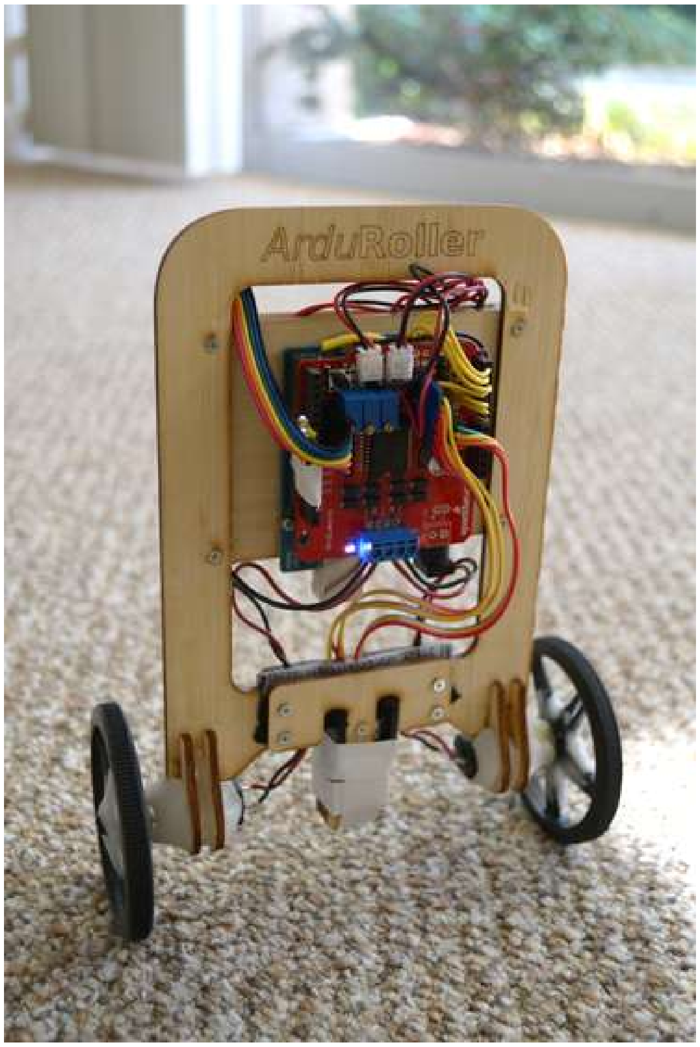
ArduRoller balance bot

by fasaxc on August 21, 2011

Table of Contents

Intro: ArduRoller balance bot

This instructable shows how to build my ArduRoller balance bot. It balances quite well on the spot and responds to most knocks pretty quickly but sometimes giving it a more gentle push sends it gliding across the room at a constant speed. I'm still working on that.



Step 1: Parts

Here are the parts I used:

1 x Arduino Uno
1 x Sparkfun Ardumoto motor driver shield
1 x Sparkfun BlueSmirf Bluetooth modem
1 x 150 degree/s gyro
1 x 1.7g Accelerometer
2 x Arduino header kits
2 x screw terminals
2 x 24:1 gear motor
1 x set of 70mm wheels
2 x JST connectors
2 x LiPo batteries
1 x basic LiPo charger
3 x Multi-turn 10k potentiometers
1 x SPST switch (Radioshack)
1 x Laser-cut bamboo chassis via Ponoko (link should allow you to make one from my shared design)
1 x LED
1 x Normally-open push switch
Assorted M2-04 machine screws (6mm - 16mm) (found on Amazon)
M2-04 nuts to match machine screws above
Assorted straight and right-angle break-away headers
Assorted jumper wires
Solid core wire
Stranded core wire
Instamorph (aka Polymorph) low-melt-point thermoplastic

Notes:

Chassis: the motors didn't quite fit the mounts I made so I had to sand them down and rebuild them with instamorph. I think the sensor bundle suffers from too much vibration, it might have been better to make it more solid rather than sticking out as it does.

