DeltaSat

Ground Station Documentation

Finn Carlsvi – April 2014

Software

There are two separate programs:

1. High level Python script which handles satellites' azimuth and altitude locations depending on NORAD data. This runs on the computer
2. Low level C code which handles the stepper motors on the tracker. This runs on an Arduino.

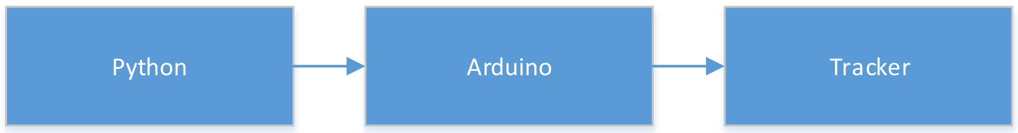


Figure 1 - Python sends azimuth and altitude commands to the Arduino, which controls and keeps track of the stepper motors.

Capabilities

The user can input multiple satellite identifiers (using NORAD ID only) into the high level Python program, ordered by priority. The program then downloads the satellite orbital elements from www.celestrak.com and converts the data from an equatorial coordinate system to a horizontal coordinate system. Then, the C program controls the stepper motors in order to track the satellite.

The tracking program does not:

- Handle any communication data
- Track balloons

High Level - Python script

The high level software used for the tracker is Python, which can be run through any operating system as a script. The python code handles the satellites to track and calculates the latitude, longitude, azimuth, and altitude of any satellite that exists in the NORAD database. Also, the python code displays a GUI with the orbits of the satellite, including when the satellite is in sight of the tracker.

Program Outputs

Program output to the screen can be controlled by setting the VERBOSE variable in constants.py to either True or False. Regardless of the setting, the program will always output to the file *tracker.log*.

Outputted data includes:

- Initialization data and constants from constants.py
- Tracking data in form of raw numbers
- Change of satellite tracking
- Warnings
- Errors
- Timestamp for each occurrence

The structure of the tracker.log file can be found in appendix A.
When debugging the program, always refer to this file.

Satellite Prioritization

When tracking multiple satellites, some might be visible in the sky at the same time. Therefore, it must be established which satellite should be prioritized in this instance. Whichever satellite is higher up in the *norad_id.txt* file will be prioritized. There are two types of prioritization modes, as defined in the **constants.py** file.

Hold

If HOLD is set to True in *constants.py*, then the tracker will finish tracking a satellite until the satellite moves below the horizon, regardless of if a higher priority satellite appears above the horizon.

No Hold

If HOLD is set to False in *constants.py*, then as soon as a higher priority satellite appears in the sky, the tracker will cancel its current tracking to follow the high priority satellite.

User Interface

The graphical user interface (GUI) displays the paths of the satellites that have been selected in the *norad_id.txt* file. The GUI exist to validate that the correct satellite is being tracked and that the tracker is pointing in the correct direction.

