Drive Motor Integration

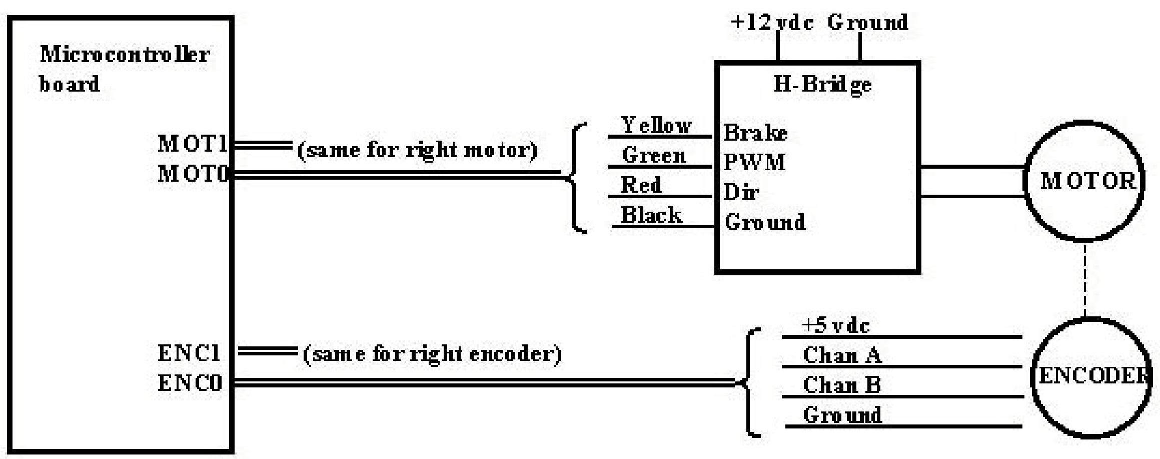
Alex Brown rbirac@cox.net ,leafproject.org 1/12-07

This document covers rev 0 to rev 3 microcontroller boards.

This procedure shows how to hook up two drive motors to a Leaf robot and calibrate the drive parameters.

The microcontroller board can be configured to drive either a convention sign-magnitude PWM h-bridge or a motor controller which accepts RC type commands. To select RC operation, two jumpers must be installed on the microcontroller board as described elsewhere.

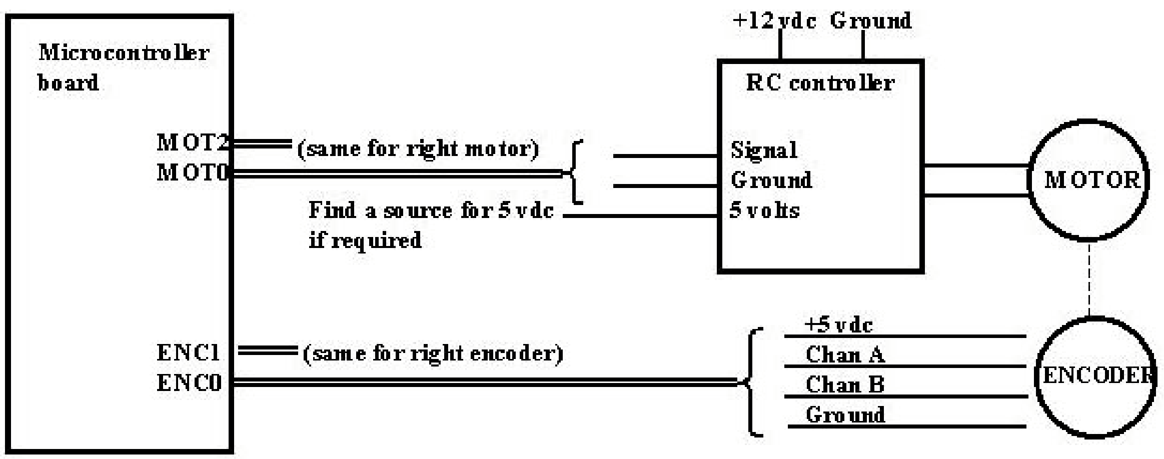
Wiring for sign-magnitude PWM:



Notes: Channel A & B encoder wiring may be reversed depending on your installation
For rev 0 microcontroller board without mod 003 installed, change ENC1 to ENC2

Note: MOT0 and ENC0 go to the left side motor, and MOT1 and ENC1 to the right side motor for H-bridge operation.