Accelerometer: I originally tried building the bot with only an accelerometer for tilt sensing and no gyro. It turns out that approach is a non-starter -- the accelerometer gets overwhelmed by the acceleration due to the motors so it can't be used to estimate tilt while the bot is accelerating. OTOH, using only a gyro would make the bot susceptible to drift over time so you really need both.

Gyro: I used a 150 degree/s rate gyro. From looking at the telemetry from my bot, I'm pretty sure it sometimes clip if you give the bot a knocks so if I was starting over I'd probably look for a 300 degree/s model.

Wheels: the wheels are a little fragile, after a few knocks I noticed cracks around the axle so I strengthened them with instamorph.

Motors: I also tried sparkfun's 100:1 gear motors but they weren't fast enough. The 24:1 versions have plenty of torque and speed.

Bluetooth: I use the bluetooth modem for telemetry right now but I'm also planning to use it for remote control from my Android phone. If you omit it then the robot will still work but tuning it will be harder.

Pots: I added 3 10k multi-turn pots to the design to allow me to easily tweak internal values. Using 3 might have been overkill since I tend to tweak only one thing at once.

Instamorph: Amazing stuff. It's a tough, white plastic (resembling solid nylon) at room temperature but if you heat it in boiling water it turns into a pliable goo that's really easy to work with your hands. A heat gun is great for working with it too, allowing you to melt small areas.

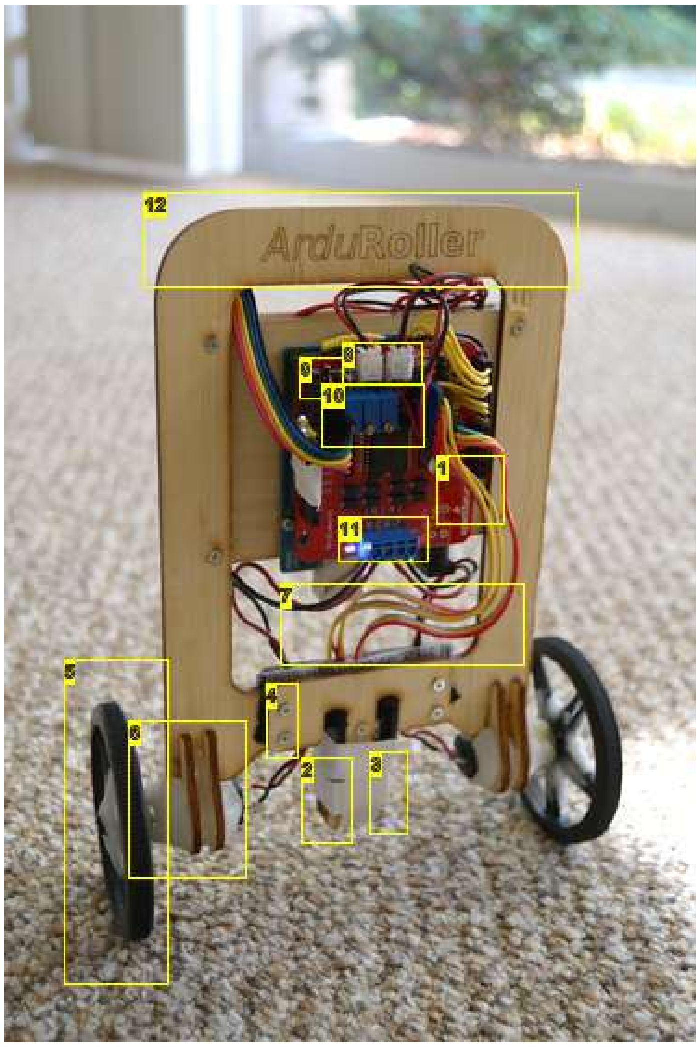


Image Notes
1. Arduino Uno Ardumoto
2. Gyro
3. Accelerometer
4. Machine screws and nuts
5. Wheel, repaired with instamorph
6. Gear motor, mount strengthened with instamorph
7. Assorted lengths/sizes of jumper wires
8. JST Connectors
9. Push switch
10. Trim pots
11. Screw terminals
12. Laser-cut chassis