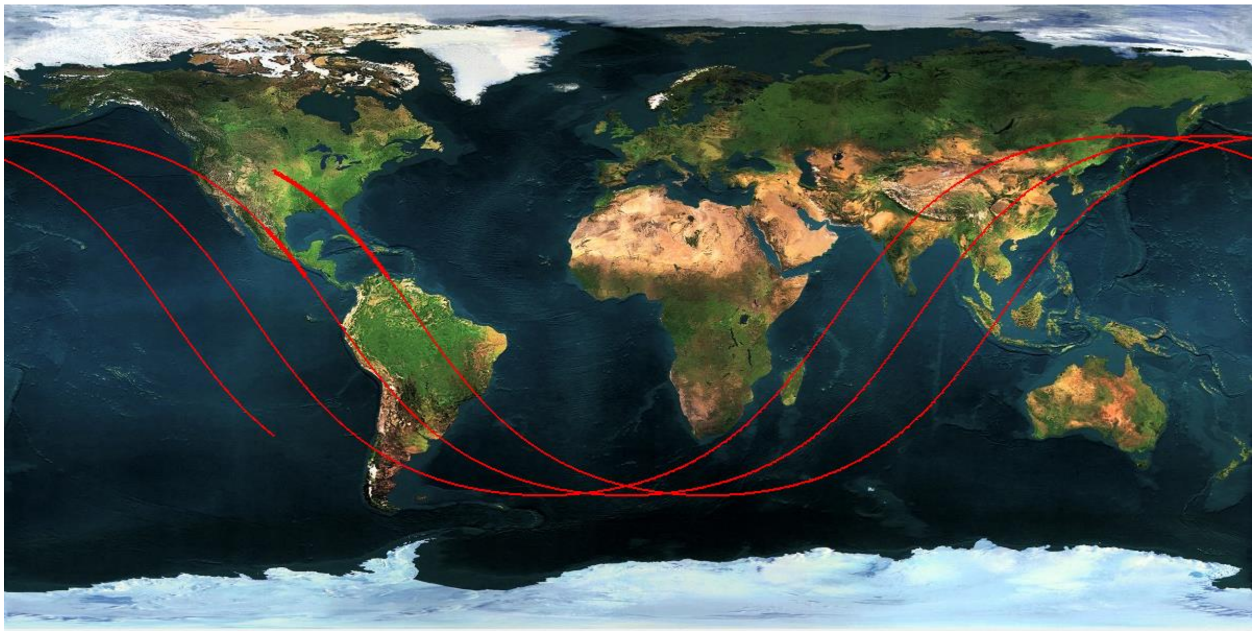


Figure 2 - Example of the python GUI tracking ISS. The bold line means that ISS is within line of sight of the tracker.

NORAD Data and Orbit determination

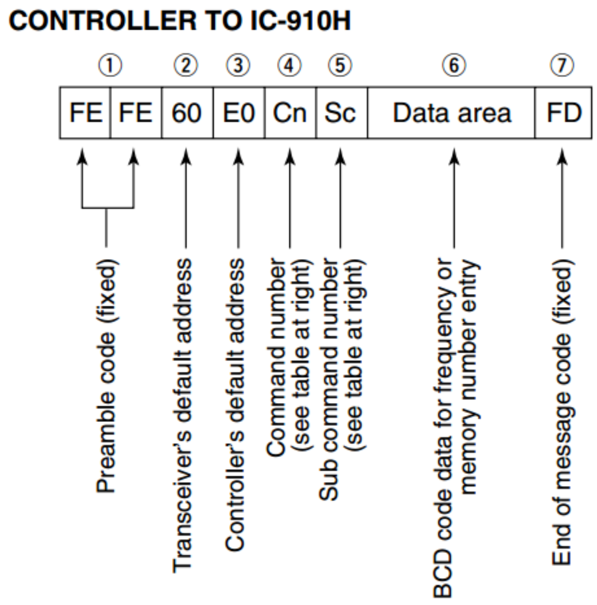
Orbital data is retrieved from CelesTrack (www.celestrak.com/NORAD/elements/) through a python script that downloads all NORAD data (update_norad.py). The NORAD data files are then downloaded into a folder named *norad_data*.

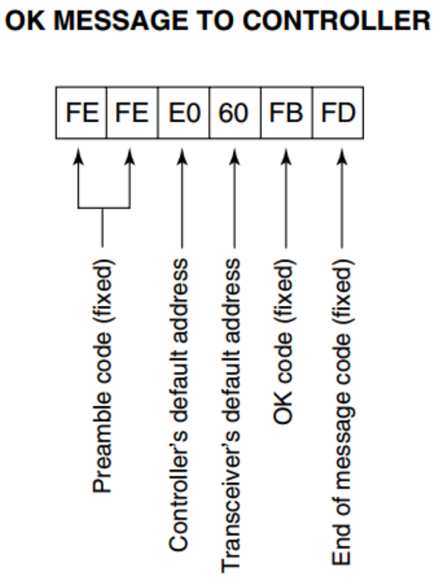
Next, the NORAD data is converted into latitude, longitude, azimuth, and altitude. For this, the PyEphem library is used (rhodesmill.org/pyephem/) which has a complete library that can take as input the NORAD two-line element data.

