Django as a Mission Planning Tool Interface for the CYGNSS Mission

Tim Ewing
Southwest Research Institute
1050 Walnut St, Suite 300
Boulder, CO 80302
tim_ewing@boulder.swri.edu

Jillian Redfern
Southwest Research Institute
1050 Walnut St, Suite 300
Boulder, CO 80302
jillian.redfern@swri.org

Amanda Alexander
Southwest Research Institute
1050 Walnut St, Suite 300
Boulder, CO 80302
alexander@boulder.swri.edu

Richard Medina
Southwest Research Institute
1050 Walnut St, Suite 300
Boulder, CO 80302
medina@boulder.swri.edu

Emma Birath
Southwest Research Institute
1050 Walnut St, Suite 300
Boulder, CO 80302
emma.birath@swri.org

Abstract—**The successful operation of spacecraft requires careful planning. Each mission comes with a unique set of challenges that must be met by tools and techniques developed on a mission-specific basis. In the past, these tools have been created as Command Line Interfaces (CLIs), Remote Command Line Interfaces (Remote CLIs), or Graphical User Interfaces (GUIs). This paper presents a method for the development of these custom tools implemented for the mission planning of the CYGNSS (Cyclone Global Navigation Space System) mission: using the Django web framework to act as a remote Graphical User Interface.**

CYGNSS, the NASA Earth Venture Class mission which launched in late 2016, is a constellation of eight microsatellites in low Earth orbit which perform ocean wind speed measurements using reflected GPS signals to aid in weather modeling. Several tools have been developed to aid in the extensive and ongoing workload in mission planning. Now operating in the extended mission phase, the CYGNSS team is small. However, each team-member works on several projects aside from CYGNSS and performs the necessary mission planning and operations from their individual computer. Since, when using traditional tools, each operator's computer generally has a different combination of hardware, software, and operating system, developers are often required to perform custom installation and debugging for each new user of the system. This results in a system which is prone to user-specific bugs, and is not suitable for the low-cost environment in which CYGNSS operates.

We present a method which alleviates these problems. We use Django, a Python-based website framework, to host a suite of mission planning tools on a local website. This framework is split into three components: the Object Relational Mapping (ORM), the Template, and the View. The Django ORM is used to access the backend database from Python. Django Templates are used to control how the tool is displayed to the end user. Django Views tie the previous two components together by taking a request from the user, retrieving data from the database, rendering a template using that data, and returning the rendered template to the user where it is displayed by a web browser. We show how each of these components is used and the benefits of using such a web based system over a traditional tool.

978-1-7281-7436-5/21/$31.00 ©2021 IEEE

TABLE OF CONTENTS

1. BACKGROUND

The CYGNSS Mission

CYGNSS is a constellation of 8 microsatellites in low Earth orbit. Each satellite is approximately two meters long across the solar panels (Figure 1). Although the satellites lack thrusters, they are able to change position in orbit relative to each other by performing a high drag maneuver where the broad side of the solar panels is rotated to face prograde, slowly lowering the orbit. Each satellite contains a Delay Doppler Mapping Instrument (DDMI): a set of sensitive antennae that track ocean wind speeds (Figure 1) through the scattering of GPS signals reflected off the surface of the ocean. Communication with the satellites is not constant; the satellites can only downlink data to the ground when visible from one of three Swedish Space Corporation (SSC) ground stations located in central Chile, western Australia, and Hawaii.

The vast majority of the mission planning for CYGNSS is in support of nominal data downlinks. During each approximately 10-minute ground station overpass (hereafter referred to as a 'pass'), automated software in the Mission Operations Center (MOC) checks the status of the spacecraft, downlinks the most recent 24 hours of recorded data, and replays any data missing from previous passes.

We nominally plan passes four weeks in advance in 7-day increments. These passes are are generally not staffed by flight controllers. Occasionally, however, the science team will request additional data or a spacecraft will enter safe-mode, usually due to a radiation strike in the South Atlantic Anomaly (SAA) (2). These events require additional mission planning on short notice and must be staffed by a flight controller.