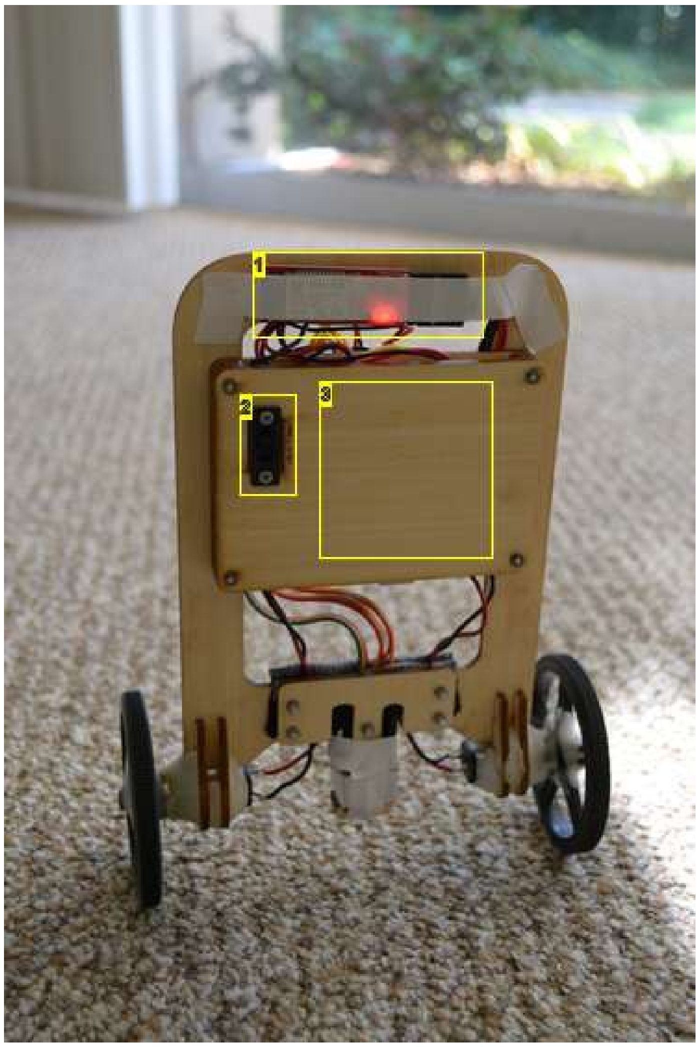


Image Notes
1. BlueSMIRF
2. Radioshack switch
3. Batteries inside this compartment

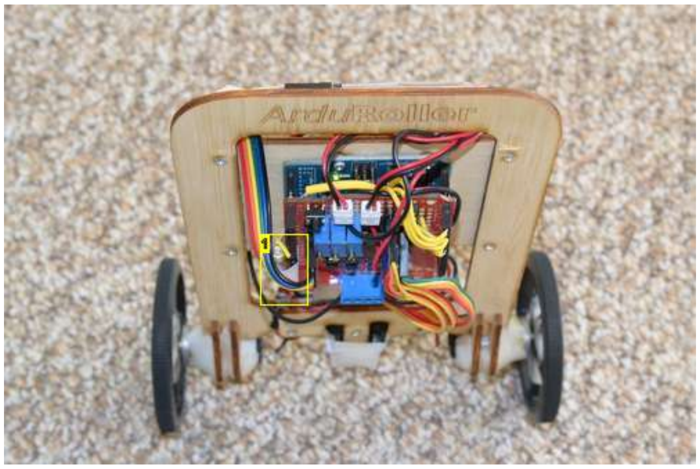


Image Notes
1. Status LED.