It is important to note that PyEphem uses UTC time as its standard, so *constants.py* must be edited during daylight savings.

ICOM connection

The python script connects to the ICOM radio through a DigiMaster CT17 CAT interface which connects to the USB port. Multiple commands can be send, which are found in the ICOM910H manual or on *www.plicht.de/ekki/civ/civ-p4.html*. Currently, the python script can change the frequency on the radio.


CONTROLLER TO IC-910H
①
②
③
④
⑤
⑥
⑦
FE
FE
60
E0
Cn
Sc
Data area
FD
Preamble code (fixed)
Transceiver's default address
Controller's default address
Command number (see table at right)
Sub command number (see table at right)
BCD code data for frequency or memory number entry
End of message code (fixed)


OK MESSAGE TO CONTROLLER
FE
FE
E0
60
FB
FD
Preamble code (fixed)
Controller's default address
Transceiver's default address
OK code (fixed)
End of message code (fixed)

The command to change the frequency is $05, and the structure of the frequency is in binary coded decimal format with the least significant byte first. For example, to send the frequency 145'282.376, the code would be:

$FE	$FE	to	fm	$05	**$76**	**$23**	**$28**	**$45**	**$01**	$FD

Much more information can be found on the website *www.plicht.de/ekki/civ/civ-p4.html*.

Arduino

The Arduino used is the Mega 2560. The reason an Arduino is used is because it is easier to use for prototyping. Currently, the pins used are the following:

Pin 7	Altitude limit switch (left)
Pin 8	Altitude limit switch (right)
Pin 9	Azimuth limit switch
Pin 10	Altitude stepper motor pulse
Pin 11	Altitude stepper motor direction
Pin 12	Azimuth stepper motor pulse
Pin 13	Azimuth stepper motor direction

The Arduino's main function runs once, setting all the pins and initializing the tracker via the limit switches. Initialization means that the tracker locates the direction that it is pointing by moving until it finds its limits, and then using those as a reference.