The CYGNSS MOC is located within Southwest Research Institute's Planetary Science Directorate in Boulder, Colorado. It is staffed by six personnel, none of whom work

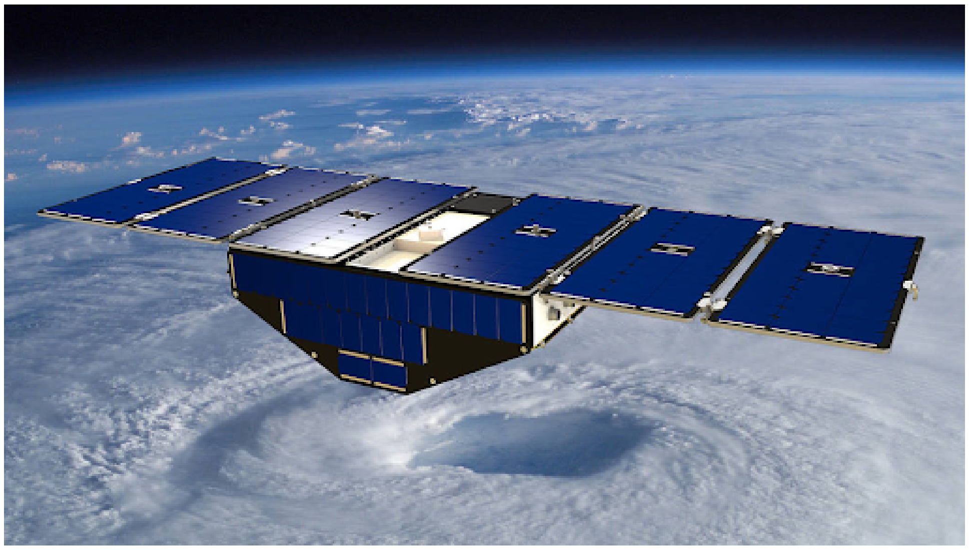


Figure 1. **Computer rendering of a deployed CYGNSS satellite.**

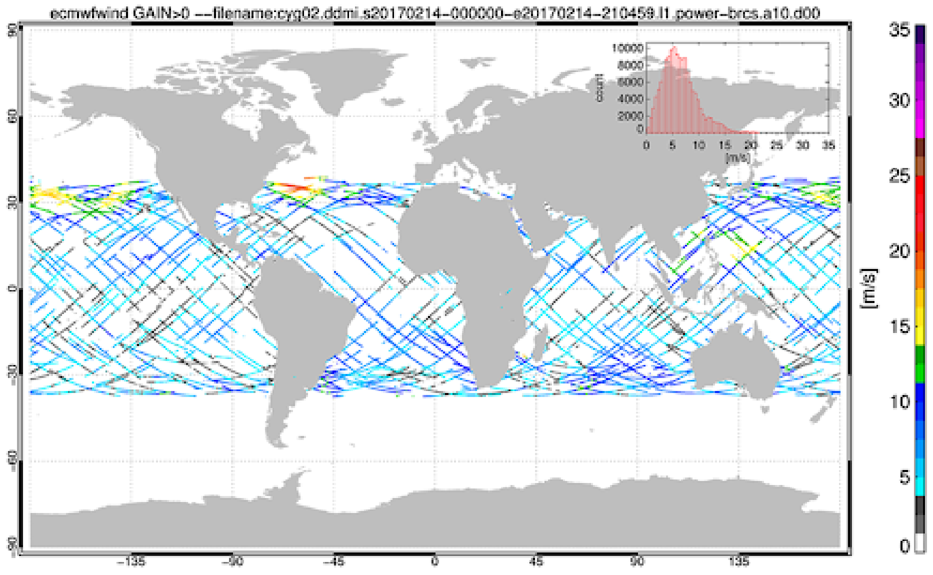


Figure 2. **Example of CYGNSS data (1).**

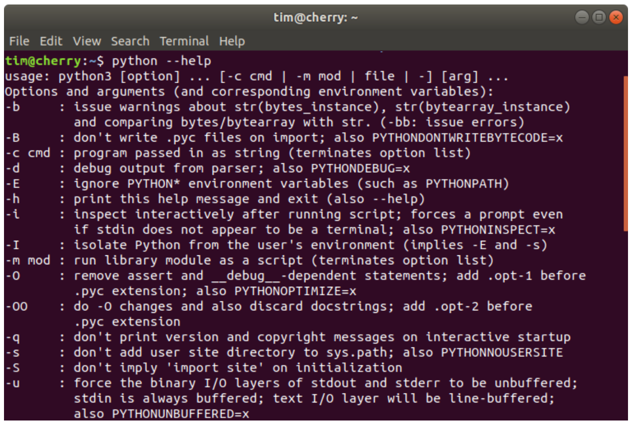
exclusively on CYGNSS. As such, the majority of CYGNSS operations must be automated. Staff are alerted by an automated system when faults occur on the spacecraft so that contingency operations can be executed, but for standard operations data playback and processing is managed automatically.

2. Traditional Tools for Mission Planning

Command Line Interfaces

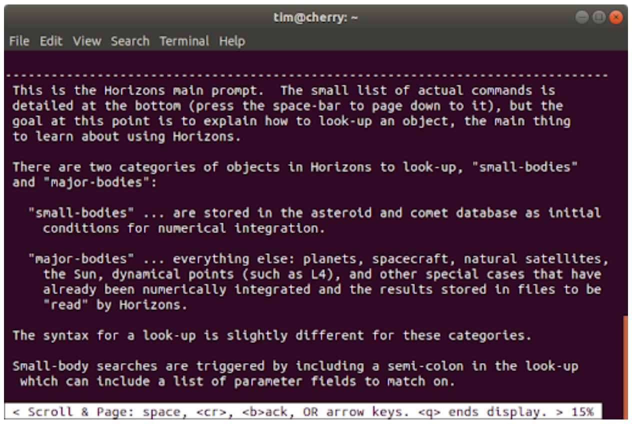
CLIs (Figure 2 are the default form of interface for custom tools. Every modern programming language has a standard method for input and output from the command line, so creating a CLI for a tool is trivial. However, the types of input and output from these interfaces are limited to text and very basic graphics. Features as basic as colored text or tables can require significant work and result in inconsistent visual results. Further, a graphical interface through the command line is infeasible; all input must be entered as text, and most programs only allow user input once as arguments to the program call. Examples include utilities like grep, date, and time, all of which are included in most modern unix operating systems.

Another complication with CLIs is the tedious setups required per user. Each operator must install not only the tool, but for non-compiled languages like Python, must also install an interpreter and any associated libraries – all of which are typically version-sensitive. For compiled