Autonomous Satellite Tracker

Elwood Downey, WB0OEW

Last updated Wednesday, November 18, 2015

Abstract

This paper describes a completely autonomous Earth satellite tracking mount we call the Tracker. In conjunction with a 9 DOF sensor attached to the antenna boom and a GPS receiver, the 2 axis gimbal will track any Earth satellite within 2.7 degrees in real time without any orientation setup calibration of any kind. The Tracker system includes a built-in web server and WiFi access point which performs all monitoring, command and TLE upload from any web browser including smart phone or tablet. All components are off-the-shelf so no custom electronics, machining or other skills are required except that which is needed to attach an antenna boom to a flat plate. Everything can be powered from a single DC supply of at least 6.5 volts such as a LiPo battery pack or solar charge system. At the time of this writing, total cost for the electronics and gimbal is less than $400.

Introduction

Observers have been tracking earth satellites with gimbal mounts since the beginning of the space age[i]. To this day one of the challenges has always been to align these mounts so the theoretical calculations of satellite azimuth and elevation could be transformed to the mount co-ordinate system. After reading about the availability of low cost MEMS[ii] devices that directly measure absolute spatial orientation I wanted to build a mount that avoided the tedious calibration step by measuring directly the pointing direction of the payload.

Going one step further, I also wanted to eliminate the need to have any prior knowledge of (or make assumptions about) the gimbal geometry, axis orthogonality, motor assignment and axis rotation angles. Achieving this would simplify the mechanical requirements of the gimbal and wiring, and allow the use of simple off-the-shelf hobby servo motors and robot hardware for light weight antennas.

The final goal was the ability to control and monitor the entire system from my smart phone without needing to first install an app. The most flexible way to accomplish this is by imbedding a web server so I needed enough memory in the controller to accomplish this.

The first goal is achieved by measuring the spatial orientation of the antenna directly using a combination of 3D magnetic, accelerometer and gyroscope sensors. When packaged together, these devices are referred to as having 9 Degrees-of-Freedom, or simply 9 DOF sensors. The magnetic sensor provides the direction of the local magnetic field, including tilt. This is combined with knowledge of the local vertical gravity vector from the accelerometer to produce local elevation and azimuth with respect to magnetic north. To get a bearing from true north, the latitude and longitude from the GPS is combined with the World Magnetic Model[iii] to compute the local magnetic declination correction factor. Information from the gyroscope provides additional stability and repeatability information. The final correction is to apply a simple model for atmospheric refraction to elevation based on nominal assumptions for air temperature and pressure (although these too could be measured quite easily, the maximum effect at the horizon is about one half degree which I decided was not worth refining further). Taken together, these measurements provide an absolute measure of antenna direction in the local horizon coordinate system which is exactly what is produced by the orbit propagator.

Now that we have the measured antenna direction and a computed predicted direction from the propagator in the same coordinate system, the second goal is to drive the gimbal motors in such as way as to reduce any difference between the two. Historically this was done in closed-form by using a transformation matrix determined ahead of time that relates the gimbal axis coordinates to the local horizon coordinates. With the 9 DOF sensor reporting absolute pointing, this is no long necessary. My second goal is achieved by just moving each motor a known amount and measuring directly the change it causes in azimuth and elevation. Then all subsequent moves are made based on the difference in sensor and target positions using the motor that best reduces the error in each direction.

The main thing to note here is that this does not require any knowledge of the gimbal orientation or even which motor operates which axis. The antenna makes three calibration moves the first time a target is tracked and from then on just maintains pointing automatically. The calibration need only be repeated if the gimbal orientation is changed drastically, modest changes of a few tens of degrees in any direction are accommodated automatically. So, for example, the Tracker can be mounted on a rather flimsy mount and still works very well.

Finally, the web server turned out to be straight forward also. I am already fluent programming with Javascript, HTML, CSS and the HTTP headers that are used between browser and server so I wrote my own server state machine from scratch on top of the basic Arduino Ethernet library. The main interactive page is sent immediately when connecting to the server URL address. From then on, the page uses XMLHttpRequest objects to poll for updated values. All state variables are updated and reported using a consistent NAME=VALUE syntax, where the NAME usually matches the corresponding DOM display element id. Setting a new value is per-

formed with a POST command. These simple AJAX techniques allow the Tracker to be controlled from other client devices if desired. More details of using the web interface are provided later.