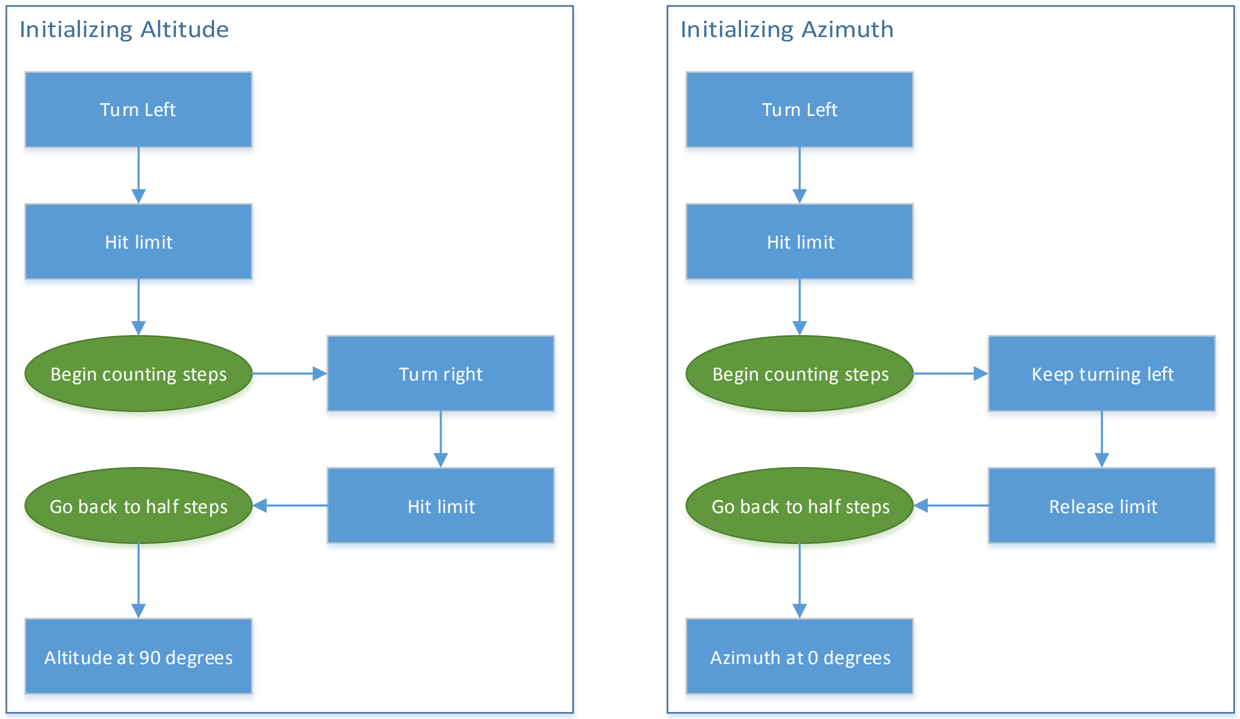


Figure 3 - How the initialization of the tracker works. The limit switches are used as reference points to zero the altitude and azimuth.

⚠ The Arduino restarts the program every time a connection is made from the computer. This will reset the program every time the Python script is run. Do not attempt to disconnect the reset enable ports on the Arduino, as this will cause problems when uploading new firmware.

Hardware

The hardware consists of the tracker, the mast that holds the tracker, and the power supply and stepper drivers.

Tracker

The tracker was custom designed and is meant to replace the previous Yaesu 5500 that was located on Canaveral Hall. The new tracker has much finer speed control and precision, with added torque. It uses NEMA34 standard motors with 10:1 reduction gearing, giving it 110 Nm holding torque and 0.09 degree accuracy. At 500 steps per second.

