DIY Segway Technical Documentation

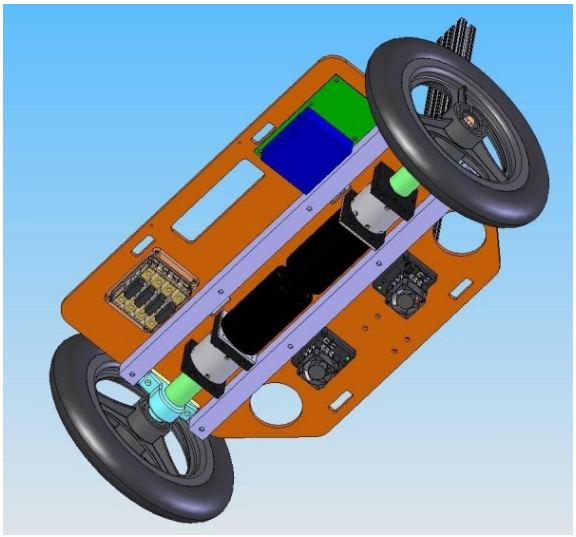
Rev. 0, 8/23/2007

Note/Warning/Disclaimer: Segways, like any large machines, can be dangerous if appropriate safety precautions are not observed. DIY Segways, including this one, are *particularly* dangerous because they often lack the redundant safety features of commercial Segways. This technical documentation is intended for informational, not instructional, purposes. Attempt/build at your own risk!

Overview

Building a DIY Segway-like scooter was an incredibly fun project, and we would like to share the experience with others in as much detail as possible. The intention of the technical documentation is not to provide step-by-step instructions on how to build this particular machine, but to share some of the resources that made it possible. Hopefully, others will find these resources useful in their own projects, self-balancing or otherwise. If you have further questions or comments about our project, please contact:

seg-info@mit.edu



Files

In **segspecs.zip**, you can find the following files:

- **segspecs.pdf**: This document.
- **BOM.pdf**: Bill of materials, including supplier and vendor information.
- **PCB.zip**: PCB manufacturing files, in gerber format.
- **CAD.zip**: All of the SolidWorks files, plus a few .dxf files of the base plate waterjet cut.
- **SegwayDash.zip**: The custom VB dashboard source (used for wireless debugging).
- **controller.pdf**: Some more visual notes on the custom controller.
- **filter.pdf**: A detailed explanation of the digital filter we used to estimate angle from sensors.
- **segwaycode.c**: The actual code implemented on the PIC microcontroller for control.

Quick Specifications:

Overall Footprint: 27”x15”
Wheel Diameter: 12.5”
Ground Clearance: 5”
Weight: 52 lbs. with battery, 39 lbs. without
Rider Capacity: 250 lbs.
Peak Motor Power: 343 W (0.46 HP) each
Peak Motor Torque: 2.45 N-m (347 ozf-in) each
Max. Continuous Motor Current: 40 A each
Gear ratio: 16:1
Theoretical Top Speed: 5 m/s (11 MPH)
Software-Limited Top Speed: ~3 m/s (~7 MPH)
Battery: 12V Sealed Lead-Acid, 18 A-hr
Normal-Usage Battery Life: ~45-60 min
Controller Update Rate: 100 Hz
Telemetry Transmit Rate: 15 Hz
Telemetry Transmit Range: 300 ft
Total Materials Cost: <$1,000
Number of Cup Holders: 2

