Software Design Document

for

CubeSat

Version 2.0 approved

Prepared by

Dagraca, Mark
Ha, Kit
Tahod, John
Tobgui, Pierre
Ung, Jonathan

The Aerospace Corporation

9/16/2019
6.2.4 Uses/Interactions.................................................................................................<14>
6.2.5 Resources........................................................................................................<15>

6.3 CubeSat (Navigation Software & Sensors)... ..............................................................<15>
6.3.1 Responsibilities.....................................................................................................<15>
6.3.2 Constraints...........................................................................................................<15>
6.3.3 Composition.........................................................................................................<15>
6.3.4 Interface/Exports....................................................................................................<15>
6.3.4-1 Input..............................................................................................................<15>
6.3.4-2 Storage............................................................................................................<16>
6.3.4-3 Output.............................................................................................................<16>
6.3.4-4 Formulas..........................................................................................................<16>

6.4 CubeSat (Camera)....................................................................................................<17>
6.4.1 Responsibilities.....................................................................................................<17>
6.4.2 Constraints...........................................................................................................<17>
6.4.3 Uses/Interactions..................................................................................................<17>
6.4.4 Composition.........................................................................................................<17>
6.4.5 Limitations...........................................................................................................<18>
6.4.6 Resources............................................................................................................<18>

7. Detailed Lower level Component Design

7.1 dataStream.py..............................................................................................................<18>
7.1.1 Classification.......................................................................................................<18>
7.1.2 Processing Narrative(PSPEC)................................................................................<18>
7.1.3 Interface Description.............................................................................................<18>
7.1.4 Processing Detail...................................................................................................<18>
7.1.4.1 Design Class Hierarchy......................................................................................<18>
7.1.4.2 Restrictions/Limitations.....................................................................................<18>
7.1.4.3 Performance Issues............................................................................................<18>
7.1.4.4 Design Constraints...........................................................................................<18>
7.1.4.5 Processing Detail For Each Operation...............................................................<18>

8. Database Design...........................................................................................................<20>

9. User Interface

1. Overview of User Interface.......................................................................................<21>
2. Screen Frameworks or Images...............................................................................<21>
3. User Interface Flow Model.......................................................................................<22>
# Revision History

<table>
<thead>
<tr>
<th>Team Members</th>
<th>Date</th>
<th>Reason for Change</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dagraca, Ha, Tahod, Tobgui, Ung</td>
<td>9/16/2019</td>
<td>Initial release</td>
<td>v1</td>
</tr>
<tr>
<td>Dagraca, Ha, Tahod, Tobgui, Ung</td>
<td>11/1/2019</td>
<td>Revision</td>
<td>v2</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Purpose

This was created to document how the requirements for CubeSat shall be implemented. It will expand on the functionalities described in the Software Requirements Specifications v2.0, as well as provide details on how each module is implemented, their purposes, and how they all interact with each other.

1.2 Document Conventions

Bold lettering signifies that it is a new section and shall have information given below. Also, bolded words shall have bigger font size to emphasize importance.

1.3 Intended Audience and Reading Suggestions

See SRS document in References.

1.4 System Overview

The SimPlat’s purpose is to carry the CubeSat payload and simulate motion in space in response to commands to move or spin on a horizontal plane. This vehicle will be provided to the computer science team. It uses Omni Wheels for full two-dimension mobility including the ability to rotate in place. When the vehicle receives a signal that the Cub is thrusting to move in a direction, the SimPlat will accelerate in that direction while the force is in effect. When the thrust stops, the vehicle will hold its velocity and continue to move at that speed and direction until countered. To stop the vehicle, an opposite force will have to be applied to reverse the velocity vector precisely. The SimPlat will obey wireless commands to move. It will report its own actions to the GCS.