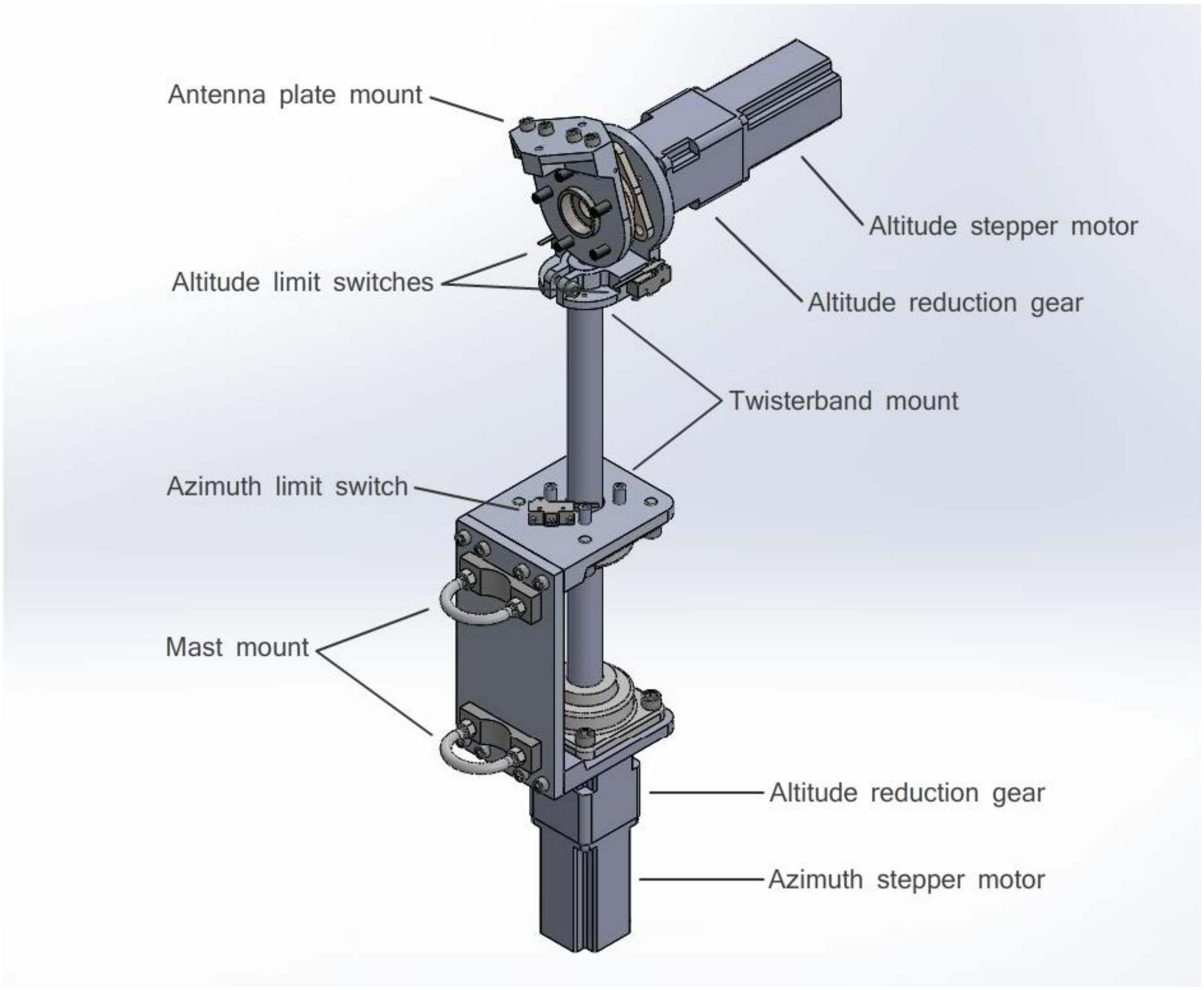


Figure 4 - SolidWorks view of the tracker, excluding the IGUS twisterband.

⚠ When adjusting any nut or bolt, note that blue loctite is used, not red. Replace the blue loctite as motor vibration can loosen screws and the weight and size of the tracker parts pose a serious risk to people below.

Control

Each stepper motor uses an independent stepper controller and power supply. The stepper motors have 8 leads while the controllers only output two phases each. Since speed is not important, the stepper motors are wired in bipolar series, giving them more torque.

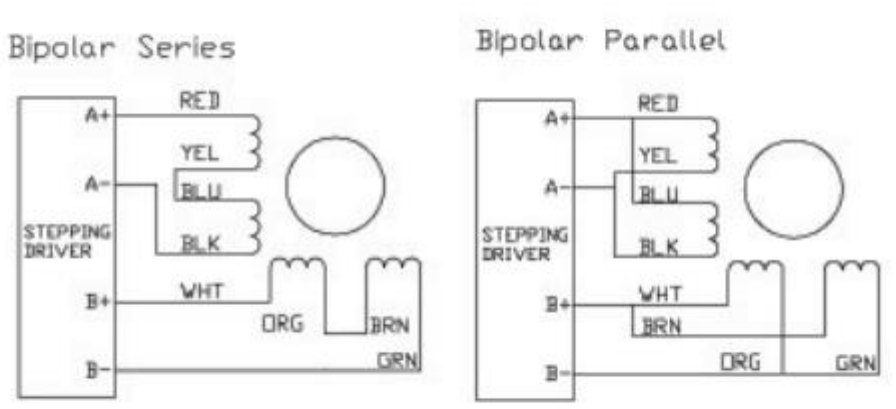


Figure 5 - The stepper motors are wired in bipolar series to provide more torque at lower speeds.

