CUAir Distributed Systems Documentation Documentation

Release 1.0

Rahul Madanahalli Ram Vellani Maria Sam Sameer Khoja Sam Rin

Jan 28, 2017

Contents

CHAPTER 1

Welcome

Yay, documentation!

1.1 How to edit this page

1.1.1 Easy Mode

- Click the "Edit on GitHub button at the top of any page"
- Edit the page
- Click 'Preview changes' before committing to make sure you haven't made a mistake with markdown syntax.
- Add a change message and commit when you have finished your changes
- In a few minutes, readthedocs will be updated

1.1.2 Leet Mode

Installation

```
$ git clone https://github.com/CUAir/DistributedSystemsDocs/
$ cd DistributedSystemsDocs
$ virtualenv venv
$ source venv/bin/activate
$ pip install -r requirements.txt
$ cd docs
$ make html
```

Use

- To create a page, make a new file called foobar.rst.
- Add your page to the table of contents in the file index.rst under the api and groundstation pages.
- Edit your page using sphinx markup.
- To see what your page looks like, compile it with `make html`
- When you are happy with what the page looks like commit your changes with git (`git add --all; git commit -m "change message"; git push`).
- The online documentation will update sometime in the next few minutes.

To learn more about how to use sphinx, see the following guides

http://www.sphinx-doc.org/en/stable/tutorial.html

http://www.sphinx-doc.org/en/stable/rest.html#rst-primer

1.2 Support

If you are having issues, please slack Sameer.

1.3 License

The project may or may not have a license.

Contents:

1.3.1 Ground Server

Contents

This section provides the use and design of the distributed systems ground server.

Overview

The CUAir ground server is designed to fulfill two tasks: target detection/localization and delivering a care package (airdrop). In order to fulfill these tasks, the ground server must keep track of and store various settings and states.

More importantly, it should be able to handle client requests reliably.

Full documentation of the 2016-2017 Ground Server API can be found here.

The ground server is built using the Play web framework in Java. It's an MVC framework that separates the logic for the view (our frontend), controller (API endpoints that allow clients/servers to communicate with us), and model (interfacing with the database layer, running any algorithms or business logic).

Ground Server Features Overview

- Tagging (Tagging page of UI):
  - TargetSighting creation
    - User tags an image by right clicking, moving the mouse until the red circle covers the entire target sighting. Make sure that red radius line is pointed towards the direction that the target is facing, and then left click again.
    - Check Target sighting in postgres database:
      - has an associated geotag
      - tagged portion of image is displayed in target sighting object
  - User enters the alphanumeric character, alpha color, shape, and shape color of the target into the respective fields within each target sighting object.
  - User can
    - save a target sighting
      - Saved target sightings from tagging page are displayed on merging page with the corresponding target information (alpha, alpha color, shape, shape color)
    - zoom in and out of image on tagging page by left clicking on the image
    - remove target sightings
    - use 'next' and 'previous' buttons to move from one image to the next
    - only view the images sent to his computer, not another client's computer
    - receive a new image from the ground server upon clicking 'next' when viewing his most recent image
      - This creates a new assignment (assigned to MDLC) in the database
  - When user is done processing (clicks next for another image), assignment is marked as done
  - All images sent to the ground are processed
- Merging (Merging page of UI):
  - All unassigned target sightings are displayed in the right sidebar
  - User can drag a sighting from the sidebar to an empty target or the "new target" field, and the sighting will be displayed in the target with the correct target information displayed.
  - User can:
    - drag a sighting to an existing target
      - Target object will be updated in postgres database
      - It's corresponding geotag is also updated
    - un-assign a sighting from a target by dragging the sighting from a target to the unassigned target list.