The SimPlat consists of:
- Omni Wheel vehicle with platform and CubeSat mount
- Wireless two-way communication
- Onboard computer processor
- Software to report status to GCS, process commands from GCS, and react to motion telemetry from Cub

The prototype of this vehicle was developed by the 2018-2019 CSULA Senior Design Team in the computer science area and utilized machine learning techniques to learn how to
move around obstacles. The project targeted anomaly prediction and prevention. The CS team will program the vehicle to move, transmit, and receive appropriately.
2. Design Considerations

2.1 Assumptions and Dependencies

- For Computer vision we will be relying on OpenCV.
- We will assume an artificial velocity and acceleration speed for the SimPlat for demonstration purposes only.
- The connection between GCS and CubeSat will be at the discretion of the team with the approval of Aerospace.

2.2 General Constraints

- CubeSat in our example is limited to a Raspberry PI 4 and it computing and memory processing power.
- 1080p infrared camera feed.
- Must be within the range of a wireless Network.
- Limited onboard battery power.
- Accuracy of physical sensors.
- The SimPlat will be limited to two dimensions.

2.3 Goals and Guidelines

- Use as much open source software as possible to reduce engineering time.
- Employ KISS (“Keep it simple stupid”) principle as much as possible.
- Design with modularity in mind for ease of rapid development.
- The software will aim to employ multiple redundancy to limit the risk of physical damage or injury.
- The entire has to be easily set up for demonstration purposes.
- Goal is to complete the project by the end of Spring 2020 semester.

2.4 Development Methods

We will be using the MD design processing workflow method which is a byproduct of the Agile Development method.
3. Architectural Strategies

The system will be broken down into three distinct types, the team will similarly be divided up. The types are CubeSat, SimPlat, and Ground Control System (GCS). The team employ modern team software engineering practices including use of slack for inner team communication, Google drive for document sharing in a designated team folder and the uses of Git best practices for software development.

4. System Architecture

DFD Level 0
5. Policies and Tactics

5.1 Specific Products Used

Python 3.7
OpenCV version 3.4
Node JS 12.13.0 or greater
HTML 5
Grafana V 6.3.5
Influx DB v1.7.8
Bootstrap 3

5.2 Requirements Traceability

Git best practices will be employed whenever possible.

5.3 Designs and Development Support Tools

A simulator built in Box2d-py is being used to test code in a virtual environment. The simulator will mimic how SimPlat responds to navigation commands meaning that any code that works successfully in the simulator will likely be successful in a physical environment.

5.4 Plans for testing the software

Our team will develop tests for each requirement listed in section 4.1 of the SRS. Then we will use the method of integration testing to test for any faults in the interactions between all integrated components.
6. Detailed System Design
6.1 Ground Control System

6.1.1 Responsibilities

Ground Control System (GCS) will behave as the User Interface that users will be able to access via computer, laptop, or mobile device. Please refer to figure 1 found in section 9.2 of this document as a reference to what GCS will look like. By accessing GCS, users will be able to view data being sent from the CubeSat such as its acceleration, magnetic field, and rotational motion from its 9 Degrees of Freedom sensor. The user will also be able to view the camera feed that is attached to the CubeSat. GCS will also allow users to control the CubeSat by either clicking or tapping on buttons built into GCS. These buttons will allow users to control the CubeSat’s movement, select the target satellite, or set it to automatic movement.

6.1.2 Constraints

A user that wishes to access GCS on their choice of device must be connected to the same network as GCS. This also applies to CubeSat and SimPlat. They must be connected to the same network in order to communicate with GCS.

6.1.3 Composition

GCS consists of our database InfluxDB, user display, and user control. Our database is InfluxDB which is a time series database that is receiving data from CubeSat’s sensors and is storing them for use later. For more details about InfluxDB, please refer to Section 8 in this document.

The user display will show the user data received from CubeSat and CubeSat’s camera feed. Though CubeSat is sending 3-axis data to InfluxDB from its 9 Degrees of Freedom sensor, we will only display the X and Y components of the data since we are only simulating a 2D space with CubeSat.