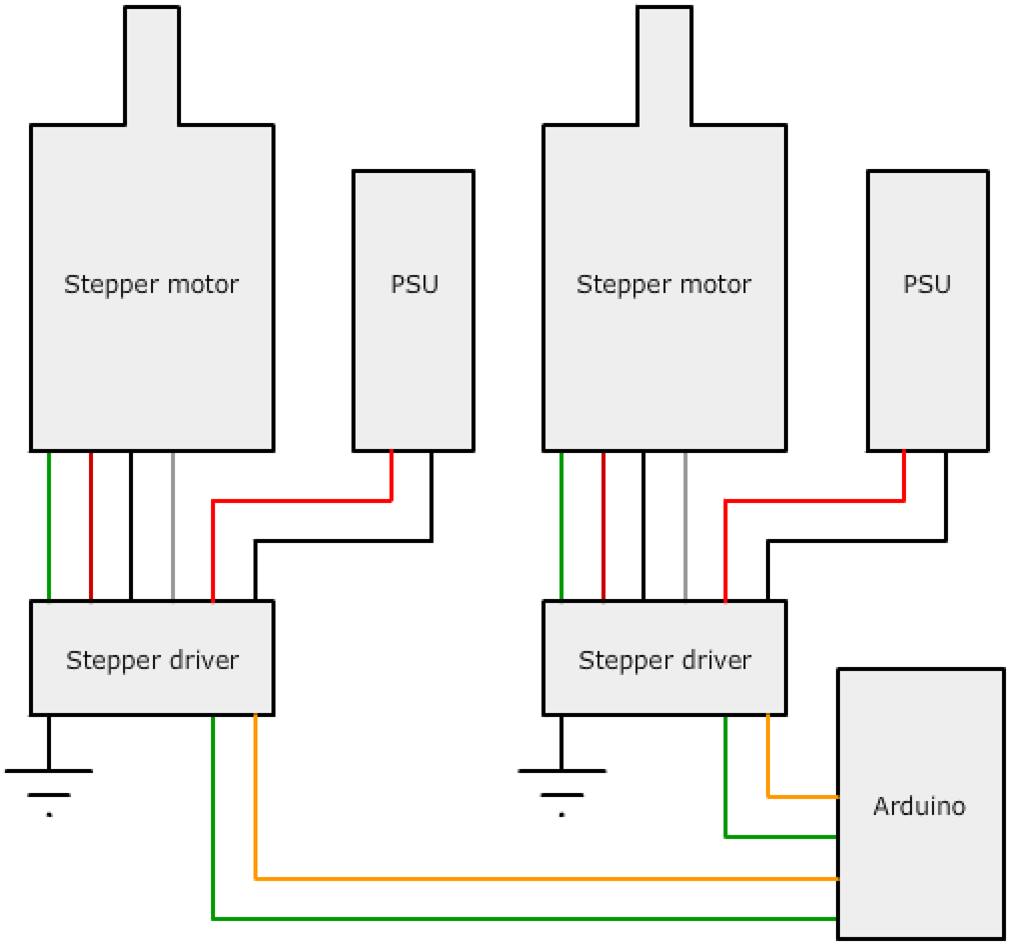


Figure 6 - Wiring diagram of the control system. Note that the color is accurate, but does not depict the 8 leads coming from each stepper motor.

Mast

The mast used for the tracker will be a Rohn 25G model standing 30ft on the southwest roof of the College of Arts and Sciences building. A mast must be used since the fifth floor, the emergency staircase, and some architectural aesthetics would otherwise obscure the antenna line of sight at a lower azimuths.