It is worth noting that the default Arduino Ethernet library is very inefficient when transmitting strings stored in Flash. I discovered this is because the EthernetClient class subclasses from the Print class to inherit the handy print() and println() family of methods. Unfortunately, the Print class sends each character as a separate write command to the Ethernet layer, which means each character is sent in a separate packet. I made a small change to the Print::print() method in Print.cpp that handles Flash strings so it buffers more characters per packet. This sped up the web page loading time by a factor of 50. My modifications are available as part of the software for this project. This change is entirely optional; if you are uncomfortable modifying standard Arduino library code everything will still work just fine, but the web page will update slower and use more bandwidth.

The only function the Tracker does not perform is rotating the antenna along the boresite axis in response to changes in polarization. This would certainly be good to have but so far I have found neither a means to measure polarization nor to implement the maneuver. So even though the Tracker maintains pointing perfectly well, it can still be necessary to give the antenna an occasional twist by hand to maximize signal strength.

Implementation Decisions

Figure 1 shows the Tracker attached to a tripod and supporting an Elk 2m/70cm LPDA antenna[iv]. Figure 2 shows the inside view of the Tracker electronics box. Figure 3 is a block diagram showing how each electronic subsystem interconnects. Table 1 shows the major bill of materials. Next I elaborate the role of each and share my experiences that lead to each choice.

The main processor is the Arduino Mega 2560. I began with the model Uno but eventually I could no longer squeeze everything into its 32KB Flash and 4KB RAM storage. The Mega has 8x as much Flash and 2x RAM which is plenty. The Tracker uses about half the Flash on the Mega for code and constant strings and about half of the RAM is used for mutable variables, leaving about 4KB of RAM for stack.

To control the two hobby servo motors I started by using the Arduino software servo library. However, the servos did not move smoothly and the cause turned out to be interference to the pulse timing by other libraries that lock out interrupts, even briefly. Servo position is directly related to pulse duration which is sensitive to changes on the order of a few microseconds so it doesn't take much timing change to cause unwanted motion. The solution I chose was the 12 channel servo controller from Adafruit. This offloads all the timing from the Arduino and only requires a two-wire connection using the I2C bus to issue the desired pulse length for each channel. It also has the added benefit that the servos actively hold position until a new command is issued.

The 9 DOF sensor chosen is the BNO055 made by Bosch. It is available on a convenient breakout board from Adafruit and is compatible with their Sensors library. The advantage of this sensor package is it includes an onboard processor that performs all the consolidation of the three sensors automatically and outputs directly its absolute spatial orientation as Euler angles[v] interpreted here as azimuth, elevation and roll. As anyone who has tried to manually fuse together these types of sensors knows, this saves quite a lot of tedious mathematics. This sensor also connects to the Arduino using the same I2C bus as the servo controller but does not interfere because they each have a separate bus address.

The GPS module I chose is from Adafruit. It has a built in antenna which works pretty well but I also allowed for the connection of an external antenna if necessary. This module communicates with the Arduino using a UART, or “serial” connection. This revealed an additional advantage to using the Arduino Mega because it has four hardware serial ports available which allows one to be dedicated to the GPS. The Uno only has one and it already serves duty with the USB boot loader. A software serial library could be used with the Uno to use other pins but at the expense of higher overhead and a more limited bandwidth.

I wanted WiFi ability so I could control the system from my smart phone. However, I tried several WiFi modules and shields but found none to be reliable. Even the best one from Adafruit would work for a random time, anywhere from seconds to hours, and then just mysteriously stop. In stark contrast, all models I tried of wired ethernet proved to be 100% reliable, even including the oldest modules that use the WizNet W5100, so I ended up using a generic version made by Sunfounder and it works just fine. In order to accomplish my goal for WiFi, I connected the wired ethernet directly to a $20 WiFi adaptor made by TP-Link, model TL-WR702N. In stark

contrast, this combination works beautifully. The adaptor I bought can be configured either as its own Access Point to broadcast a separate WiFi network just for the Tracker, or it can transparently connect the Arduino to an existing WiFi network. To configure this adaptor, just plug in power, use a laptop or other computer to connect to its WiFi node using the password printed on the case and the IP assigned by your DHCP router, then proceed with the Quick Setup for either AP or Client mode as preferred.