On the user display, there will be panels made with Grafana that will display data from InfluxDB:

- Acceleration X and Y values
- Angular Velocity Z for rotation
- Magnetic Field X and Y values
- Temperature

For an example of what a panel will look like, please refer to figure 2 in section 9.2
User control will be in the form of buttons that users will be able to click on that will send commands or inputs to CubeSat for it to process and send commands to SimPlat based on the user’s input. The buttons are explained in further detail in the next section.

A description of the use and meaning of the subcomponents that are a part of this component.

6.1.4 Uses/Interactions

GCS interacts with both the CubeSat and SimPlat in different ways. Please refer to figure 1 in section 9.2 for visual reference. When the user clicks the Emergency Stop button, GCS will send an input directly to SimPlat’s movement software, telling it to come to a complete stop by stopping wheel movement. GCS has several interactions with CubeSat as it is in constant communication with CubeSat:

- CubeSat is constantly sending data to InfluxDB which is running on the same router GCS is running on. GCS then displays the appropriate data to the user via Grafana’s dashboard panels. (Section 9.2 Figure 1-2)
- CubeSat’s camera feed is streamed directly to GCS interface, so users are able to see CubeSat’s camera feed in real time. (Section 9.2 Figure 1-1)
- GCS will have three control modes to select from and only one can be selected at a time. Each control mode works differently from one another:
  - Manual: There are buttons that will allow users to click on send thrusters inputs to CubeSat to process so that CubeSat can tell SimPlat which directions to activate its thrusters.
    - The center button found in the middle of manual controls will send a “zero” command to CubeSat, telling it to adjust its thrusters to stop moving. (Section 9.2 Figure 1-6)
  - Semi-Auto: The user will be able to select the target by creating a bounding box with the mouse by clicking and dragging to form a box around the desired target. The bounding box will be sent as an input to CubeSat
  - Autonomous: Set the CubeSat to full autonomous mode.
- The Emergency Stop button will be a button only meant for emergencies or situations that requires the SimPlat to immediately halt all movements in its motors. Clicking on the button will send a command direct to SimPlat with an input to set its motor movements to zero.

6.1.5 Resources

A raspberry pi that is established as a router is needed as this will allow GCS to communicate with other components of the project. The same router is also needed as it will host our database InfluxDB, our dashboard service Grafana, and a HTML server for our Ground Control System interface.
GCS user interface will be using Twitter’s free CSS framework Bootstrap to format the layout of the user interface.

6.1.6 Interface/Exports

GCS provides a user interface that displays data stored in InfluxDB as described in 6.1.3, buttons in the interface to allow the user to send inputs and commands to CubeSat as described in 6.1.4, and an emergency stop button for the user to click on to send a command to SimPlat as described in 6.1.4.

The set of services (classes, resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meaning of the GCS of values, parameters, etc. For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

6.2 SimPlat

6.2.1 Responsibilities

The primary responsibility for SimPlat is to physically move CubeSat around a physical surface. Simulating a 2D zero gravity environment. The SimPlat does this by responding to thruster commands and maintaining velocity until an opposing thruster command is sent.

6.2.2 Constraints

Currently SimPlat in its current form cannot move forward and rotate at the same time.

6.2.3 Composition

SimPlat consist of 4 servos connected to a Raspberry PI GPIO pin. All of this is mounted on a 3D printed base platform.

6.2.4 Uses/Interactions

Truster commands are sent by connecting to the SimPlat through an open source socket based connection called socketIO through which a message is sent. Specifying the thruster and power level. The SimPlat will continue to accelerate the respective “thruster” until another message is received that overwrites the thruster command. These commands are ideally sent by the CubeSat module.