Base Plate Design and Fabrication

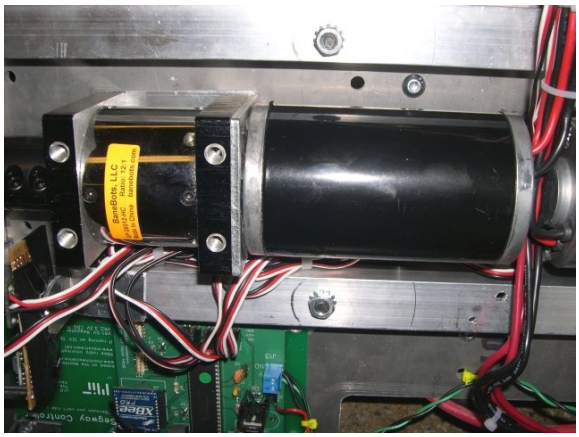
The base plate is really the key to much of the mechanical construction of our machine. All of the tricky alignment of motors, gearboxes, and bearings was consolidated onto one piece to be precisely machined. The base is made from ¼"-thick 6061 aluminum plate (vendor: McMaster Carr[1]). It has been pointed out to use several times that there are much cheaper alternative materials and/or vendors. One easy and inexpensive alternative is Big Blue Saw[2], which offers waterjet cutting straight from CAD files and has reasonable prices which include material costs. They offer a wide range of materials and thicknesses, as well. (Clear polycarbonate base, anyone?) The plate itself contributes only partially to the rigidity of the base. Most of the strength comes from two 1"x1"x1/8" aluminum box extrusion cross-beams. Placing the bearings as close as possible to the edge of the base ensures that the load is carried directly from the wheels into these two cross beams without causing significant deflection of the drive shaft or motors.

The base plate was collaboratively designed in SolidWorks[3] and cut on the waterjet at the MIT Hobby Shop[4]. Holes for mounting the motors, bearings, handlebar, electronics, and cross-beams were included, as well as the now-famous cup holders. It weighs approximately 7 lbs. A .dxf file of the plate is available in **CAD.zip**.

[1] http://www.mcmaster.com
[2] http://www.bigbluesaw.com
[3] http://solidworks.com/pages/products/edu/studenteditionsoftware.html
[4] http://hobbyshop.mit.edu

Motors and Gearboxes

Picking motors was one of the first things we did. After briefly considering larger, more powerfully 24V NPC[5] motors such as those used on Trevor Blackwell's inspirational design[6], we chose to use the familiar and inexpensive 12V CIM motors provided in the FIRST Robotics kit of parts. At just under ½ HP peak output each, they were a compromise that we thought would work okay with our lightweight design. They also offered the advantage of being compatible with a compact 16:1 in-line planetary gearbox (vendor: BaneBots[7]).



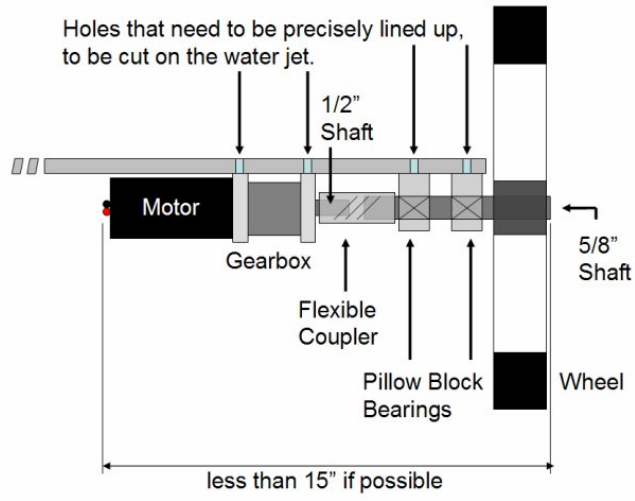
BaneBots also sells a two-motor adaptor for this gearbox, creating potential for a design with twice the power. Backlash in the gearbox is noticeable, but it is less than five degrees. Note that these gearboxes have some well-known assembly quirks, particularly the axial alignment of the motor pinion gear, so read the BaneBots documentation carefully.