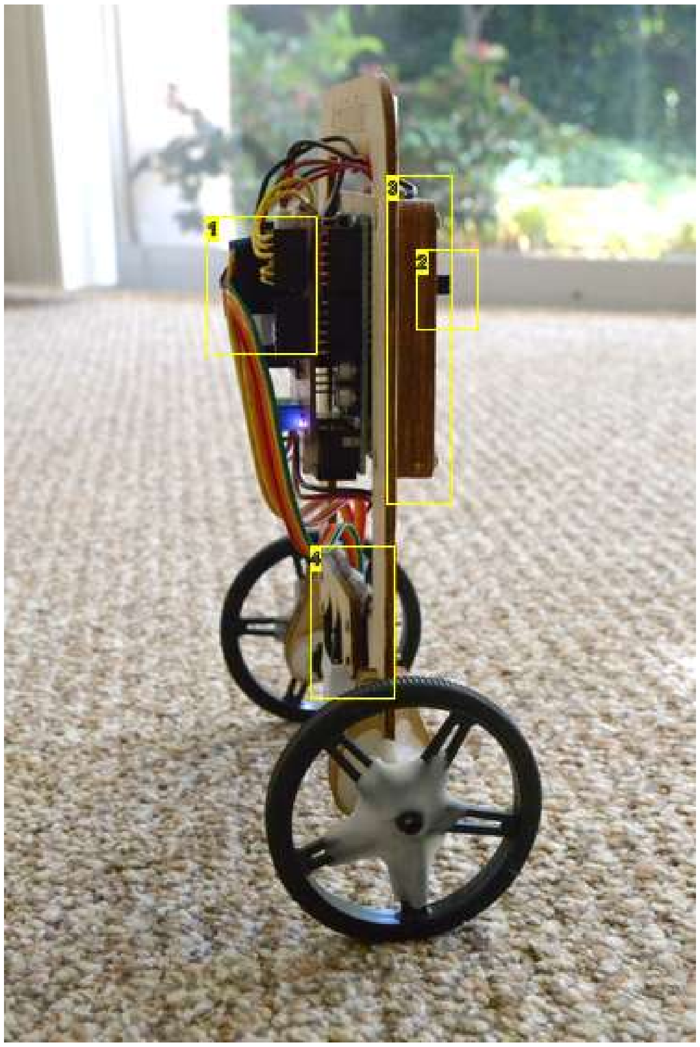


Image Notes
1. The headers on my model stick out quite a lot. They often get knocked when the bot falls over.
2. The switch can get knocked when the bot falls over. Might have been better to place it elsewhere.
3. Batery compartment is made by stacking several U-shaped pieces.
4. There's a T-shaped piece front and back with a gap between for cabling.

Step 2: Overall design

This step covers the overall design of the bot. I'll try to explain what each of the parts is for:

The brain of the system is the Arduino Uno, which contains an AVR microcontroller running at 16MHz.

1000 times per second, the microcontroller reads the current state of the gyroscope and accelerometer; updates its internal model of the bot and from that model decides how fast to run the motors to best balance the bot. (The code is all shared on my Github repo .)

The gyro is a rate gyro, which means that its output is proportional to the current rate of rotation. To estimate the current tilt, the microcontroller has to sum the incoming values, which it reads using an analog to digital converter. Unfortunately, no gyro or ADC is perfect, resulting in errors in the summation that tend to grow over time. If the bot used only a gyro to try to balance then its idea of "up" would slowly drift over time and it'd eventually fall over.

To counteract the gyro's tendency to drift, the bot uses a 2D accelerometer to measure the direction of gravity. When summing the gyro values it adds in a tiny fraction of the accelerometer estimate into the calculation. Just enough to balance out the drift. It can't add too much because the accelerometer is a very noisy sensor - it picks up vibration from the wheels and the acceleration of the motors.

The gyro and accelerometer are mounted on the axis of rotation of the wheels to get the best signal.

Once the microcontroller has decided how fast to run the motors it uses pulse width modulation to vary their speed and drives them through the Ardumoto shield. The shield is necessary because the motors draw far more current than the microcontroller can supply on its own.

The BlueSMIRF module provides serial-over-Bluetooth, allowing the bot to communicate in both directions with another Bluetooth-enabled device. I use my Android phone to relay the serial data to the console over the Android Debug Bridge. I'm also planning to send signals the other way to use my phone as a remote control.