Wiring for RC type controllers:



Notes: Channel A & B encoder wiring may be reversed depending on your installation
For rev 0 microcontroller board without mod 003 installed, change ENC1 to ENC2

Only two wires from the MOT0 and MOT2 jacks are used. The ground wire which goes to ground on the motor controller, and the PWM wire which goes to the signal input on the motor controller. If the motor controller requires 5 volts on the control connector, please find a way to supply it yourself.

Notes:

For microcontroller Rev 0 boards without mod 003 installed, all references to the ENC1 jack should be interpreted as the ENC2 jack.

Wiring of the motor drive wires and the encoder channels must be compatible with the polarity conventions of the microcontroller board. This is most easily determined by experimentation.

All the numerical parameters necessary to set up the motors are in a configuration file in the laptop called Microcontroller.ini. These parameters can be edited using a simple text editor such as Notepad. However, in order to get your changes to take effect, it is important to understand how they are transferred from the config file to the microcontroller.

When the Nav and Control program on the laptop is started, it reads the config file and stores the contents in memory. It then transfers the information to the microcontroller over the USB bus.

The laptop monitors the microcontroller data stream to make sure the config data has been downloaded. If the microcontroller is reset and loses its config data, the laptop will resend the config data as stored in MEMORY; it does not reread the config file. E.g., if the laptop and the microcontroller are running, and you make a change to the config file and just reset the microcontroller, it will NOT get your change. You have to stop and restart the Nav and Control program. In this case, you do not have to reset the microcontroller since Nav and Control will download the new data at startup and overwrite the previous data in the micro.

Before starting calibration, you should set the contents of Microcontroller.ini as follows:

```
; MicroController.ini
;   constants for Rocky   (change to your robot's name)


[Odometry]
        odomLnum      = 10000
        odomLdenom    = 10000
        odomRnum      = 10000
        odomRdenom    = 10000

[Motors]
        PWMbaseL      = 0
        KvL           = 100
        KaccL         = 0
        KdistL        = 0
        PWMbaseR      = 0
        KvR           = 100
        KaccR         = 0
        KdistR        = 0
        PWMmax        = 100
        RCoffsetL     = 0
        RCoffsetR     = 0
        MotorType     = 0                 0 = H-bridge, 1 = RC motor driver

[Steering]
        PivotConv   = 3000
```

IMPORTANT: Set MotorType above to 0 or 1 as appropriate for your setup.

ENCODER WIRING:

The microcontroller reads the encoders and computes the distance each wheel has traveled and reports the computed distances as the DistActL and DistActR signals (DistanceActual). These two signals are available directly from the microcontroller over the serial bus or may be read from the laptop.

There is just one way to see the DistAct signals that is currently built in. The microcontroller has a serial bus port (yes, the same one you program it from). This port is capable of outputting a continuous data stream which includes the DistAct signals. In order to read it, hook up a terminal and set the baud rate to 115K.

Then, using a copy of Nav and Control.exe posted in April 2006 or later, with the microcontroller and Nav and Control running, select “Micro” from the top menu bar and select “1 Datalog”. This should cause the micro to start printing out data to the terminal. The beginning of the printout lists what each parameter is. Each line of data is printed out 61 times per second, so it can be a bit hard to read. Probably, just two parameters will be changing when you rotate a wheel, so you can find them that way.

While monitoring the DistActL and R signals, rotate the left drive wheel in the forward direction. The DistActL signal should increase in value smoothly as the wheel is rotated. The DistActR signal should be unchanged.

Repeat this test rotating the wheel backwards. The DistActL signal should decrease (move in the negative direction).

Troubleshooting:

If the DistActL signal decreases when the wheel is rotated forward and increases for backward rotation, the encoder polarity is backwards. This can be corrected by reversing the wiring of the channel A and B signal wires from the encoder.

If the signal does not change:
Verify there is +5 vdc on the power wire to the encoder and ground on the appropriate wire.

Verify there is a signal on the wire entering the board on the channel A wire. The expected signal is a square wave which changes between 5 vdc and ground as the wheel is rotated.

If the signal always increases, or always decreases, regardless of direction of motion, verify that there is a similar signal on the channel B wire to the microcontroller board.

Note the DistActL reading and rotate the wheel a full revolution. determine how many counts DistActL changes for a full revolution.