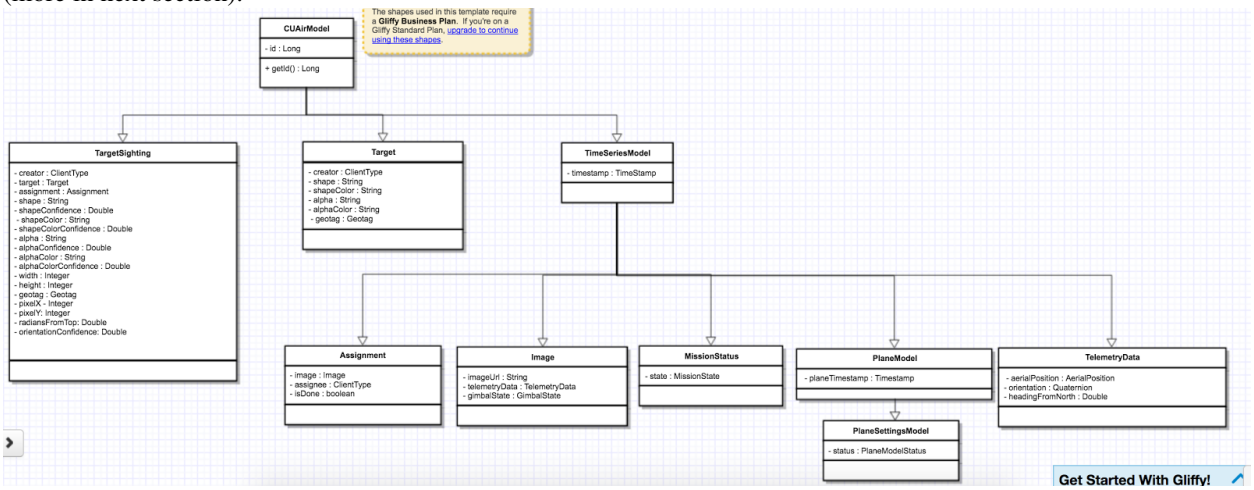
        * create an empty new target
            · Target object will be created in postgres database
        * remove one target at a time by clicking the "x"
        * save a target
        * export the targets (all at once) by clicking "export targets" at the bottom of the page, creating a .txt file with all of the target information
- Camera (+ Camera page of UI):
    - User can change (within appropriate ranges):
        * Brightness
        * Shutter Speed
        * Gain
        * Frame Rate
        * Whether the camera is capturing photos
        * User can save the camera settings and this will update the settings of the actual camera
        * Camera settings objects are being created in postgres database
        * User can see the most recent image captured by the camera
        * If camera is capturing images, they are being saved to /ground-server/plane/files/plane/
        * Image objects are being created in the postgres database
        * Corresponding telemetry and gimbal data objects are being created in the postgres database
- Airdrop Settings (+ Airdrop and Gimbal page of UI)
    - User can indicate:
        * Arm status
        * Target latitude
        * Target longitude
        * Target threshold
    - User can save airdrop settings
    - When user updates above fields, these settings are sent up to the airdrop server
    - Airdrop setting object is created in database
    - User can enable a manual airdrop override
    - User can override the airdrop (when arm status is true and manual airdrop override enabled) and the payload is immediately dropped
- Gimbal Settings (+ Airdrop and Gimbal page of UI)
    - User can indicate:
        * The longitude and latitude of a coordinate on the ground that the camera should point at (only valid when mode is gps)
        * The pitch and roll of the gimbal (only valid when mode is angle)
        * The mode of the gimbal ('retract', 'ground', 'gps', 'angle')

  - User can save gimbal settings
  - Gimbal settings object will be created in database
  - Settings will be sent up to gimbal server
- Geotag test (only during test flights)
  - Record GPS coordinates of targets in field
  - Make sure target GPS coordinates are within 50ft
- Reconnection
  - Settings are queued when ground server disconnected from plane servers
  - Once connection re-established, settings are sent back up to plane

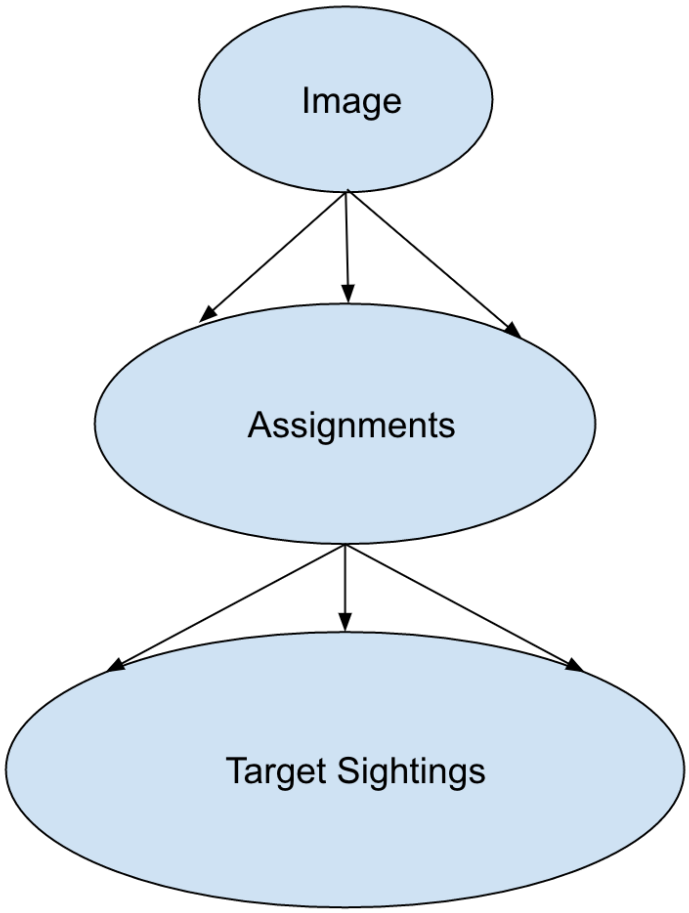
System Design

Models

Below is a class diagram of the ground server models. One can see the one-to-one as well as many-to-one relationships (more in next section).



Many-to-One Relationship



The above figure demonstrates the "one-to-many" relationships between the ground server abstractions. Each image has multiple assignments, which are distributed among various clients, and each assignment can have multiple target sightings.

While this accurately represents the relationship among our abstractions, our software design takes a different approach:

- Target Sighting
  - Assignment
    - Image
- Target Sighting
  - Assignment
    - Image
- Target Sighting
  - Assignment
    - Image

In this approach, we see that there is a "many-to-one" relationship between TargetSighting and Assignment, and between Assignment and Image. The reason we take this approach rather than "one-to-many" is that for one, many-to-one is much simpler and cleaner to serialize into an SQL database. Additionally, this design accurately represents the underlying operations of the ground server. Whether or not TargetSightings are added into an Assignment, Assignment

is only concerned with the Image to which it was assigned. Similarly, Image should not bother with Assignments, as it is only concerned with the image data itself.