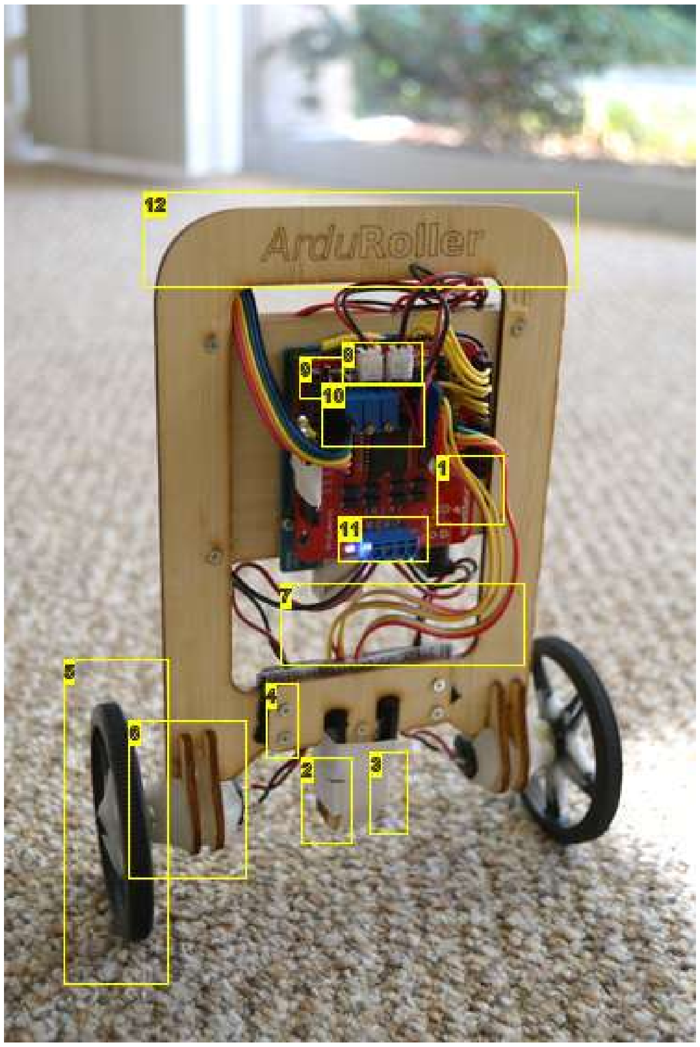


Image Notes
1. Arduino Uno Ardumoto
2. Gyro
3. Accelerometer
4. Machine screws and nuts
5. Wheel, repaired with instamorph
6. Gear motor, mount strengthened with instamorph
7. Assorted lengths/sizes of jumper wires
8. JST Connectors
9. Push switch
10. Trim pots
11. Screw terminals
12. Laser-cut chassis

Step 3: Chassis

I designed the chassis using Inkscape on Linux and used Ponoko for laser cutting. The SVG version is in my Github repository and I've published it on Ponoko .

I was pretty pleased with how it turned out given it was my first try at using Ponoko. The only part that didn't work so well was the motor mount, which I had to sand down and augment with Instamorph.

Using Ponoko, the wood grain runs left-to-right so some parts need to be placed sideways. The laser cutter is incredibly precise and does a great job on screw holes.

Be sure to read Ponoko's design rules if you intend to modify the chassis design. There are gotchas like needing to put "nodes" in push-fit slots to take account of material differences and the fact that the laser removes 0.2mm around your line.