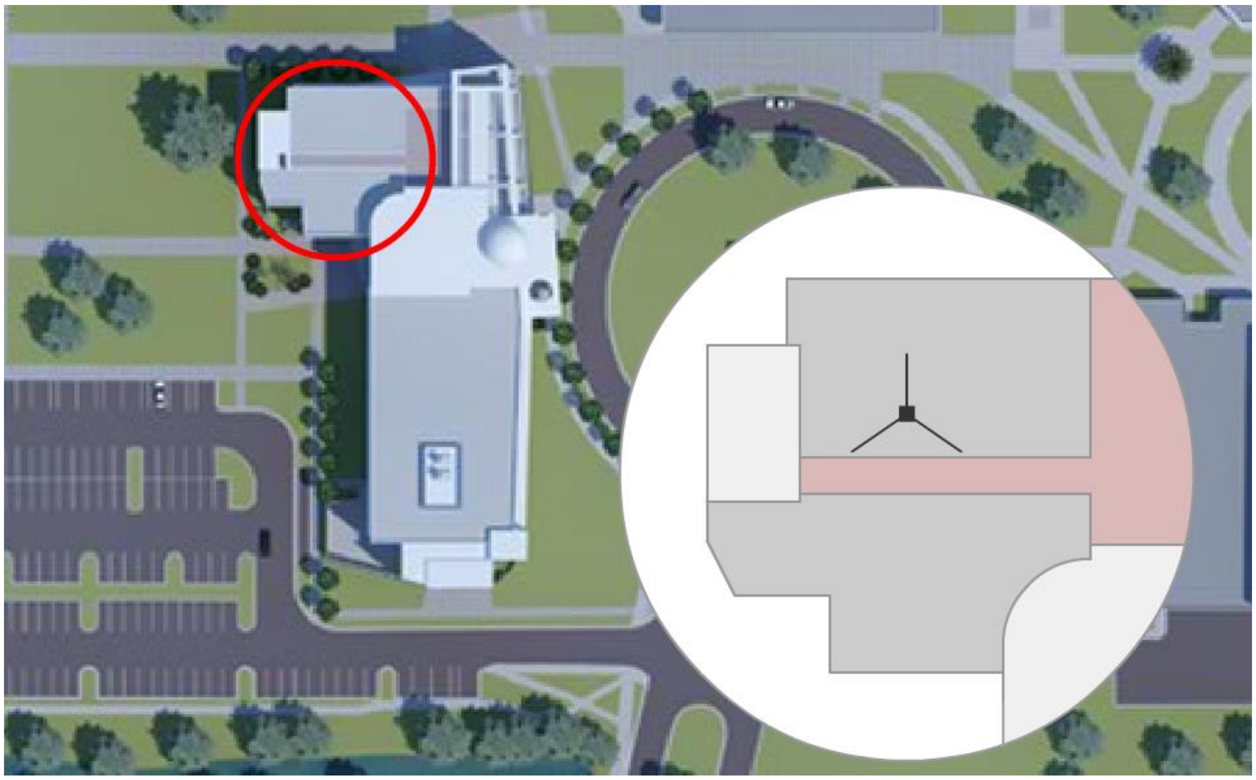


Figure 7 - Overhead view of the College of Arts and Sciences and a detailed diagram of the antenna location. The pink area is the patio, and the grey areas are fenced off.

The mast location depicted above is the only one available that provides a mount for the bottom plate and a cable port for the antenna and control wires. In addition to the mounting plate, it is a requirement that the mast be guyed. Rohn documentation requires that for the mast to withstand winds at 110 MPH, the 30ft tower requires three guys at 50 degree angles to the ground. It is due this in conjunction to the ground plate’s proximity to the fencing that 30 ft is the maximum height attainable.

Appendix A: File structures

norad_id.txt
Each 5-digit NORAD ID is separated by a newline, take care not to write this file in Windows style, where new lines are \r\n and not just \n.