I wanted everything to run off one self-contained power source. To do so, I ended up using two separate power conditioning modules. I used modules based on the LM2596 DC-DC adjustable buck converter, widely available for a few dollars. One supplies 5 V to the Arduino and its peripherals, the other supplies 6V dedicated to powering the servo motors. This separation provides clean power to the electronics and isolates them from the wide load swings and voltage spikes that occur from the motors. The LM2596 adds 1.5 volt overhead and the servos will still operate well enough with 5 volts, so the minimum supply voltage is 6.5 volts. This is perfect for a 2S LiPo battery. With a 2200 mAh pack the Tracker runs for about five hours before needing to be recharged. If you use a 3S LiPo pack, be sure to use a protection circuit to avoid running the pack below the minimum safe discharge voltage of about 9 volts. If desired, solar cells could also be used.

The gimbal is one channel-mount pan platform and one tilt platform from ServoCity.com. By raising the tilt platform these provide a full 400º of azimuth motion and 135º degrees of elevation. The pan platform has a hollow shaft which simplifies cabling to the 9 DOF sensor and tilt motor.

Assuming the Bosch spatial sensor is accurately aligned with the antenna, the largest contribution to pointing error is the sensor itself, which claims a maximum magnetic heading error of ±2.5 degrees. The next largest source of error is the orbit propagator. The code used here is based on a very clean rendering[vi] by Mark VandeWettering, K6HX, of PLAN-13[vii] by James Miller, G3RUH. After the updates to the solar elements posted in 2014 the code produces topocentric values within 0.2 degrees compared to a more rigorous SGP4 code[viii] within a few days of the TLE epoch. Thus, combined, if the Bosch sensor behaves according to its published specification, this Tracker should maintain pointing to within 2.7 degrees, which is sufficiently accurate for antennas with symmetric directivity up to 31 dBi[ix].

Construction Tips

In a nut shell:

1. assemble the electronics box, gimbal and wire everything together
2. attach the Tracker to a small tripod or other platform
3. attach your antenna to the tilt plate and attach the Bosch sensor
4. power up and program the Arduino
5. connect to the Tracker web page
6. check the Gimbal and Sensor (and GPS if used) are reporting correctly
7. set the minimum and maximum limits on each Gimbal axis.

Attach your antenna to the tilt platform so it points straight up when the tilt platform is run all the way over on its side such that the plane of the tilt plate is also vertical. I attached my Elk LPDA to the tilt plate with two bolts passing through the PVC support pipe. Position your antenna of choice on the tilt plate so the antenna load is roughly balanced to help reduce the load on the tilt servo. The tilt platform develops over five foot-pounds of torque so it should be fine to add a rear counter-weight to the boom if it allows you to balance the antenna better. Be aware that when the target is near zenith the rear of the antenna might hit the pan servo or your support platform. I added an extension tube to the pan platform to allow it full sky access with no interference.

Attach the Bosch sensor breakout board such that:

- the **short** sides are **parallel** to the antenna boom,
- the **populated** side of the board faces **skyward** and
- the **long** side with the control pins (SDA, SCL etc) points towards the **rear** direction of the antenna pattern.

Take some care to make this accurate and secure because the overall pointing accuracy is entirely dependent on how parallel the sensor is to the antenna bore site. The position along the boom does not matter but since one of the sensors is measuring magnetic fields, mount it as far as possible from anything containing iron. If you decide to use machine screws be sure to use brass and not steel. If you encounter erratic measurements while transmitting, you can wrap the sensor in clear wrap then wrap again with aluminum foil. If you still have interference, you can try shielded cable.

Power up the Tracker. Using the Arduino IDE, connect to your host PC with a USB cable and program the Arduino. Be sure to select the model Mega 2560. If you use a normal USB cable be sure to turn off the Tracker battery power while the cable is connected. This is because the two supplies are wired in parallel so one will inevitably back-drive the other which may be damaging. I found it much more convenient to prepare a “programming cable” by carefully slitting length-wise about an inch into a spare USB cable and cutting the red wire. This disconnects the USB power source and avoids any chance of conflict but does mean you will need to power the Tracker from its own source while programming.