Coupling, Axle, and Bearings

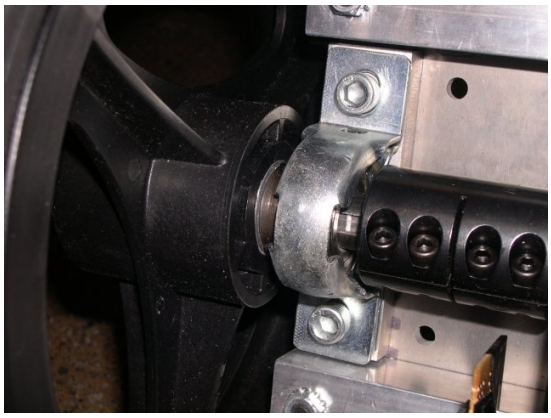
Our original design called for a flexible coupling between the 1/2" gearbox shaft and the 5/8" drive shaft, to allow for misalignment and minimize shock loading on the gears. The problem we encountered with this design was that in order to accommodate two bearings, spaced out enough to support the drive shaft, our total width would be

[5] http://www.npcrobotics.com
[6] http://www.tlb.org/scooter.html
[7] http://www.banebots.com

over 32". Besides looking somewhat ridiculous, this would cause problems getting through doors.



We pretty quickly decided to give up on the flexible coupling in order to get the width under 30". With a rigid coupling between the gearbox and the drive shaft, the bearing on the gearbox provides a second point of support for the rigidly-coupled shafts. Quick calculations indicated that with a 5/8" steel keyed drive shaft and bearing placed as close to the edge of the base as possible, there was enough support in the system so that the gearbox would not be damaged by deflection of the shaft. (We've also seen these gearboxes survive under much less adequate constraints on FIRST robots.)



We did not know how the rigid coupling would affect alignment and shock loading transmitted to the gearbox. We bought some shim stock in case the alignment was a problem, but wound up not using it. As for shock loading, we'll have to wait and see how well the carrier plates inside the gearbox hold up. So far, they seem okay.

One major design oversight was the fact that the motor actually sticks out a bit further than the edge of the gearbox. We had to shim up the gearbox and bearings with sheet metal to allow for this, although it could easily have been avoided by cutting a slot or milling a pocket on the base plate for the motor to rest in.

Wheels

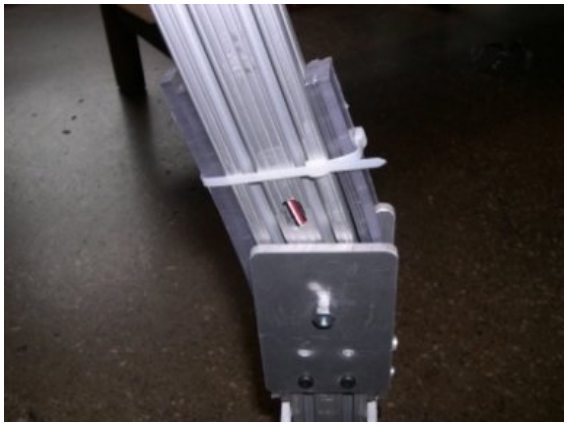
Based on our motor torque/speed characteristics, we needed to use relatively small wheels to get adequate performance. We chose 12.5" pneumatic wheels made by Skyway[8] because they were inexpensive, light, and had the 5/8" keyed hub we needed. Skyway has been a long-time supplier for FIRST teams and offers special pricing for them.

Handlebar

The handlebar was supposed to be the easy part…until the decision to go for lean steering. Aside from that part, it is just a piece of 80/20[9] 1"x2" extrusion. This stuff is great because it allows easy adjustment via sliding t-nuts, making height and angle modifications simple. We cut a few custom brackets for it on the waterjet out of the left-over aluminum from the base plate (see CAD files).

As for the lean-steering joint, we went through a bunch of iterations, including one with a combination of compression and tension springs that sounded like an old Buick suspension. The current design uses four strips of ¼" polycarbonate as leaf springs to center the steering joint. The forward/backward rigidity of the joint is workable, but not great and is something to look at for future modification.