The ground server models are used to store data in a SQL database through serialization. The ground server utilizes Ebean to handle this serialization. Ebean is an Object Relational Mapping (ORM), which is a Java library that allows us to execute SQL commands on our database tables.

Database Accessor Objects

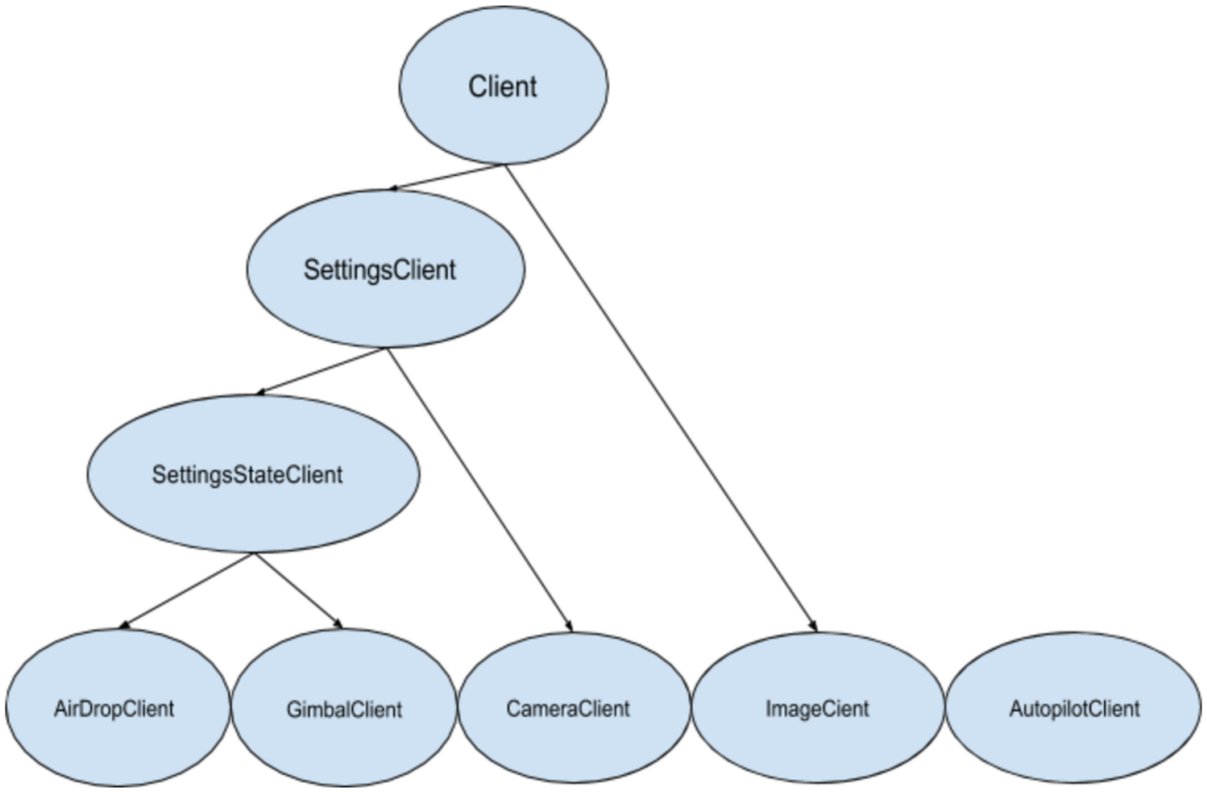
For each model in the ground server, there exists a database accessor object, or DAO. DAOs utilize Ebean methods to retrieve data from the SQL database. DAOs are an abstraction around accessing the the database from the controller, as many of the methods used to retrieve data are similar across the controllers (get, create, delete, update). The DAO combines these methods into one interface that allows us to handle these requests for any CUAirModel. When we want to make more complicated requests, we can simply extend the DAO and add the necessary method. (i.e. retrieving all target sightings for a certain image). The DAO abstraction is also useful as it prevents us from accessing the database directly. So, if we need to migrate to another ORM or library, we will simply need to modify the DAO rather than the controller code, which would be more complex.

Clients

The client abstractions are designed to process requests to get and set settings and state of the plane servers (Gimbal, Camera, Airdrop, Autopilot). Due to the possibility of a failed connection, the client abstractions include threads separate to the application thread that are meant to continue trying to send requests up to the server until a non-timeout response is received.

The underlying pattern with the Client abstractions is that each server on the plane (Gimbal, Airdrop, Camera) contains a client class which handles requests to set the settings, as well as to get the settings and/or state.

ImageClient is a unique case which involves obtaining information from Autopilot and the Gimbal in order to get the telemetry data for a particular image. Since all of the plane servers are on the same onboard computer, they have the same timestamp. This plane timestamp, therefore, can be taken from the Image and used in the queries in AutopilotClient and GimbalClient. ImageClient runs two parallel threads which attempt to get autopilot telemetry data and the gimbal state for an image, respectively.



Client Class Diagram The Client abstraction simply defines a thread that continuously executes run().

The SettingsClient abstraction contains a queue of requests and extends Client. The setSettings() method, which is called by the Client, will add the request to the queue and return a 200 response as an indication that the request was successfully received and is currently being processed. When it is run, it will poll the queue and attempt to send the request (if any) to the server. Once a 200 response is received in the thread, indicating that the settings were successfully sent to the server, the update gets reflected on the front-end. This is extended by CameraClient.

StateSettingsClient, which extends SetttingsClient, allows one to get state. This is extended by AirdropClient and GimbalClient.