languages, like C++, only the installation of the tool is required. However, this does not necessarily result in less installation overhead since the tool must be compiled differently for different software and hardware combinations. For both non-compiled and compiled languages, each computer must be updated any time the tool changes. There is also significant learning



```
tim@cherry:~$ python --help
usage: python3 [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-b     : issue warnings about str(bytes_instance), str(bytearray_instance)
         and comparing bytes/bytearray with str. (-bb: issue errors)
-B     : don't write .pyc files on import; also PYTHONDONTWRITEBYTECODE=x
-c cmd : program passed in as string (terminates option list)
-d     : debug output from parser; also PYTHONDEBUG=x
-E     : ignore PYTHON* environment variables (such as PYTHONPATH)
-h     : print this help message and exit (also --help)
-i     : inspect interactively after running script; forces a prompt even
         if stdin does not appear to be a terminal; also PYTHONINSPECT=x
-I     : isolate Python from the user's environment (implies -E and -s)
-m mod : run library module as a script (terminates option list)
-O     : remove assert and __debug__-dependent statements; add .opt-1 before
         .pyc extension; also PYTHONOPTIMIZE=x
-OO    : do -O changes and also discard docstrings; add .opt-2 before
         .pyc extension
-q     : don't print version and copyright messages on interactive startup
-s     : don't add user site directory to sys.path; also PYTHONNOUSERSITE
-S     : don't imply 'import site' on initialization
-u     : force the binary I/O layers of stdout and stderr to be unbuffered;
         stdin is always buffered; text I/O layer will be line-buffered;
         also PYTHONUNBUFFERED=x
```