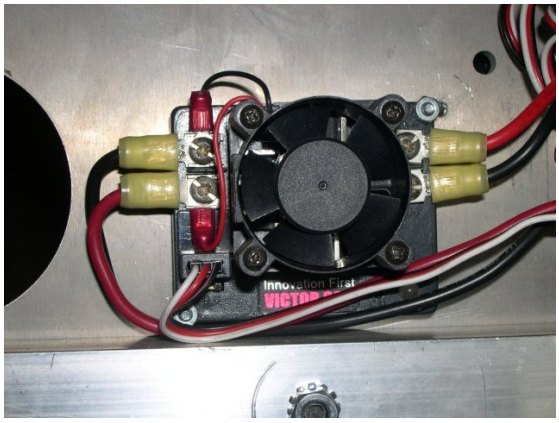
[8] http://www.skywaytuffwheels.com
[9] http://www.8020.net



Battery and Power Electronics

All of the power electronics on our machine are FIRST-legal kit components. The power source is a 12V, 18 A-h sealed lead-acid motorcycle battery that weighs 13 lbs. It is connected to a 120A main breaker with 6-gauge wire, then to a distribution block. Each motor line has a 40A circuit breaker. The controller also has a 40A breaker, but it is also protected by a 1A thermal polyswitch. The grounds are grouped together in the remaining slot of the distribution block. (Notice the aluminum ground jumper? Not the most elegant solution, but it works.) The chassis is *not* grounded.

The motor controllers are the reliable Innovation First[10] Victor 884 model found in the FIRST kit. Aside from being virtually indestructible (they have survived being rained on), they can supply 40A continuous and much higher peak currents without ever getting hot thanks to fans directly cooling the MOSFETs. They are driven by 1-2ms PWM signals, the same signal used by RC servos. These signals are easily generated by the PIC controller. The speed controllers can be updated at up to 100Hz.



Sensors, Signal Electronics, and Controller

We made use of three sensors: a gyroscope and an accelerometer for balancing, and a second accelerometer for steering. Unlike the commercial Segway, ours has no redundant sensors – you pretty much need all three to be working correctly for it to be rideable.

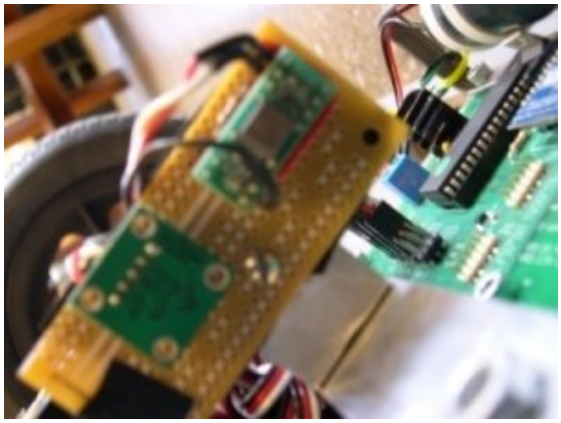
The sensors are all from the Analog Devices[11] iMEMS line (see B.O.M.). They report an analog voltage between 0V and 5V to the controller, with neutral angle or zero rate being near 2.5V, although each requires some calibration with regards to the exact offset. The gyroscope is used simply to measure angular rate. The accelerometer is used to indirectly measure the direction of the force of gravity, since it is really sensing force per unit mass along a given axis. This, along with a small angle approximation, gives an estimate of the angle to horizontal.