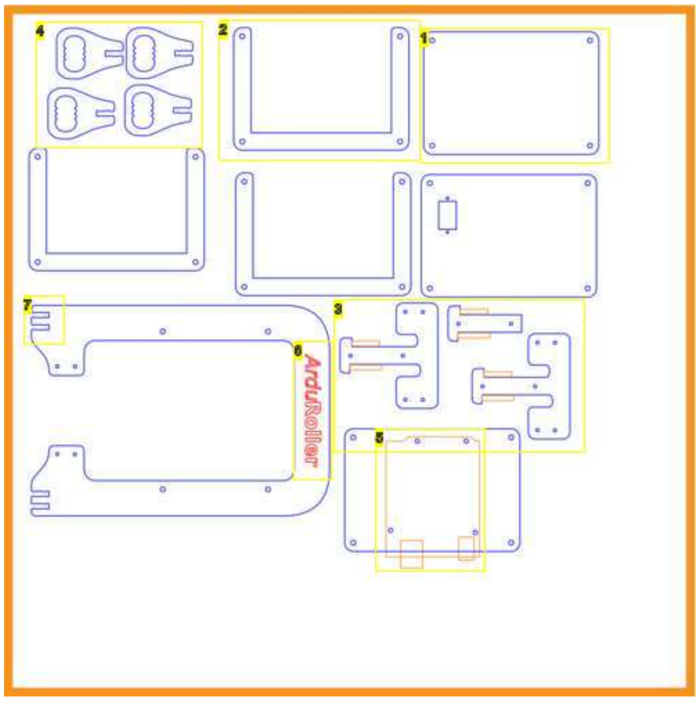


Image Notes
1. Didn't end up using this piece
2. These pieces stack to make the battery compartment.
3. These stack to make the sensor bundle. The bottom is asymmetric so that the larger sensor sits lower. The chips on both sensors should sit on the axis of rotation.
4. The motor mounts were troublesome. I had to sand off the nodes on the inside of the oval because the motors didn't fit.
5. Arduino mount. Ponoko ignores orange lines.
6. Ponoko etches red lines rather and cutting them.
7. Motor mounts fit here. It's quite snug.

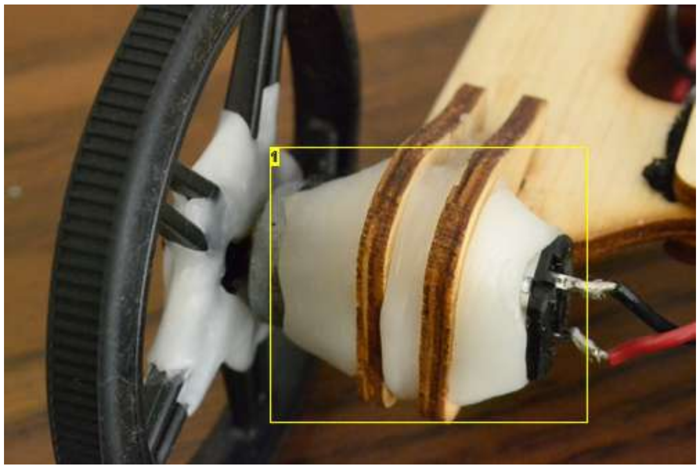


Image Notes
1. Squishing Instamorph into the gaps and shaping it round the motor did the trick to hold it in place.

Step 4: Electronics

The schematic in this step shows how I wired up the electronics for the ArduRoller. I used the prototyping area on the ArduMotor to hold the components. It was a tight fit but I managed to get it all in there even though I added components pretty much at random.

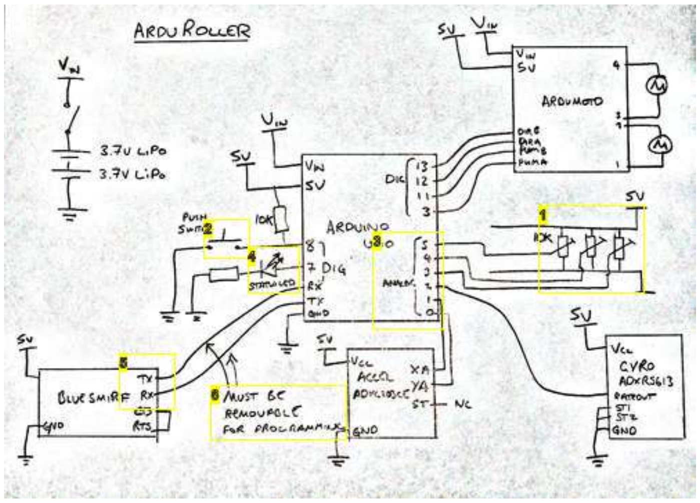


Image Notes
1. Trim pots
2. Push switch
3. Analog inputs
4. Status LED
5. TX on the SMIRF connects to RX on the Arduino
6. The SMIRF must be disconnected to program the Arduino.