Figure 3. **The result of the 'python –help' command, typical of a CLI**



```
----------------------------------------------------------------------------
 This is the Horizons main prompt.  The small list of actual commands is
 detailed at the bottom (press the space-bar to page down to it), but the
 goal at this point is to explain how to look-up an object, the main thing
 to learn about using Horizons.

 There are two categories of objects in Horizons to look-up, "small-bodies"
 and "major-bodies":

   "small-bodies" ... are stored in the asteroid and comet database as initial
     conditions for numerical integration.

   "major-bodies" ... everything else: planets, spacecraft, natural satellites,
     the Sun, dynamical points (such as L4), and other special cases that have
     already been numerically integrated and the results stored in files to be
     "read" by Horizons.

 The syntax for a look-up is slightly different for these categories.

 Small-body searches are triggered by including a semi-colon in the look-up
  which can include a list of parameter fields to match on.
< Scroll & Page: space, <cr>, <b>ack, OR arrow keys. <q> ends display. > 15%
```

Figure 4. **The TELNET Remote Command Line Interface for JPL's small body SPK generator (6).**

required for each user. A basic understanding of the terminal commands is critical and can be expected from technical users like developers. However, many users may find the terminal interface unwieldy and may have difficulty learning to use the tool.

Remote Command Line Interfaces

Remote CLIs (Figure 2) are standard CLIs that have been set up on an external server which is accessible through SSH, TELNET, or a similar alternative. Many of the benefits of a CLI are retained with Remote CLIs. Generally, a program is able to accept user input in the same way as if it were a local CLI. A user establishes a connection with a remote machine allowing input to be sent to the remote machine and the response displays within the user's terminal. The rest of the user interface is the same; all input is via text commands and often input is only accepted once at the beginning when calling a program.

Although Remote CLIs alleviate some of the setup for each user, they do not fully solve the problems with a command line-based tool. The central nature of the codebase allows easier management of tool versions since only a single machine must be updated when the tool changes. However, the requirement for technical expertise is not resolved. Users must still be familiar with a terminal, and will still be limited to purely text-based input and output from the tool. Remote CLIs also raise new problems with file manipulation. With

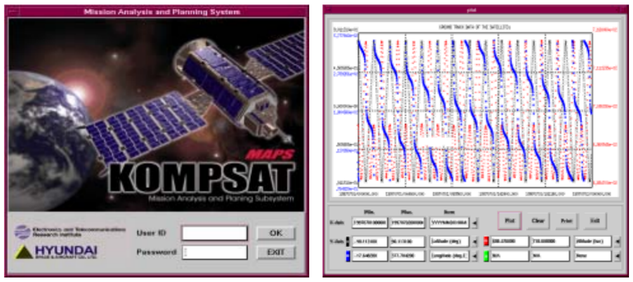

Mission Analysis and Planning System
MAPS
KOMPSAT
Mission Analysis and Planning Subsystem
HYUNDAI
User ID
Password
OK
EXIT
Plot
Clear
Print
Exit

Figure 5. **GUI for MAPS, a mission planning tool for KOMPSAT-I (5), a low earth orbit remote-sensing satellite comparable to CYGNSS.**

a standard CLI, the tool will have access to the filesystem of the user by default. However, with a Remote CLI, there is extra work required using a protocol such as SFTP (SSH File Transfer Protocol) or FTPS (File Transfer Protocol for SSL). There are also inconsistencies within the tools for accessing Remote CLIs. Functionality that users are used to, such as copy-paste and scrolling, often break when accessing a remote terminal due to inconsistent standards or implementations of protocols.

Graphical User Interfaces

Traditionally, when a tool needs to be used by a wider audience, developers create a Graphical User Interface (GUI) (Figure 2). This is a custom interface that uses both mouse and keyboard input to create a more natural user experience. The output of the program is generally interactive, allowing the user to see in real-time the results of their actions. Some examples of programs with GUIs include file explorers, Adobe Photoshop, and the Microsoft Office suite.

GUIs are powerful alternatives to CLIs. Patterns in data which are hard to discern when using a text-based interface are made more apparent when the data is displayed visually. For example, the periodic nature of the data displayed in the right side of Figure 6 would not be obvious when listed as a series of numbers on a command line. Moreover, GUIs have a much shallower learning curve. Users familiar with the use of a computer should have little trouble making inferences about how a GUI operates when implemented correctly.