AutopilotClient simply gets autopilot telemetry data at a particular timestamp and has no concept of changing the settings or state. Therefore, it is not extended by any client abstractions.

ImageClient is a unique case which involves obtaining information from Autopilot and the Gimbal in order to get the telemetry data for a particular image. Since all of the servers are on the same computer, they have the same timestamp. This timestamp, therefore, can be taken from the Image and queried for in AutopilotClient and GimbalClient. ImageClient runs two parallel threads which attempt to get autopilot telemetry data and the gimbal state, respectively.

Settings and States The "state" is information that the plane inherently knows that the ground server cannot directly change but can certainly query for. The plane settings, however, are directives of the plane and can be changed by the ground server. A change in setting can and does induce a change in state. The state and the settings breakdown for the plane servers as follows:

- **Airdrop Server**
  - State: Whether the drop has occured or not (the ground server can try to arm/override but only the plane knows whether the physical mechanism was activated)

  - Settings: Target latitude and longitude, acceptable threshold for drop accuracy, arm and disarm, override drop

- **Gimbal Server**
  - State: The quaternion values that the gimbal has assumed
  - Settings: Gimbal mode (retract, ground, gps, angle) and the subsequent values
- **Camera Server**
  - State: None (Ground server can directly change all values pertaining to the camera, therefore they are all settings)
  - Settings: Everything else (see the Camera Server section to learn more)

Controllers

The controller abstractions are meant to interact directly with Java's Play framework. (More information on Play specifications can be found here). They utilize the client and dao methods in order to process client requests and return a meaningful response.

Installation for Development

1. Install Java 8
2. Install git
3. Install VirtualBox
4. Install Vagrant

5. Access ground server through vagrant

```
git clone https://github.com/CUAir/ground-server.git
cd ground-server/
vagrant up
vagrant ssh # Now you're on the VM!
cd ground-server/
```

6. Start the ground server on port 9000

```
./activator run
```

To start tests, run

```
rm -rf conf/evolutions/*
./activator clean
./activator compile
./activator test
```

To access the database on VM, run

```
sudo -i -u postgres
psql -U postgres plaedalus
exit
```

Front-End Overview

The ground server front-end is built primarily in React and it's in `ground-server/app/assets/javascripts`. However, some parts, specifically those that interact with the backend use Nuclear and most of the stylesheets are written in LESS.

Pages

Location: `ground-server/app/assets/javascripts/pages`

These are the individual pages of the frontend that you will see and access. They're made of the components described in the following section.

- **App**: the default page and is located in `/javascripts` rather than in `/javascripts/pages`. If you want to add any components that are applied to all pages, put it there.
  - Components: Drawer, Header
- **Tag**: the first page that you will encounter when starting the server. Meant primarily for tagging targets from images that are fed from the plane. As of now, it also includes starting and stopping the plane's mission status.
  - Components: MissionControl, ImageViewer, ColorSelect, ShapeSelect, TypeSelect
- **Merging**: for merging target sightings with targets and creating new targets. All targets are shown and can be deleted.
  - Components: ColorSelect, ShapeSelect, TypeSelect
- **CameraSettings**: controls the camera's settings and shows what the resulting images look like.
  - Components: ImageViewer
- **GimbalAirdrop**: controls the gimbal and airdrop functions.
  - Components: Airdrop, Gimbal

Components

Location: `ground-server/app/assets/javascripts/components`

The individual UI elements of the system that are built as React classes.

- **ColorSelect**: drop down menu to select the color of the target and also assigns a unique id for the selected color in the following format: `color_select_<integer between 0 and 100,000>_<integer between 0 and 100,000>`
  - Used in: Merge, Tag
- **Drawer**: manages everything in the page below the header. Everything that renders on the page besides the header is wrapped inside of the class "main" which is part of the component. Also sets the sidebar on or off.
  - Used in: all pages (it's in App)
- **Header**: the top bar of the page and includes a button to give access the sidebar.
  - Used in: all pages (it's in App)
- **ImageViewer**: the primary way images from the plane are viewed. Also includes the target selector tool (the big circle that is drawn around a target) for manual detection classification and localization (only active in Tag).
  - Used in: CameraSettings, Tag

- **MissionControl**: displays and sets the plane's mission status through AJAX calls with the API. Note: due to the way the API works, setting the mission status to COMPLETED will prevent any further changes to the mission status. Also, whoever works on this next should use Nuclear instead of AJAX if they can figure out Nuclear.
  - Used in: Tag
- **ShapeSelect**: drop down menu to select the shape of the target and also assigns a unique id for the selected shape in the following format: `shape_select_<integer between 0 and 100,000>_<integer between 0 and 100,000>`
  - Used in: Merge, Tag
- **Sidebar**: main navigation tool within ground server. Opening and closing is controlled by Drawer.
  - Used in: all pages (it's in App)
- **TypeSelect**: drop down menu to select the type (alphanum or emergent) of the target and also assigns a unique id for the selected type in the following format: `type_select_< integer between 0 and 100,000>_<some between 0 and 100,000>`
  - Used in: Merge, Tag

The following two components are in `ground-server/app/assets/javascripts/pages/gimbalAirdrop`:

- **Airdrop**: controls the airdrop's settings and allows you to arm and set the airdrop
  - Used in: GimbalAirdrop
- **Gimbal**: controls the gimbal's settings
  - Used in: GimbalAirdrop

Adding a Component

Once you create a component, go to `ground-server/app/org/cuair/ground/views/main.scala.html`. The `main.scala.html` file is where all the system's CSS and Javascript files are linked to.