The controller is based on the PIC16F877 board of the Machine Science[12] starter kit. It is protected by a 1A thermal polyswitch, a diode to prevent reversing polarity, and a large filter capacitor before the 5V regulator (LM7805). The PCB was drawn out it out in a freeware program called FreePCB[13] and manufactured by Advanced

[10] http://www.ifirobotics.com

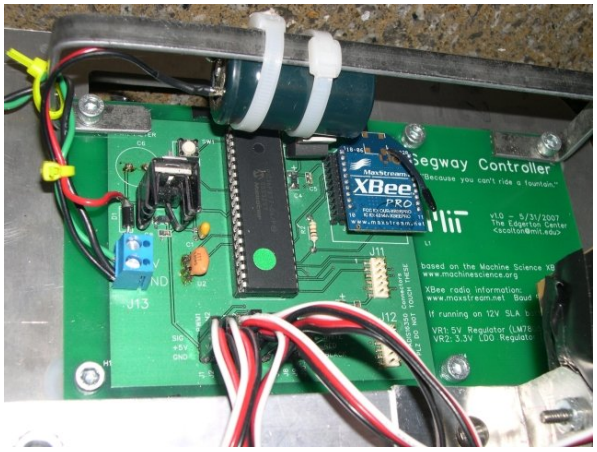
[11] http://www.analog.com
[12] http://www.machinescience.org
[13] http://www.freepcb.com

Circuits[14], which has an excellent student discount of $33/board. The PCB layout and Gerber files are available in **PCB.zip**.



The CPU clock for the PIC is generated by a 4 MHz oscillator. (Compare this to 4 GHz Pentiums…) The actual instruction and timer clock is ¼ of that, 1 MHz. This is relatively slow even for a microcontroller, and in a future upgrade we plan to move to a faster microprocessor. But even with the current setup and a good amount of floating-point control math, we can keep our control loop running at 100 Hz.

One thing we think is fairly unique about our controller is that its interface is *entirely* wireless. It can be reprogrammed without attaching any cables to the Segway and can transmit data from the sensors or controller to a laptop for debugging. This is all done via MaxStream[15]'s XBee radios. In a future mod, it might even be capable of wireless self-balancing control with no rider.



Signal Filtering

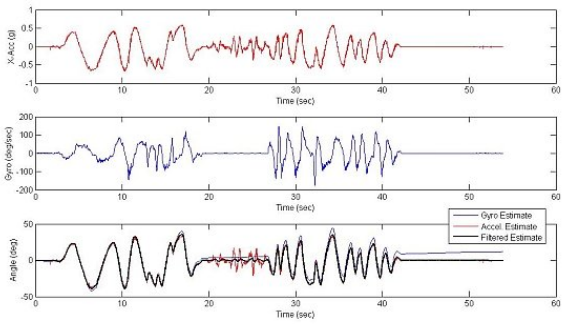
There are a number of problems with using direct sensor data for control. For one, with two half-horsepower electric motors on the same power and ground line as the controller, there is bound to be noise in the system even with a 6800µF power supply filter capacitor for the controller.

There are also physical reasons why the data from the accelerometers and gyroscope has to be filtered. The accelerometers measure a change in angle by the component of the force of gravity along their sensitive axis (horizontal). But they also report other horizontal accelerations from the motors or, in the case of steering, wiggling of the handlebar. The gyroscope measures angular rate and can be used to estimate angle by integration, multiplying the rate by the small time step to get the small change in angle each time through the program loop. But this method can lead to drift: the angle changes slowly over time if the sensor is not perfectly zeroed (which it never is).

For the steering, we implemented the simple hardware solution of adding capacitance to the output filter of the accelerometer, creating a "low-pass filter" that smoothes out short periods of acceleration and lets through only the long term effects of gravity. The ADXL203 data sheet explains how to do this. We've been experimenting with capacitors in the 4.7-10µF range.

For the balance controller, though, this method would cause too much lag in the angle estimate. Most self-balancing robot / homemade Segway sites refer to some kind of digital filter which combines the accelerometer and gyroscope data to get a clean, fast angle estimate. The Kalman Filter is often offered as a possibility, although nobody ever seems to take the time to explain it. (And for good reason: It is mathematically complicated and would not run on a PIC.) Our solution is a much simpler software filter that we are just going to call a "digital complementary filter" for lack of any known technical reference to it. It is actually the same as the filter implemented in the Balancing Robot Wheeley[16] project.

