ranging from 1200 to 2240, for the same fixed speed:

1236
2056
1332
1332
2240
2240
1440
1440
1916
1916
1692
1792
1792
2068
2068
1204
1204
2376
1240
1240
1240
2340
1272
1272
2040
2040

```
1756
1724
1724
1876
1340
```

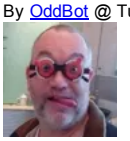
I use a stripped down version (one motor only) at fixed max speed (PWM=255), no code added, except lines marked "<< change":

```
#define rmencpin    3                          //  D3 is the pin for external interrupt 1      encod_R
#define rmdirpin    8                          //  D8  right motor direction                  DIR_R
#define rmpwmpin  9    //10   //<< change     //  D10 right motor PWM                         PWM_R

volatile unsigned int rpulse=100000;          // width of left and right encoder pulses in uS
unsigned long mtime;                          // "no go" timer used to re-start motors when both motors stopped
volatile unsigned long rtime;                 // remembers time of right encoders last state change in uS

void setup()  {
  Serial.begin(115200);
  pinMode(rmdirpin,OUTPUT);
  digitalWrite(rmencpin,1);                   // enable pullup resistor for right encoder sensor
  attachInterrupt(1,Rencoder,CHANGE);         // trigger right encoder ISR on state change
  digitalWrite(rmdirpin,true);                // set direction of right motor
  analogWrite(rmpwmpin,250);  //<< change              // speed for right motor}
}

void loop()  {
  unsigned int temp;                          // temporary variable for calculationa
  if(micros()-mtime>999)    {                 // Call motor control function every mS
    mtime=micros();                           // reset motor timer
    Serial.println(rpulse);  //<< change
  }
}

void Rencoder()  {
  rpulse=micros()-rtime;                      // time between last state change and this state change
  rtime=micros();                             // update ltime with time of most recent state change

}
```

I cant beleave the actual RPM varies that much
Can you reproduce those data ??
What am I doing wrong ??

Thanks for this very interesting contribution

By OddBot @ Tue, 2014-02-25 16:42



Yes I also got that

Login or register to post comments

Yes I also got that fluctuation but it is not because the motor's speed is changing so quickly.

These days the Rover 5 uses a magnetic encoder because it proved more reliable than the earlier optical encoders however unlike a printed black and white pattern, the size and strength of each magnetic field is not exactly the same. This causes the fluctuation.

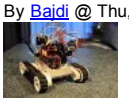
Fortunately the momentum of the motor's rotor, gears and wheels or tracks all work as a filter to average the results of the computer trying to compensate for the variations.

In fact, if you were to print the PWM values instead of the time values you will see that the fluctuation is much smaller.

Note: this post covers my earliest experiments with motor speed control using encoder feedback. My code continues to evolve with the code for OB1's motor controller being the latest version for the Rover 5.

My newest robot, Scamper has a setup closer to the one in this post but the code seems to work much better.

By Bajdi @ Thu, 2013-10-03 19:04



Thank for sharing the

Login or register to post comments

Thank for sharing the interesing code :) Will have to take a good look at it. There is also a good Arduino encoder library: http://www.pjrc.com/teensy/td_libs_Encoder.html

I actually bought a Dagu encoder kit from Rocket Brand studios some time ago. But haven't used it yet... http://www.bajdi.com/?attachment_id=1028

By OddBot @ Fri, 2013-10-04 02:46

I realise there are

Login or register to post comments

I realise there are libraries about but you don't learn anything by just installing a library. I wrote the code from scratch so I could experiment with the encoders and teach myself something.

Since I was teaching myself I decided to write a tutorial and teach others at the same time. There are other methods that could be used such as using timer interrupts but I wanted the code to use standard Arduino commands that noobs can understand.

Note: The picture above is of a new kit that uses motors with extended shafts. This gives the encoders much better resolution.

1 2 next › last »

ALL LMR ARE BELONG TO US!
Let's make robots!