In a new line in the file, type the following:

```
<script type='text/javascript' src='@routes.Assets.versioned("javascripts/components/<component's nar
```

This should allow any page in the ground server to access the new component.

Nuclear

Location: `ground-server/app/assets/javascripts/nuclear`

All files built using Nuclear that are meant to allow the frontend to access the databases through API calls using the internal API.

Actions: manages functions related to target sightings and targets. Includes API calls for saving, deleting, and updating targets.

Troubleshooting

Ground Server Cannot Connect to Plane Servers

- Make sure laptop can ping NUC
- Make sure plane servers are running

- Make sure plane you've updated the /ground-server/conf/application.conf file with NUC IP address and plane server port number
- Make sure you've correctly identified plane server port number
- `fping -ag 10.148.0.0/24` (List all IP on the local network)

Ground Server Laptop Cannot Ping NUC

- Make sure laptop is connected to switch
- Make sure switch is connected to antenna tracker router or directly to NUC
- Make sure you've correctly identified NUC IP address
- Make sure the NUC is turned on

Cannot Connect to MDLC UI From My Laptop

- Make sure laptop is connected to switch
- Make sure ground server laptop is connected to switch
- Make sure ground server is running

Geotag Documentation

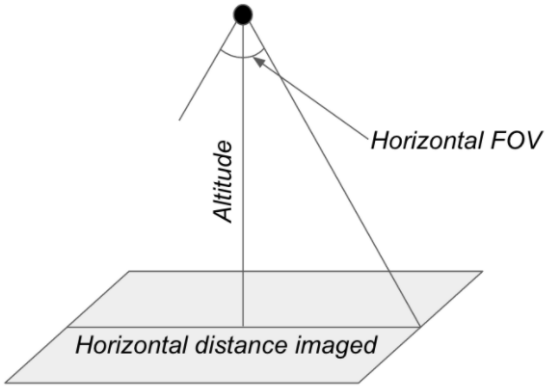
This section provides a mathematical explanation for how Distributed Systems handles image geotagging.

Constants

- FOV_H : the horizontal field of view of the camera lens in radians (source)
- FOV_V : the vertical field of view of the camera lens in radians (source)
- $IMAGE\ WIDTH$: the width of the image captured by the camera in pixels
- $IMAGE\ HEIGHT$: the height of the image captured by the camera in pixels

Math

1. Figure out how much of the ground is captured by each image (in feet). Since the vertical and horizontal field of view is different calculate distance imaged for both axes. Note that the altitude is measured above ground.



$Horizontal\ distance\ imaged = 2 \times altitude \times \tan\left(\frac{FOV_H}{2}\right)$

$Vertical\ distance\ imaged = 2 \times altitude \times \tan\left(\frac{FOV_V}{2}\right)$

2. Figure out how much distance is covered by each pixel in the image. Note that since the camera sensor cells are not square, the horizontal and vertical distance covered is different.

$Horizontal\ distance\ covered\ per\ pixel\ =\ Horizontal\ distance\ imaged / IMAGE\ WIDTH$

$Vertical\ distance\ covered\ per\ pixel\ =\ Vertical\ distance\ imaged / IMAGE\ HEIGHT$

3. The pixelX, pixelY location of the target is from the top left of the image. Since the plane rotation is around the center of the image, we want to find the pixelX and pixelY location of the target using that coordinate frame.

$Relative\ pixel\ X = pixelX - \left(\frac{IMAGE\ WIDTH}{2}\right)$

$Relative\ pixel\ Y = \left(\frac{IMAGE\ HEIGHT}{2}\right) - pixelY$

4. Rotate the pixel location based on the plane's yaw. Basically, the plane is rotated from North by the plane Yaw, and the target's pixel location in the image is in this rotated reference frame. In order to calculate the latitude and longitude of the target, we want to put the target's pixel location in a reference frame where North corresponds with the positive Y axis and East corresponds with the positive X axis.

The above corresponds to a rotation of negative plane yaw in ENU. The image coordinate is in ENU, but the plane yaw we get from the autopilot is in NED. As a result, we rotate by negative of negative of plane yaw.

Rotate $(relative\ pixel\ x\ ,\ relative\ pixel\ ,\ 0)$ where axis is $(0, 0, 1)$ and angle is $planeYaw$ to get $rotated\ pixel\ X$ and $rotated\ pixel\ Y$

5. Calculate the resulting delta latitude and longitude. Based on the location where we're flying, we can get feet per degree of latitude and longitude from the Internet.

$\Delta latitude = (Rotated\ pixel\ X \times Horizontal\ distance\ covered\ per\ pixel) / Feet\ per\ degree\ latitude$

$\Delta longitude = (Rotated\ pixel\ Y \times Vertical\ distance\ covered\ per\ pixel) / Feet\ per\ degree\ longitude$

6. Now we can just add the delta latitude and longitude to the GPS location of the plane when the image was taken to get the target latitude and longitude.

1.3.2 Camera Server

Contents

This section provides the use and specifications of the distributed systems camera server.

Overview

The CUAir camera server is designed to capture images and manage camera settings. Image transferring is handled by the camera server posting the ground server with an image.

Full documentation of the 2016-2017 Camera Server API can be found here.