___________ Left encoder clicks per revolution.

Determine the circumference of the wheel in millimeters.

___________ Left wheel circumference in mm.

Determine the number of clicks per mm. (clicks per revolution) / circumference

___________ Left encoder resolution (clicks/mm)

Hopefully this is a fairly high number, greater than 1 at a minimum. 20 to 40 is good.

Repeat for right wheel:

While monitoring the DistActL and R signals, rotate the right drive wheel in the forward direction. The DistActR signal should increase in value smoothly as the wheel is rotated. The DistActL signal should be unchanged.

Repeat this test rotating the wheel backwards. The DistActR signal should decrease (move in the negative direction).

Note the DistActR reading and rotate the wheel a full revolution. determine how many counts DistActR changes for a full revolution.

___________ Right encoder clicks per revolution.

Determine the circumference of the wheel in millimeters.

___________ Right wheel circumference in mm.

Determine the number of clicks per mm. (clicks per revolution) / circumference

___________ Right encoder resolution (clicks/mm)

Initial calibration:

The Microcontroller.ini file on the laptop contains four numbers which are used to calibrate the odometer counts. This initial calibration will be based on the numbers determined above and is a rough estimate. Final calibration will be performed later.

For each wheel, there are two calibration numbers. E.g. for the left wheel, there are odomLnum and odomLdenom, which are the numerator and denominator of a constant multiplier which converts clicks to millimeters.

These two numbers are usually set to 10000 each (a multiplier of 1.0) which would be correct if your encoder just happened to supply exactly 1 click per millimeter.

The following assumes your encoder has more than 1 click per millimeter.

For both the left and right wheels, divide 10000 by the encoder resolution. Edit the Microcontroller.ini file to make odomLnum and odomRnum equal to the numbers calculated. (numbers should be rounded off to the nearest integer). If your two drive wheel assemblies are the same, these numbers are probably almost the same.

Restart the laptop Nav and Control program to load the new settings into the microcontroller.

Rotate both the wheels forward by 1 revolution. Each DistAct signal should increase by the circumference of the wheel in millimeters. (a little error doesn't matter much, this is just a rough number)

___________ Left wheel.

___________ Right wheel

Motor drive setup.

Note: the RC motor drive setup has one additional parameter for each motor. They are called RCoffsetL and RcoffsetR. Motor drivers are supposed to command a stop when receiving a signal of 1.5 msec. If your driver commands the motor to move (slowly, I hope) at 1.5 msec., it is possible to correct this situation by giving RCoffsetL/R positive or negative values to get the motor to stop. One unit of these values represents one percent of PWM. Other than this, the procedure should be about the same. RC motor controller usually have a deadband around 1.5 milliseconds. It might be good to vary these setting to see when the motor starts moving in each direction and use a final setting which is in the middle of the deadband.

IMPORTANT: For this test, put the robot up on a stand so the wheels can rotate freely!

The Microcontroller.ini file has 4 parameters for each motor and a single parameter for both motors to set the maximum PWM duty cycle.

To verify motor drive direction, set the parameters as follows:

```
[Motors]
PWMbaseL = 0
KvL = 100
KaccL = 0
KdistL = 0
(same for right motor)
```

Restart the laptop program to download the new .ini values (if you had to change them).

We will use the Lisp program to send a calibrated motor drive signal.

Open the Leaf.lisp file in your Lispworks compiler window. Use Edit/Find to search for “forward”. This should get you to a line that starts “(defun forward(……..”. This defines a function which runs when you tell your robot to go forward either by speaking or by typing in “(forward)” in the listener window.

The next line reads as follows and defines the command message which is sent to the microcontroller via the Nav and Control program.

```
(SendCommand 0 1 400 250 millimeters 0 0 0 0 0 0 0 0))
```

Revise the line above to change “millimeters” to “4000”. This make the command for the robot to move forward at an acceleration of 400 mm./sec^2 and at a maximum speed of 250 mm/sec for a distance of 4000 millimeters. You can change the number 4000 to any other distance if it is more convenient.

Now, SAVE the file and then load LeafInterfaces as usual and compile and load the new Leaf.lisp file.

With the microcontroller and Nav and Control running, you should now be able to enter “(forward)” in the Lisp listener window and the motors should start to turn in the forward direction.