[14] http://www.4pcb.com
[15] http://www.maxstream.net
[16] http://www.dena.demon.nl/balansbot.html



"After a lot of trials and calibration the performance of the Kalman filter was not satisfying. I developed another simple filter again on trial and error."

-Balancing Robot Wheeley page

Although it is fairly simple to explain, we will leave it out of this document and instead point you to **filter.pdf** for a more colorful explanation.

Balance Control

For all the controller setup (timers, wireless communication, etc.) and signal conditioning, the actual balance control is a fairly short bit of code:

```
motor += (KP * angle) + (KD * (float)gz_vel);
```

It's a "PD" controller, standing for Proportional + Derivative. The motor output is scaled proportionally to the (filtered) angle estimate and its derivative, the angular velocity measurement. Using the angle alone would have a similar effect, but with more oscillations. This of it this way: The angle term provides a spring-like effect (F = kx) restoring the base to the horizontal position, while the angular velocity term is more like a damper. There are a lot of great references[17] available on PID control theory.

There is more to be done after the simple PD controller, some of which we've gotten to and some of which we are still working on. For one, steering must be taken into account. This is done simply by adding an offset to one motor and subtracting it from the other. Also, motor values must be limited so as not to overflow their variable types or exceed the limits of the motor controllers.

One major piece of control that we have yet to add to our code is the speed limiter. Ideally, the controller should push back harder if you try to lean forward/backwards at high speed, to prevent you from getting into a condition where the motors can no longer catch up to you. Testing this bit is difficult, so we've put it off so far and worked only at low speeds. **Do not try to take a DIY Segway up to high speeds without a helmet/pads/etc. because you are almost guaranteed to fall off.** Best to think of it as an extreme sport…

The current controller code, with comments, is in this zip: **segwaycode.c**.

Design Notes

If you've made it this far through the documentation, you deserve a nice summary of what worked and what still needs work on this project. This way, when you are working on your own project, you can learn from our successes and not-so-successes. So, in no particular order, things that worked really well:

- The base. It is light, but wonderfully rigid and easily supports the load of the rider *jumping* on, even with the simpler one-bearing setup. The cross-beams take all of the weight and the bearing/gearbox alignment stays true. Designing it in SolidWorks and machining it on the waterjet paid off big time.
- The sensors. Yes, they are noisy analog sensors. But with some signal conditioning, they absolutely work. They are also tiny and dirt cheap now. We had an ADIS16350 digital IMU, but decided not to use it because these are far simpler.
- The XBee radios. These things are so easy to use and cut debugging time in half, easily.
- The Victor884 speed controllers. After an hour of riding, the motors get pretty hot, but the speed controllers are always cool to the touch. They are incredibly efficient and robust. Only minor issue is the dead band, but we accommodate for that in software.

[17] http://www.chiefdelphi.com/media/papers/1823
http://www.chiefdelphi.com/media/papers/1911

And things that could be better:

- The motors. They may be just a bit underpowered for this type of application. For normal operation on flat surfaces, they work great. Speed bumps, turning on rough terrain, etc, not as well. But they are consistent with the lightweight, compact design and serve their purpose.
- The microcontroller. The Machine Science online IDE is excellent, but we are eager to move from the PIC to their new development environment for the Atmel AVR line. These have more code space and are significantly faster (and cheaper).
- Steering. A combination of mechanical and control problems still need to be worked out. We got the kinks out of the joint, finally, but are still working on getting the steering to be smooth and controlled at any speed. The dead band on the motor drivers makes turning while coasting a somewhat involuntary adventure, but that should be fixable in software.

Wrap-Up

This was a great project and we think proves that even seemingly complicated technology is within reach for high school-level engineering projects. We're not suggesting that everyone go out and build a Segway (although wouldn't that be interesting), but the technologies we used can be applied to any number of cool projects that we can't wait to see.