Either connect with WiFi or attach a CAT5 cable to the wired ethernet controller. The default IP is 192.168.0.122. If your computer is on the same network you can surf to that address and immediately see the main web page appear. If you want to change the IP of the Tracker, you have two choices. One choice is to edit the source code file Webpage.cpp and load a new image into the Arduino. The other choice is to temporarily change your computer’s network to 192.168.0.0 so you can surf to the default address above; use the Tracker web page itself to set a different IP; reboot the Arduino then change your computer’s network back to your desired setting.

Once your web page is accessible, use the Gimbal section at the bottom to experiment with the motion range of each axis. You may use either motor for azimuth or elevation. The minimum and maximum values are stored in EEPROM so they will retain their values through a power cycle. The tracking algorithm will never exceed these limits. **Special note:** due to a limitation imposed by the internal calibration algorithm, set the minimum and maximum travel on the azimuth motor so the antenna rotates no more than 540 total degrees.

Web Page Description

Turn on the Tracker controller and surf to its network address with any browser. You should see the web page shown in Figure 4.

The page is basically in two vertical portions. The top portion contains some housekeeping fields but mainly allows setting and inspecting the Two Line Elements used to define the motion of the satellite of interest. The bottom portion is a table showing detailed information for each of the Tracker subsystems of Target, GPS, Sensor and Gimbal. Look through the table carefully because it provides a lot of information and control capability; hopefully most fields will be self-explanatory. You will note that some of the data fields may be overridden. Using these turns off the automatic setting of these values and allows you to enter your own values.

Also available is a popup that displays an all-sky map, the pointing directions of the sensor and target, the next pass and the great circle angle between the target and the sensor. You must disable popup blockers to see this map. Clicking on the map will slew the gimbal to point in that direction if tracking is enabled. Since the Tracker never computes information in the past, if you override the time and jump into the middle of a pass in progress, only the path from that moment onward will be drawn.

Now for a more detailed description of the web page. Across the very top is the title. Hovering over this title for a moment will display the software version. To the left is the network **IP** address of the Tracker. If this value is edited and you click **Set**, a new value will be stored in EEPROM and will be used the next time the Tracker is powered up or rebooted. To the right is a button to

Reboot the Arduino, mainly for this purpose. Below the title is a general purpose message line. Look here for confirmations, additional information and general messages as you use the Tracker.

Besides this bit of housekeeping, the top portion of the page mainly allows you to enter and upload the **TLE** for the satellite you wish to track. There are two text areas for showing TLEs. The top-most text area, with the darker background, is read-only and displays the TLE currently loaded into the Tracker, if any. The text area just beneath, with the white background, is writable and can be set in any of three ways:

1. Copy/paste a TLE directly from another document, including name in first of three lines
2. Search a local file for the named satellite
3. Search the AMSAT or Celestrak web sites for the named amateur satellite (if the Tracker has Internet access).

In the latter two cases, the Tracker will search through the entire file or web page for the name you enter in the given text field, ignoring case and non-alphanumeric characters. The Tracker expects the file format to have the name on the line just before the TLE in typical fashion. If the satellite is found with a valid TLE, it will appear in the white text area.

With the satellite name and TLE in the white text area, the TLE is still just in your browser. To actually send it to the Tracker, click **Upload**. After successfully uploading a valid TLE, it will appear in the darker read-only text area. This is the TLE the Tracker will follow if you click **Start Tracking**. You can edit or even **Erase** the white text area all you want and it won't matter unless you Upload it again.

The lower portion of the page is the main table for monitor and control. The first table section is for the **Target** to be tracked. In the left half you will see observing details of the uploaded satellite elements at the time and location shown in the GPS section farther down. In the right half you will see information about the next pass. Note the Tracker never computes information in the past, so if a pass is already underway (the satellite is currently above the horizon) then the Next information for Rise will be for the subsequent whole pass, since that event has already occurred for a pass that is underway. Note you can override the computed Azimuth and Elevation. If tracking is enabled, this allows you to point your antenna at any desired fixed sky location.