It is also possible to stop all motors in a case of an emergency by the Emergency Stop Software. This is done by sending a GET request to a specified address.
6.2.5 Resources

Servoblaster: https://github.com/richardghirst/PiBits/tree/master/ServoBlaster
APScheduler: https://apscheduler.readthedocs.io/en/latest/

6.3 CubeSat (Navigation Software & Sensors)

6.3.1 Responsibilities

The primary responsibility of the Navigation Software and Sensors is to collect data from the cube satellite’s environment and convert that into commands that move the satellite. This role is similar to the brain of the project as it turns input data from cameras and movement into meaningful commands that result in physical actions by the cube satellite. For the user, this CubeSat is like a black box that simplifies sensory data into commands that the user can physically see.

6.3.2 Constraints

The CubeSat is limited to the sensory data that it receives. It assumes that all its sensors are working correctly and accurately and if any sensor is damaged or malfunctioning, it does not have the capacity to correct it.

6.3.3 Composition

CubeSat can be divided into two main components: Sensory data collection and Navigation processing. The sensory data componen

6.3.4 Interface/Exports

6.3.4-1 Input

Lidar input: floats which represent the distance the CubeSat is from an object on it’s sensor in mm.
9dof input: floats which represent the CubeSat’s acceleration in vector form (x,y,z). This data is in m/s/s.
Camera input: Input is provided in floats which represent distance in mm and an angle in degrees. This is meant to represent a target’s distance away and it’s angle relative to the front of the CubeSat.

6.3.4-2 Storage

Storage of the data is in JSON format on an internal server. More information can be found at http://www.json.org/. The information that is being stored will be all input data. The detailed format for the JSON file is in “Section 8: Database Design”.

6.3.4-3 Output

Commands: Commands are given to SimPlat in the form of a key value pair. There are six different keys that denote which thruster to accelerate. These keys are ‘forward’, ‘back’, ‘left’, ‘right’, ‘RL’, and ‘RR’. The value for each key is the form of a float value which represents the percentage of power to send to a thruster and must move to achieve a certain acceleration.

Example Format 1 (translation):

sio.emit('thruster', {'right':60,'left':60})

In this example, there are two key-value pairs we are passing as commands. These are ‘forward’ and ‘left’ for keys and ‘60’ and ‘60’ are there respective value. This command would tell the left thruster and right thruster to both move at 60% of its max acceleration. This would ultimately result in the SimPlat moving forward at 60% of its max speed forward due to the placement of the left and right thrusters.

Example Format 2 (Rotation):

sio.emit('thruster', {'RL':40})

This command would tell the SimPlat to rotate left at 40% of its max acceleration. There are only two keys that indicate rotation which are ‘RR’ and ‘RL’ which means rotate right and rotate left respectfully.

Note that rotation and translation commands can not be sent at the same time. This is a limitation of hardware as the SimPlat cannot rotate and translate at the same time.

6.3.4-4 Formulas

Let:

- \( P_t \) = Position as a function of time.
- \( P_0 \) = Initial position
- \( V_t \) = Velocity as a function of time.
\[ V_0 = \text{Initial Velocity} \]
\[ T = \text{Time} \]
\[ P_t = P_0 + V_0'T + 0.5(A_0T^2) \]
\[ V_t = V_0 + \frac{\Delta P}{\Delta t} \]
\[ \text{Pos} = P_t + (P_t' - P_t)d_T \]

6.4 CubeSat (Camera)

6.4.1 Responsibilities
The primary role for the software and camera on the CubeSat is to detect and keep track of the either the Aruco Marker or a selected object. The Aruco Marker is a generated square marker that is made up of a wide black border and an inner binary matrix which determines its identification. The software will provide whether the object is in the view of the CubeSat camera as well as its location in respect to the camera frame.

6.4.2 Constraints
The software that interacts with the Camera on the CubeSat is to be calibrated accordingly to the type of camera being used. Different cameras require different calibrations. Once calibration is done accurately the software will be able to give angles with respect to the camera and ArucoMarker. Not accurately calibrating the camera and software will result in errors of distance and angles between the marker and the CubeSat.