The camera server is built using the NodeJS web framework. The camera that will be used for the 2017 SUAS Competition is the Z-Cam.

Design Decisions

- Decision to use Z-Cam
  - Were able to capture high quality images at altitude on Atlas
  - Were able to mitigate rolling shutter effects with a simulation on the ground using the Atlas' vibrations
  - Communicated with electrical to ensure that the OBC could send requests to the Z-Cam over its WiFi while simultaneously running a server on electrical's WiFi
  - Simplicity of HTTP API far surpasses Point Grey's poor SDK
  - Reliability of capturing far surpases Point Grey's
- Decision to use Node.js
  - High performance and inherently concurrent (important for continuously accessing images from the camera over HTTP)
  - Conducive to developing HTTP API's
  - Code brevity
  - Easy to use (relatively)
- Decision to use Express

  - Easier interface for developing HTTP API's
  - Code brevity
- Decision to use Sqlite
  - Size of settings and state databases is very small so the simplicity of Sqlite outweighs its lack of efficiency at scale

Field of View

- h : half the height of the lens
- f : focal length

Horizontal No Zoom

$2 \times \arctan\left(\frac{18}{22.75}\right) = 76.7\ (approx.\ 80)$ $2 \times \arctan\left(\frac{h}{f}\right) = 2 \times \arctan\left(\frac{.375_{\text{J}}}{12mm}\right) = 2 \times \arctan\left(\frac{.375}{0.472441}\right) = 76.88\ (approx.\ 80)$

Horizontal Full Zoom

$2 \times \arctan\left(\frac{18}{63.3}\right) = 31.75\ (approx.\ 30)$ $2 \times \arctan\left(\frac{h}{f}\right) = 2 \times \arctan\left(\frac{.375_{\text{J}}}{32mm}\right) = 2 \times \arctan\left(\frac{.375}{1.25984}\right) = 33.15\ (approx.\ 30)$

Vertical No Zoom

$2 \times \arctan\left(\frac{18}{31.5}\right) = 59.49\ (approx.\ 60)$

Vertical Full Zoom

$2 \times \arctan\left(\frac{18}{85.5}\right) = 23.777$

Installation for Development

There are two options for capturing images and changing settings wirelessly from the Z-Cam. One uses a web front end developed by Distributed Systems while the other directly leverages the HTTP API provided by the Z-Cam via the Postman HTTP client.

NOTE: please note down all meaningful settings (shutter speed, aperture, iso, zoom etc.) when capturing images for testing purposes.

Using the Web Front End

1. Install NodeJS
2. Install git

3. Access the Z-Cam server

```
git clone https://github.com/CUAir/z-cam-server.git
cd z-cam-server/
git checkout capturing
npm install
```

4. Turn on the Z-Cam
5. Remove lens (but do not lose it!)
6. Set the camera zoom to full zoom or minimum zoom (but record which one you are using for testing purposes!)
7. Take one picture manually pointing at a very far distances (this focuses the camera)
8. Navigate to the settings and check that WiFi is on
9. Connect to Z-Cam's WiFi network on your computer (password: 12345678)
10. Start the web server

```
npm start
```

11. Use the web front end to capture images and change settings
12. If you use the software at a test flight, please contact Ram to export the image and settings information from your computer so Distributed can save the data for testing purposes
13. Please submit any issues with the front end to our Github repository and/or contact Ram

Direct Requests Using Postman

1. Install NodeJS
2. Install git
3. Install Postman
4. Turn on the Z-Cam
5. Remove lens (but do not lose it!)
6. Set the camera zoom to full zoom or minimum zoom (but record which one you are using for testing purposes!)
7. Take one picture manually pointing at a very far distances (this focuses the camera)
8. Navigate to the settings and check that WiFi is on
9. Connect to Z-Cam's WiFi network on your computer (password: 12345678)
10. To change Z-Cam settings, use Postman to send a `GET` request to `http://10.98.32.1:80/ctrl/set?{setting}={value}`. Possible settings and values are outlined in the "Settings" section of this document.
11. To capture a single image on the Z-Cam, use Postman to send a `GET` request to `http://10.98.32.1:80/ctrl/still?action=single`.
12. If you use the software at a test flight, please contact Ram to export the image and settings information from your computer so Distributed can save the data for testing purposes
13. Please submit any issues with the front end to our Github repository and/or contact Ram

Z-Cam Server API Overview

Properties

- Battery (integer, 0, 100)

Settings

- Brightness (integer, -256, 256)
- Saturation (integer, 0, 256)
- Sharpness (“Weak”,”NormalNoise”,”Strong”)
- Contrast (integer, 0, 256)
- Exposure Value (integer, -96, 96)
- Meter mode (“Center”, “Average”, “Spot”)
- Flicker (“Auto”, “60Hz”, “50Hz”)
- ISO (“Auto”, “100”, “125”, “160”, “200”, “250”, “320”, “400”, “500”, “640”, “800”, “1000”, “1250”, “1600”, “2000”, “2500”, “3200”, “4000”, “5000”, “6400”)
- White Balance (“Auto”, “Manual”)
- Aperture (“5.6”, “6.3”, “7.1”, “8”, “9”, “10”, “11”, “13”, “14”, “16”, “18”, “20”, “22”)
- Auto-Focus Mode (“Normal”, “Selection”)
- Focus (“MF”, “AF”)
- Continuous Auto-Focus (“0”, “1”)
- Burst (“Off”, “On”)
- Drive Mode (“single”, “continuous”, “time_lapse”)

Capturing

- Capture Image

1.3.3 Airdrop Server

Contents

This section provides the use and design of the distributed systems airdrop server.