However, GUIs still have some disadvantages. Often, installation comes with many headaches for developers. Graphics libraries are not well standardized, often leading to the need for platform-specific fixes for each user. Sometimes, the libraries are unavailable on other operating systems, leading to even further problems for developers. These tools can also require access to data stored on external servers, forcing the need to develop an access control scheme. Finally, the implementation of user and password management is not trivial and mistakes have severe security implications.

3. Django as a Web Based Tool Framework

What Is Django?

Django is a Python-based web framework used to create websites and is designed to allow for the rapid development of modern web applications. It includes a built-in backend database with ORM to facilitate fast and flexible database management. This database interface includes built-in user

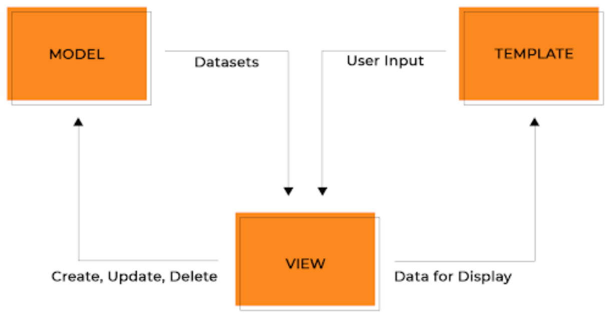


Figure 6. **Django MVC data flow diagram.**

```
{% for some_item in some_list %}
  <td> This is {{ some_item }}.</td>
{% endfor %}
```

Figure 7. **A Django template segment before rendering.**

```
<td> This is one.</td>
<td> This is two.</td>
<td> This is three.</td>
<td> This is four.</td>
<td> This is five.</td>
```

Figure 8. **A Django template segment after rendering.**

and password management, allowing developers to bypass the time-intensive and difficult task of creating a secure user management system. The database can also be managed through the Django Admin interface, a web-based database viewing and managing tool. Django includes a test server which allows developers to get a working website running within the first hour of starting a new Django project. Modern website development styles can be easily applied since the tools are being developed on a web platform, and standard tools such as Jquery and Bootstrap can be easily integrated to give the tool a modern feel with minimal effort. These features are all well-documented by Django and a large online community exists to help solve any problems that arise.

The Django Framework

The Django architecture can be loosely compared to a standard Model-View-Controller (MVC) framework. Under the standard architecture, the Model is responsible for communication between the developer's code and the database. The View is a collection of HTML templates that are later rendered; each template is a standard HTML file with placeholders for data that needs to be filled in by the Controller. Finally, the controller connects the View and the Model; when a page is requested, the user's data is passed to the Controller from the server, the Controller uses data from the Model to render a View, and the rendered view is passed back to the server to be displayed to the user.

Django's alternative to the standard MVC framework consists of three comparable components (Figure 3). The Django ORM is the equivalent to the Model in a standard framework. It abstracts the actual creation of database tables by representing them with Python code. Django uses HTML templates and a templating engine as its version of the View. It supports both a built-in Django template engine and Jinja2. Finally, Django adopts Python functions as the Controller. Each function written by the developer is called with a Django request object as an argument by the Django server and must return a Django HTTPResponse object. Confusingly, the functions for the Django controller are usually stored in the file 'views.py'.

The Django ORM

The Django ORM is a system that allows developers to create and manage a database using Python classes. Each table in the database is represented by a Python class and each column in the table is represented by a field in the class. New objects can be added to the database by creating an instance of the class, assigning values to the fields, and calling the save() method. Django handles the conversion of standard Python types to data that can be stored in the database, so developers are able to write code generically and switch between specific backends later. Official support is provided for PostgreSQL, MySQL, SQLite, and Oracle databases. Changes in the structure of the database are tracked by migrations, allowing developers to preserve data when changing the backend. Models are usually stored in the file 'models.py'.

Django Templates

Django templates inform the program how to display data to the end user. The templates are standard HTML files with placeholders for any field that needs to be passed in by the Controller. Developers have the ability to use loops and conditionals to change how data is displayed on the screen for the user which eliminates repetition in the HTML templates leading to more readable code. Developers can replace large chunks of repetitive HTML with a single loop (Figures 3 and 3). Django also allows developers to use and create filters for data passed into templates. For example, replacing name with name—lower would result in the name variable being input in lowercase. Django also has built in support for Jinja2, an alternative Python templating engine (4).