Example:

```
12345,436.034,44.234
23456,436.034,44.234
34567,436.034,44.234
```

tracker.log
This is the main log file for the program. Data will be outputted here, regardless of if VERBOSE has been set to False in constants.py. The tracker.log file contains information on:

- Timestamps
  Each line will contain the date and timestamp in UTC time, of the form:
  'yyyy:mm:dd-hh:mm:ss '

- Initialization data
  All constants will be printed, on separate lines, in the form:
  '<constant_name> <value>'

  All satellite data will be printed on separate lines, in the form:
  '<norad_id> <norad_name> <elements FIX THIS>'

  The startup data on if the tracker.data file exists, and the azimuth, altitude, and revolutions of the tracker:
  'tracker.data <found/not found>'
  'azimuth <integer>'
  'altitude <integer>'
  'revolutions <integer>'

- Tracking data of satellite being tracked. Satellite real time position will be outputted to the log file, every amount of time that the constant OUTPUT_TRACK in constants.py determines:
  '<satellite NORAD ID> <azimuth> <altitude> <revolutions>'

Example:

```
2014:03:29-12:27:00 Initialized
2014:03:29-12:27:00 PATH /home/finn/tracker
Etc…
```

Appendix B: Program files and structure

This appendix outlines the structure of the program files. For an in depth view, open the actual files and follow the comments within the files.

main.py
This is the main program that is run by the user. It takes the NORAD IDs of the satellites you want to track as its arguments, separated by spaces. Lastly, it instantiates the Tracker class and starts the GUI.

Tracker.py
The tracker class handles the tracker and contains the satellites to be tracker. Since the entire software is very small, the tracker class essentially maintains everything except the graphical user interface.

GUI.py
The GUI class creates the map on screen for validation of tracker operation.

constants.py
All essential data in the program is edited in the constants file, this includes:

```
MAX_REV = (the maximum amount of rotations that the Twisterband can handle)
PATH = (the root path to the program)
LAT = (latitude of the tracker)
LONG = (longitude of the tracker)
ZULU_OFFSET = (hours behind UTC time)
COM_PORT_ICOM = (port for the ICOM communication)
COM_PORT_ARD = (port for the arduino)
VERBOSE = (sets whether data is output to the command line)
```

log.py
Whenever something is logged, or if there is an error or warning, the logging is handled by functions inside log.py.

Radio.py
This class handles the radio, changing the frequency and maintaining that the correct frequency is set.

Folder Structure

Program root folder

- **tracker_data/**
  - **norad_data/**
    - *amateur.txt*
    - *cubesats.txt*
    - etc…

  - **archived_logs/**
    - *2014_03_29-13_54_00.log*
    - etc…

- *norad_data.log* (log of when each txt file was last downloaded)
- *tracker.log* (log of the current tracker software)
- *tracker.data* (saved data on the tracker position)

Appendix C - Links

The ICOM910H instruction manual:
http://www.icomamerica.com/en/products/amateur/satellite/910h/default.aspx

Further information (specifically on ICOM USB control:
http://www.plicht.de/ekki/civ/civ-p0a.html

PyEphem:
http://rhodesmill.org/pyephem/
If the precompiled version of PyEphem does not work, it can be found for more OS:
http://www.lfd.uci.edu/~gohlke/pythonlibs/#pyephem

Terminal Node Controller KPC-9612 Plus manual:
http://www.kantronics.com/documents/KPC-9612PMX_Manual.pdf
http://www.kantronics.com/documents/kpckwm9612ppinout.pdf

CelesTrak elements (NORAD data):
http://www.celestrak.com/NORAD/elements/visual.txt

Rohn Mast 25G Brochure and Design Considerations
http://www.rohnnet.com/rohn-25g-tower

Stepper motor specifications
http://www.motiontek.ca/pdf/M1698.pdf

Stepper driver documentation
http://www.motiontek.ca/pdf/D80.pdf

Gear reducer specifications
http://www.motiontek.ca/pdf/GR-3410H.pdf