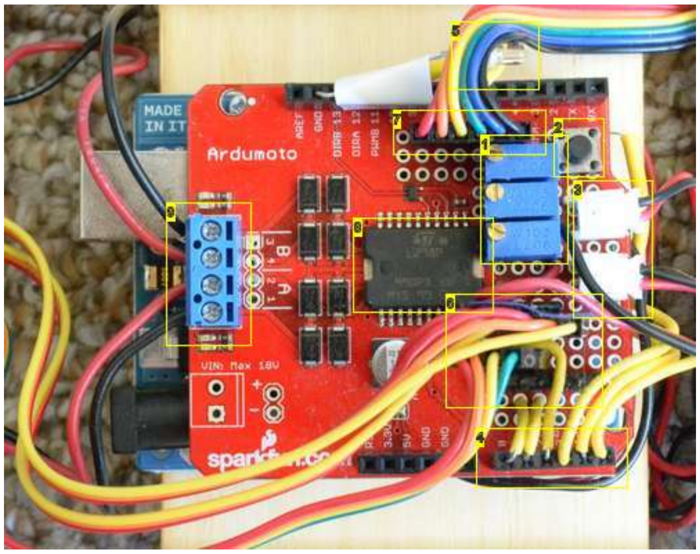


Image Notes
1. Trim pots
2. Push switch
3. Battery connectors
4. Analog inputs
5. Status LED
6. Connections to sensors.
7. Connection to BlueSMIRF
8. ArduMotor driver chip
9. Connections to motors.

Step 5: Code

I've shared the code for the project on Github . I used Eclipse rather than the Arduino IDE because it is much more robust and has a better editor. However, I did use all the Arduino libraries so the code would probably run as a sketch with a little tweaking.

The code essentially implements a PID controller for the tilt. It has quite a high integral term, which causes the bot to overshoot when it's correcting for a nudge. That tends to null out the bot's tendency to drift along, balancing upright but not stationary. This is the key line that sums the PID terms:

```
speed = tilt_rads_estimate * TILT_FACT +
        tilt_int_rads * TILT_INT_FACT +
        gyro_rads_per_sec * D_TILT_FACT;
```

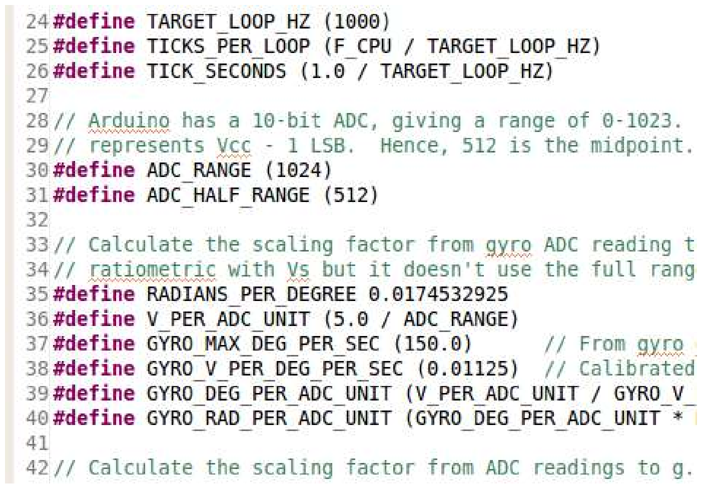
There are several places that you might need to tweak in the code to make it work with your bot:

The GYRO_V_PER_DEG_PER_SEC constant sets how much the output of your particular gyro changes for a change of one degree per second in rotation rate. I found that mine was at the end of the bell curve in the datasheet.

Likewise, the ACCEL_V_PER_G constant sets the value for the accelerometer.