With a commanded speed of 250 and a KvL & KvR of 100, the motor should be receiving about 25% of maximum PWM.

Verify that each wheel drives in the forward direction. The wheel should turn for about 16 seconds and then stop. The wheels should be turning at a moderate speed. If they are going way too fast or too slow, change the value of Kv (lower to go slower)

Troubleshooting:

If a wheel rotates in the reverse direction:
 If you are using an H-bridge,
  verify that the direction signal from the microcontroller to the H-bridge is 5 vdc.

If not, fix the wiring.

If it is, reverse the two wires between the H-bridge and the motor.

If you are using an RC motor controller, just reverse the motor wires.

If a wheel does not rotate:

If using an H-bridge,

Verify that the PWM signal to the H-bridge is active. It should be a square wave with a period of about 67 microseconds and a duty cycle of about 25%.

Verify that the brake signal (if used) is in the proper state.

if brake signal is opposite direction of H-bridge requirement, don't use the microcontroller brake and just wire the H-bridge brake input to always off.

If using and RC controller:

Look at the RC pulse signal on a scope and see that it changes from 1.5 msec. to about 1.625 msec. Check the ground and 5 vdc (if you need it).

calibration:

The first goal is to set KvL and KvR to where the wheels actually turn at about the commanded 250 millimeters/second.

The Lisp command is to go 250 mm/sec for 4000 millimeters. So, you are going to count how many revolutions the wheel makes during the command and see if it moves more or less than 4000 mm. If the distance the wheel travels is less than 4000, increase the associated Kv for that wheel. If more than 4000 mm, decrease the Kv. Keep repeating until the wheel goes pretty close to 4000 mm. This is also a rough setting and the results don’t have to be real accurate or even repeatable. Just get as close as you can.

Final KvL: ________________

Final KvR: ________________

Now, you will set the KdistL & R parameters. These should cause the distance traveled to become much more accurate and repeatable. Start by setting both KdistL and KdistR to 100 (this is a gain of 0.1 giving 0.1% PWM for each millimeter of distance error.

IMPORTANT: You will be finding the best value for Kdist by increasing the value in sequential tests until the drive system becomes oscillatory. You should be ready to shut off the motors quickly if oscillation occurs since it could be violent depending on your own drive train setup.

Run the Lisp “forward” command again while monitoring the data from the microcontroller serial bus. (you may want to record the data so you can go back and look at it later.) Looking at the parameter list in the front of the data stream, you will see there are two signals called DistErrL and DistErrR. These signals tell how far the actual wheel position is from the commanded position. These signals should become smaller in each test as the Kdist gains get increased.

If the run was not oscillatory, look at the data and see how much DistErr there was on the average during the run and record it.

Now, double the Kdist gains and repeat the above. Keep repeating until the system becomes oscillatory. When it does, stop the test, reduce the Kdist numbers towards the previous successful value and try again. You want to find at what values oscillation starts. When you do find out, then reduce the Kdist values by 20% to make sure that oscillation doesn’t happen.

OK, now look at the DistErr signals with your final gains. The signals should show only a few millimeters of error during the run and should stop almost exactly on 4000 mm.

Note: I'm not going to set the KaccL/R gains at this time. They are basically fine tuning and shouldn't matter much for basic operation.

Now, move the robot to a nice hard floor with as much space as practical for a forward run. Reduce the 4000, if necessary, to accommodate your space. Run the above test again and the robot will hopefully smoothly accelerate to about 250 mm/sec and travel 4000 mm and stop.

If you want to improve your calibration, measure how far the robot actually travels and modify the odomLnum and odomRnum so that each wheel travels as close as possible to 4000 mm. If you have both wheels set accurately, the robot should also go very straight. For really fine tuning, it may also be possible to change the denominator to make adjustments smaller than can be made with the numerator alone.

Setting for turns:

The PIVOTCONV determines the distance the wheel must travel for the robot to rotate a specified number of degrees. This number should be about 10 times the distance between the centers of your drive wheels in mm. Adjust the number until the robot rotates exactly 360 degrees when commanded to do so.

This is easiest, and most accurately, done by using Lisp to command a 360 degree turn (Lisp measures turns in tenths of a degrees, hence command 3600).

Find a reference line on your robot where you can see exactly what direction it is pointed. Perform the 360 degree turn. Sight again to see how close the robot is to where it started. Adjust PIVOTCONV as required.