Overview

The CUAir airdrop server is responsible for delivering a care package as part of the search-and-rescue mission. As such, the airdrop server is constantly running and communicating with both the ground station server and the autopilot server.

Full documentation of the 2016-2017 Airdrop Server API can be found here.

The airdrop server is built using the Flask web framework in Python.

Airdrop System Description

When the airdrop server is started, two jobs are added to a scheduler - **update and drop**. The drop job is scheduled, initially, to execute three hours in the future. The update job is set to execute at an interval, every 0.05 seconds. The update job reads the plane's telemetry data (GPS location/altitude), and if the most recent setting is in the correct mode (discussed below), performs calculations that determine the amount of time until the payload should be dropped. The update job then schedules the drop job to drop the care package at that time in the future.

From the ground station, a user saves a setting through the front end, which is saved in a Settings table. When the user saves a setting, he/she specifies the mode, the location of the target, and the threshold.

The **threshold** is how close the package should be dropped to the target in feet - this value is generally set lower than the threshold specified in the rules.

There are three different modes that the user can set. When the server is in default mode, "not armed", the package is never dropped. This mode is necessary to both save the location of the target and to make sure that we notify the judges before dropping the payload. In this mode, the update job which runs continuously does not query the autopilot server for telemetry data or perform any mathematical computations. The second mode, "armed" is set so that the server will drop the package if it is in the threshold. Now, in the update job, airdrop queries the autopilot server for telemetry information and performs calculations using this information, and the setting information. After calculations, the drop job is rescheduled with the updated time to drop.

The third mode, override, schedules the drop job to execute at the current time. The drop job, when executed, communicates with the airdrop board and to set the angles of the mechanism to "open", which drops the payload.

Also, see the presentation given on Airdrop here.

Connecting to the NUC

Turning the NUC On

What you need:

- Big Black Block Power Supply or Battery that can plug into the plug mentioned below

1. Connect the Big Black Block Power Supply/Battery to the plug on the left side of the blue board (control panel) on the plane
2. Turn the power supply on, make sure it's connected to electric socket.
3. Make sure Rocket AC (white rectangular thing) is blue (powered).
4. Then switch Comms and NUC to down.
5. Then hit the top button of the 2 push buttons on the blue board and make sure it's green.

1. This turns on the NUC!

Connecting to the NUC

(FOLLOW DIRECTIONS IN ORDER)

What you need (If you don't know where any of these boxes are just ask around):

- Another Rocket AC (white rectangular thing) it's a router
  - Antenna Tracker box
- 2 antennas (don't have to be same frequency
  - Antenna Tracker box
- POE connector (has 2 ethernet ports, 1 is named POE) has name "power active"
  - In ground station box
- Ethernet cables
  - In ground station box
- Ethernet to USB converters
  - In ground station box

1. Put antennas on the rocket AC (do this first)
2. Connect rocket AC to POE port (NOT LAN port) of the POE connector via ethernet cable
3. Then plug the LAN port to your computer via ethernet
4. Ethernet to USB converters for MAC in the ground station box.

Make sure you undo everything that you did and put everything back into the right boxes!

SSH into NUC

See the "For Connecting Through Ethernet" section of "Test Flight".

Other Notes

With the NUC and comms currently integrated in the fuselage, it's no longer convenient to hook up a monitor and keyboard to the NUC to enable DHCP so that the NUC can connect to the internet. To make things easier, the em1 interface is now permanently set to a static ip (192.168.0.21) and connected to the AC Wi-Fi comms system so that any computer connected to the ground station rocket AC can SSH into the NUC. A new eth0 interface has been added with DHCP enabled so if an ethernet cable with a usb-ethernet adapter is plugged into a usb port, the NUC will have access to the internet. This allows anyone SSH'd into the NUC to now also access the internet (and pull any relevant code). However, this requires the eth1 interface to be disabled, which is normally used for SRIC. As a result, before connecting to the internet, double check the interfaces file (/etc/network/interfaces) and make sure that the eth0 interface is enabled and eth1 interface is disabled and vice versa for SRIC.

tl;dr: Connect to the rocket under the granite table via ethernet with a 192.168.0.x IP to talk to the NUC (192.168.0.21)

Change the interfaces file on the NUC (/etc/network/interfaces) to enable/disable DHCP (eth0 for internet, eth1 for SRIC)

1.3.4 Gimbal Server

Contents

This section provides the use and design of the distributed systems gimbal server.

Overview

The CUAir gimbal server provides an HTTP interface for changing camera gimbal settings and reading gimbal state/settings.

Full documentation of the 2016-2017 Gimbal Server API can be found here.

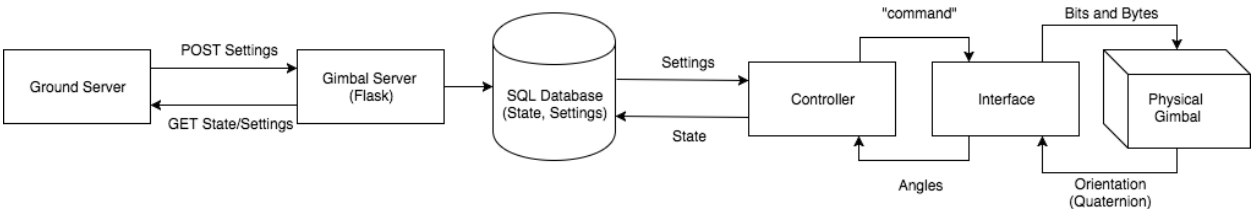
The gimbal server is built using the Flask web framework in Python.