You may also want to use the inputs form the pots to tweak other values like the X offset, which tilts the bot backwards or forwards. This needs to be set correctly else the bot will not balance.

The push button causes the bot to enter calibration mode for 10 seconds. To use it, lay the bot on its back and push the button then leave it for 10 seconds. It will automatically calibrate the gyro rate null point.



```
#define TARGET_LOOP_HZ (1000)
#define TICKS_PER_LOOP (F_CPU / TARGET_LOOP_HZ)
#define TICK_SECONDS (1.0 / TARGET_LOOP_HZ)

// Arduino has a 10-bit ADC, giving a range of 0-1023.
// represents Vcc - 1 LSB.  Hence, 512 is the midpoint.
#define ADC_RANGE (1024)
#define ADC_HALF_RANGE (512)

// Calculate the scaling factor from gyro ADC reading t
// ratiometric with Vs but it doesn't use the full rang
#define RADIANS_PER_DEGREE 0.0174532925
#define V_PER_ADC_UNIT (5.0 / ADC_RANGE)
#define GYRO_MAX_DEG_PER_SEC (150.0)      // From gyro
#define GYRO_V_PER_DEG_PER_SEC (0.01125)  // Calibrated
#define GYRO_DEG_PER_ADC_UNIT (V_PER_ADC_UNIT / GYRO_V_
#define GYRO_RAD_PER_ADC_UNIT (GYRO_DEG_PER_ADC_UNIT *

// Calculate the scaling factor from ADC readings to g.
```

Step 6: Next steps

There are lots of areas for improvement in the code:

Right now, the gyro and accelerometer values are summed using a simple complementary filter. There are much better techniques out there such as a Kalman filter but I haven't got around to getting my head around it!

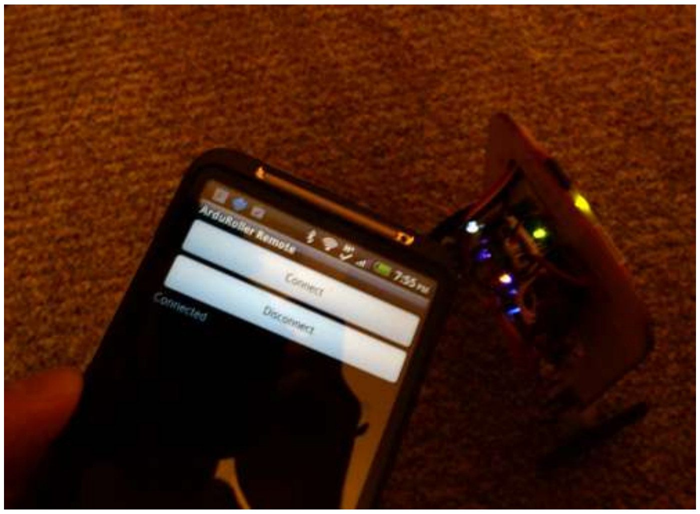
The bot doesn't currently have an estimate of its linear position or velocity so it can't correct itself if it ends up drifting across the room, well balanced but with non-zero speed. I've experimented with tracking the acceleration and integrating to get velocity but with little success.

There are better control methods for this type of problem (such as state-space control) that can better deal with multiple-input-multiple-output problems.

It could use a good tidy up.

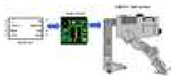
I'm currently working on better tuning the code to make it more robust, experimenting with other control mechanisms and an Android-based remote control.

Hope you enjoyed this tour of my bot!



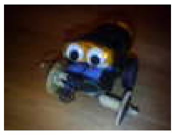
Related Instructables

Tiko-Bot 0.1 (Photos) by tarekskydiver


Build a Mini-Walking Bot by Mrdon219

Balance-BOT (Photos) by daniel2008

Domo Kun WobblyBot, Simple Self Balancing Robot by Chein


DuneBot 1.2 - An EasyBot Slideshow (Photos) by Win Guy

Bristle Bot II - from a cheap electric toothbrush by The Speaker Guy