Django Views

The Controller in Django is confusingly represented by Django Views. A View in Django is a Python function which is passed as a request object by the server and must return a Django HTTPResponse. Views have access to Django Models in order to retrieve the data required to fill in a Django template. Queries are performed on the database using the function ModelName.objects.filter, where ModelName is the name of the Django Model in models.py. Once a query has been performed, developers can iterate over and select from the resulting QuerySet using standard Python code, treating the QuerySet as if it were an ordinary list. The retrieved data is then used to construct a context (a standard Python dictionary). Each key in the dictionary must correspond to a variable within the template and the dictionary value must contain the renderable value. Then, the template must be rendered by the View. This can be done many ways, all of which are well explain in the Django documentation.

Admin Interface

Django includes an interface for managing the backend of the server (e.g. create, edit, delete database entries) without using the command line. The interface is accessed by navigating a web browser to website.com/admin where website is the URL of the server. For the demo server, this is typically localhost:8000.

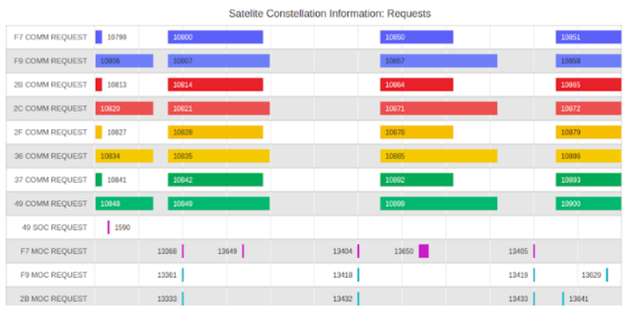


Figure 9. **The CYGNSS Gantt Chart Interface.**

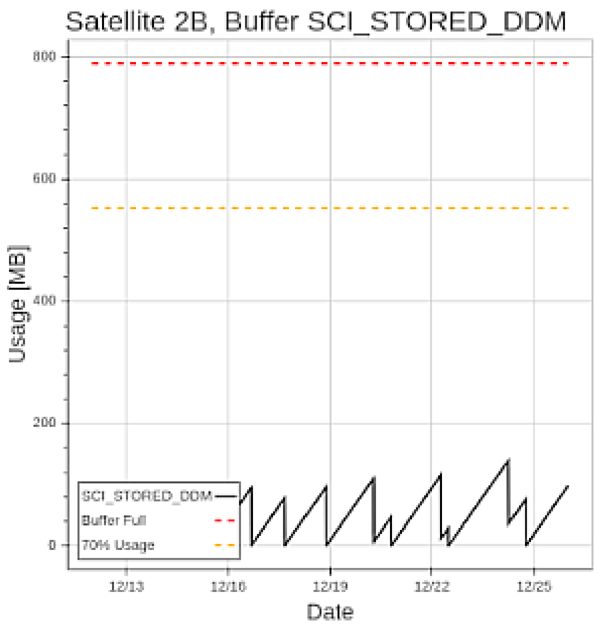


Figure 10. **The CYGNSS Resource Tracker showing the science buffer for satellite '2B'. Each dip in the sawtooth corresponds to data playback during a ground station overpass.**

User Management

The Django backend includes a system for managing users and their passwords. This system is an especially useful part of the Django framework. Securely storing passwords is a difficult problem. In Django, passwords are stored securely according to modern best practices (3) and the type of encryption algorithm used can be easily changed. Plaintext passwords are never stored in the database. Logging in a user is trivial as the developer only needs to make a login page template, store it in the correct folder on the server and link it to the built-in Django View.

4. Django and CYGNSS

CYGNSS utilizes Django to create high-quality tools for mission planning. Two of these tools, 'Gantt Charts' and the 'Resource Tracker', are presented below.

Gantt Charts

Gantt Charts are time-ordered charts that represent the schedule of a project or series of events. We use Gantt Charts to plan ground station contacts for CYGNSS to ensure that there is no overlap with other existing contacts or planned activities (Figure 4). These charts are fully interactive: as a user hovers over a specific task, additional task information is displayed and the tasks can be directly edited via a separate dialogue that appears upon clicking the task. The visual nature of this tool allows users to quickly ensure that no tasks conflict.