Below the Target section of the table is the section for the **Spatial sensor**. This displays the Az and El it is measuring and reporting to the Tracker. It also displays the current temperature and the status of the system processor and each of the individual sensors. These individual status values can range from 0 through 3, where 3 is the best. The Tracker will always use the data but it is flagged as being **Uncalibrated** unless all system status values report at least 1. Procedures for calibrating each sensor are provided in the Bosch manual[x] but basically the sensor package will need to be moved around to different orientations to get all sensors at their best values. Use the pulse length override fields in the Gimbal section (see next) to perform these manual motions. Once all sensors report state 3, their associated internal calibration coefficients can be stored to EEPROM by clicking the **Save Cal** button. Once saved, these values will be restored each time the Tracker is powered up and all sensors will often come up immediately in state 3. Note that the magnetic sensor is very sensitive to local magnetic fields and iron objects, so if the Tracker is moved to a new location, it is recommended to perform the calibration again and store a new set of coefficients.

The **GPS** section shows the reported time and location, and also displays some quality metrics. HDOP is the Horizontal Dilution of Position[xi]. This is an indication of the accuracy of the latitude and longitude, the position values most important to the Tracker. HDOP values range from less than 1 which indicates ideal conditions, up to 20 or more which indicates location can be incorrect by 300 meters or more. The number of satellites used in the fix is reported, where four or more is desirable. If you don't have a GPS connected, it does not have lock or you just want to experiment, you can override the time, date, latitude, longitude and altitude to see the effect on the passes. If you enter these carefully, you don't need a GPS at all.

At the bottom of the table is the **Gimbal** section. These data are in units of raw pulse duration. If you are aware of how hobby servo motors function, you will recall they are commanded to a given rotation angle determined by the length of a pulse issued on their control line. Pulse durations vary by manufacturer and even from one device of the same model but a pulse length of 1500 microseconds is often considered to be the center of rotation and changes of roughly 500 in either direction rotate to each extreme. The Tracker normally calculates the pulse durations with the tracking algorithm, bounded by the indicated minimum and maximum limit values. Note you can directly set specific pulse durations for each motor if you wish, doing so will automatically disable tracking if it is enabled. This is fun but also important to determine the safe as-built motion limits of each axis. The limits are stored in EEPROM and used by the Tracker to avoid exceeding the limits of each servo motor.

Operation

After everything is set up and you are comfortable with the safe operation of the Tracker motions, you are ready to track a satellite. Set it up somewhere with a good view of the sky, turn it on, connect your radio, bring up the web page, load the TLE into the white text area and click **Upload**. To begin tracking the satellite, click **Start Tracking**. Now forget about pointing your antenna and just enjoy using your radio (but do tweak the rotation angle occasionally as necessary).

Table 1 - Bill of Materials

Item	Source	Approximate Price
Arduino Mega 2560	Amazon: SunFounder Mega 2560 R3, stock number B00D9NA4CY	$18
Wired ethernet shield	Amazon: SunFounder Ethernet Shield W5100 for Arduino, stock number B00HG82V1A	$16
WiFi router	Amazon: TP-LINK TL-WR702N Wireless N150 Travel Router, stock number B007PTCFFW	$20
GPS module	Adafruit: Part ID 746	$40
Servo controller	Adafruit: Part ID 815	$15
Bosch 9 DOF sensor	Adafruit: Part ID 2472	$35
Pan platform	ServoCity: model SPG785A-CM, 5:1 ratio	$100
Tilt stage	ServoCity: model SPT400 with HS-5685MH, 5:1 ratio	$110
Lift extension	ServoCity: parts 545354x2, 545352x2, 635120, 545385	$30
	Total	**$384**

[i] http://siarchives.si.edu/collections/siris_sic_8335

[ii] https://en.wikipedia.org/wiki/Microelectromechanical_systems

[iii] https://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml

[iv] https://elkantennas.com/product/dual-band-2m440l5-log-periodic-antenna

[v] https://en.wikipedia.org/wiki/Euler_angles

[vi] https://github.com/brainwagon/angst

[vii] http://www.amsat.org/amsat/articles/g3ruh/111.html

[viii] http://www.xephem.com

[ix] $G = 10 \log_{10} (41000/(BW_v * BW_h))$, where G is directivity in dBi and the BW are the vertical and horizontal half-power beamwidths; from Kraus, John D., *Antennas,* Second edition, page 26.

[x] https://www.bosch-sensortec.com/en/homepage/products_3/9_axis_sensors_5/ecompass_2/bno055_3/bno055_4

[xi] https://en.wikipedia.org/wiki/Dilution_of_precision_(GPS)