Gimbal Hardware

The gimbal that will be used for the 2017 SUAS Competition is the Basecam Electronics SimpleBGC 32-Bit.

- A third party gimbal board used in conjunction with the server application to control and point the plane camera.
- Interface written in Python (with PySerial)
- User Manual (English)
- Serial Protocol Specification (ver. 2.5x)
- NOTE: Due to what appears to be an error in the firmware, setting a motor axis as inverted in the GUI does not actually invert the axis. As a result, a negation was added any pitch angle sent to the board so that the gimbal follows a North-East-Down frame.

Gimbal System Flow



- The ground server acts as the client and sends GET and POST requests to the Flask Gimbal Server.
- The Gimbal Server processes these requests by either responding with the Gimbal state or updating the settings in the SQL Database. (See "Settings and States" in Ground Server for more information).
- The Controller is layer between database and gimbal interface. It contains an update loop which queries the setting table to determine and send appropriate target angles to the SimpleBGC as well as polling the current IMU angles of the gimbal and adding a corresponding state object to the state table.
- The Interface module provides a serial interface for communication with the SimpleBGC 32-bit board.

1.3.5 Test Flight

Contents

This section outlines the distributed systems test flight procedure.

192.168.0.21 - NUC is static IP

192.168.0.50 - attena tracker

Pre-arrival Checklist

- Make sure you build most recent version of ground-test before getting to the field (no internet), **NEVER EXIT ACTIVATOR SHELL**
- Bring Targets (Shapes and Alphanumerics)

Ground Setup Checklist

1. Setup all the targets in the field and record gps lat/long and orientation coordinates
2. Open compass app and take screen shot
3. Make sure that you restart compass app when you go to new target location
4. Make sure database is empty (run a test in the activator shell)
5. Do "run" in activator shell
6. Turn on nuc, gimbal, and comms (on left side, down is on)
7. Turn on button on top left of dashboard (should turn green)
8. Take off lens cap from camera
9. Set focus of camera on plane to infinity
10. Make sure antenna tracker is on
11. Set a static IP address for yourself
12. Open 4 other terminals
    1. Start Gimbal server:

```
ssh@192.168.0.21
cd odysseus/gimbal-server
./run.py -a 8001
```

    2. Start Camera server:

```
ssh@192.168.0.21
cd odysseus/camera-server
./app.py -dh <your-static-ip> -dp 9000
```

3. Start Plane Autopilot server:

```
ssh@192.168.0.21
cd MAVProxy2/MAVProxy
source venv/bin/activate
python mavproxy.py ---master=/dev/ttyUSB0 57600
```

- Try USB1 and USB2
- To see which one is communicating data, do a cat on any of those USB files
- DON'T TYPE ANYTHING

13. Access Database

```
sql -U postgres plaedalus
```

14. Extra Terminal for pinging NUC (192.168.0.21)

1. If connection to NUC ever goes down, restart all above servers

Ground Pre-Flight Test

1. Open localhost:9000 (MDLC)
2. Test gimbal - point at ground and retract

1. Make sure you enter numbers in for point-at-gps coordinates or the request will fail
2. Ensure that the gimbal rotates and the motor works
3. Check in the database (select * from gimbal_settings) to see when your request goes from queued to sent

3. Test camera
    1. Start taking pictures
    2. Stop taking pictures
    3. Check in database
        1. `Select * from camera_settings;`
        1. Check that requests are getting sent
        2. `Select * from images;`
        1. Check that we are receiving images
        2. Check that they have a gimbal_id and telemetry_id (autopilot)
4. Ask for notification when about to land (to retract gimbal)

Ground Test Flight

1. A couple seconds before take off, start taking pictures
2. Once plane takes off, set gimbal to point at ground
3. When they're about to land, retract gimbal

4. Stop taking pictures

Post Flight

1. Stop ground server ONCE all images are received using `CTRL D`
2. Dump database

```
pg_dump -U postgres plaedalus > 05_11_2016_1.sql
```

3. Move all images to pictures file
4. Create `<date_testflight#>` folder with

1. Folder of all pictures
2. Sql file
3. Logs file

Debugging

1. Trying to ssh into NUC
    1. Possible error: Connection refused
        1. NUC is still powering on
    2. Possible error: No route to host
        1. Switch connecting you to the LAN is down
2. Can do "-h" for running anything and it'll specify parameters

For Connecting Through Ethernet

1. Connect to NUC through Ethernet
2. Change your local network settings to the following:

1. IP Address: 192.168.1.26
2. Subnet Mask: 255.255.255.0
3. Router: 192.168.0.1

3. In Terminal, type `ssh cuair@192.168.0.21`. The password is `aeolus`. You should be able to access and run any programs on the NUC.
4. To get all the pictures captured by the NUC, go into another Terminal window and type: `scp "cuair@192.168.0.21:~/odysseus/camera-server/*.jpg" ~/<path to the folder you want to put the pictures in>`

1. Make sure to keep the quotation marks where they are so that the regex works

Indices and tables

- genindex
- modindex
- search