Resource Tracker

The Resource Tracker informs users about resource usage on the spacecraft based on previously-downlinked data. Several values can be monitored, but the spacecraft battery state and data buffers are used most frequently. Figure 4 depicts an example of the graphs produced by the Resource Tracker. Fully interactive, the exact value at a given point on the graph is given when hovered over and clicking will open a separate page depending on which resource is being tracked.

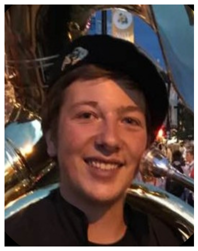
5. Conclusion

The development of high-quality tools that can be quickly developed is critical to the ongoing operations of spacecraft missions. These tools now need to be made more accessible to less technical users. Traditionally, command line tools or custom graphical tools have been developed. However, command line tools require a high level of expertise and graphical tools are difficult to develop and often come with problems when deployed to various different machines. Our method, using the Django web framework, we develop web-based applications for CYGNSS-specific mission planning tasks which alleviates the issues described as well as reduces the work required to implement many common features. Django is widely used, open-source, and well documented–and should be the focus for developing future mission planning software for large, distributed teams.

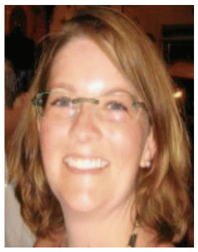
References

[1] https://www.nasa.gov/feature/nasa-s-cygnss-satellite-constellation-enters-science-operations-phase
[2] https://heasarc.gsfc.nasa.gov/docs/rosat/gallery/misc_saad.html
[3] https://docs.djangoproject.com/en/2.2/topics/auth/passwords/
[4] https://jinja.palletsprojects.com/en/2.10.x/
[5] Won, C.-H., Lee, J.-S., Lee, B.-S. and Eun, J.-W. (1999), Mission Analysis and Planning System for Korea Multipurpose Satellite-I. ETRI Journal, 21: 29-40. doi:10.4218/etrij.99.0199.0305
[6] https://ssd.jpl.nasa.gov/?horizons#telnet

Biography



Tim Ewing *is an undergraduate studying Engineering Physics at the University of Colorado Boulder. He has worked to develop autonomous snow-clearing robots at Left Hand Robotics, and is currently a part-time Flight Controller and software developer at Southwest Research Institute for CYGNSS and LUCY in Boulder, Colorado.*



Jillian Redfern *received a Bachelor of Science in applied mathematics from the University of Colorado at Boulder in 2001. She studied aerospace engineering at Massachusetts Institute of Technology. She currently serves as the Missions Operations Manager (MOM) for the NASA CYGNSS mission. She is a Section Manager at Southwest Research Institute in Boulder, Colorado.*

Amanda Alexander *is an Analyst at Southwest Research Institute. She has been involved with the NASA CYGNSS mission since 2017 as a flight controller and mission planner. Amanda is also a Student Collaborator for the NASA Psyche mission and studies impacts into small bodies like Asteroid (16) Psyche. She received a B.A. in Astrophysical and Planetary Sciences at the University of Colorado Boulder in 2018 and is now a PhD Student in the department of Geological Sciences.*

Richard Medina *is a Senior Research Analyst at Southwest Research Institute in Boulder, Colorado. He has 10 years of space operations experience in remote sensing missions ranging from low earth orbit to deep space. He is currently a mission planner for the NASA CYGNSS (Cyclone Global Navigation Satellite System) mission as well as a software developer for the CYGNSS and Lucy missions. He received a Bachelor's of Science in Physics from the United States Air Force Academy in 2008.*

Emma Birath *received her M.S. in Physics from Colorado State University in 2000, after which she joined the Imaging Science Subsystem team on the Cassini mission, as a sequencer and tool developer. Today she is a Science Operations Analyst for the New Horizons mission to Pluto and the Kuiper Belt. Emma is also the Deputy Mission Operations Manager on the CYGNSS mission (Cyclone Global Navigation Satellite System), where she oversees daily operations for the eight Earth orbiting satellites.*