6.4.4 Uses/Interactions
User will be able to obtain where ArucoMarker/object with respect to the frame. Information on what direction the detected object is moving will be kept and stored. This will allow the CubeSat to make adjustments accordingly to center objects on the screen. User will be allowed to select target/object to track in semi-autonomous mode. Once a target is selected, the directions of which way object is moving will kept track. Software will provide the change in X and the change in Y with respect to the center of the camera’s frame.

6.4.3 Composition
The CubeSat’s camera will be located on the CubeSat itself. The camera will be mounted to the Raspberry Pi.
6.4.5 Limitations

The CubeSat is limited to processing power of a raspberry pi 4 and has a storage capacity of (storage data here) which can be changed depending on the sd card used. In order to conserve memory space, the CubeSat will limit data retrieval to (⅔) and delete the oldest data as new data comes in.

6.4.5 Resources

OpenCV: https://docs.opencv.org/master/d6/d00/tutorial_py_root.html
Python: https://docs.python.org/3/
7. Detailed Lower level Component Design

7.1 dataStream.py

7.1.1 Classification
A python that file that streams data from CubeSat into InfluxDB while it is running. The kind of component, such as a subsystem, class, package, function, file, etc.

7.1.2 Processing Narrative (PSPEC)
A process specification (PSPEC) can be used to specify the processing details

7.1.3 Interface Description
The script uses python libraries to grab the data the sensors on CubeSat is generating and connect to InfluxDB through the router and store them in the database.

7.1.4 Processing Detail
The script first attempts to connect to InfluxDB. If the script is unable to connect to InfluxDB for whatever reason or the connection is interrupted, it’ll stop running. Otherwise, the script will run as normal and will continue to send data from CubeSat’s sensors and store them in InfluxDB.

7.1.4.1 Design Class Hierarchy
There are no class inheritance for this script. Class inheritance: parent or child classes.

7.1.4.2 Restrictions/Limitations
CubeSat and the machine that is running InfluxDB must be on the same network in other for this script to work properly.

7.1.4.3 Performance Issues
Occasionally the script will not run properly on the first try but will run on the following tries.

7.1.4.4 Design Constraints
The script must know the IP address of the machine and the port that InfluxDB is running, otherwise it will not be able to connect to the database at all.
8. Database Design
Our database is InfluxDB, a time series database. When InfluxDB receives data from CubeSat, it also writes down the time the data was received. This helps indicate at what time InfluxDB received each data. By using InfluxDB along with Grafana as mentioned in section 6.1, we are able to display data from InfluxDB for the user to see using Grafana. Grafana sends GET requests to InfluxDB by generating syntaxes to send to InfluxDB based on what we are trying to display on a specific panel and receives the requested data to display on the panel to the user.

InfluxDB receives data from CubeSat by using a script as described in section 7.1 of this document.

Within the database in our InfluxDB, we will have a table titled “CubeSat” that will hold data sent from CubeSat’s sensors. Table 1 lists all the data that InfluxDB will store from CubeSat.

<table>
<thead>
<tr>
<th>CubeSat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>accel_x</td>
</tr>
<tr>
<td>accel_y</td>
</tr>
<tr>
<td>accel_z</td>
</tr>
<tr>
<td>gyro_x</td>
</tr>
<tr>
<td>gyro_y</td>
</tr>
<tr>
<td>gyro_z</td>
</tr>
<tr>
<td>mag_x</td>
</tr>
<tr>
<td>mag_y</td>
</tr>
<tr>
<td>mag_z</td>
</tr>
<tr>
<td>temperature</td>
</tr>
<tr>
<td>distance</td>
</tr>
<tr>
<td>time</td>
</tr>
</tbody>
</table>

Table 1: A table that displays the name of the table and the data the table holds within InfluxDB
9. User Interface

9.1 Overview of User Interface

The user should be able to select from three different modes: manual, semi-auto, and full-auto mode. The user should be able to control the vehicle in manual mode by clicking the arrow buttons on the page to move the vehicle in the respective direction. In semi-auto mode the user will be able to highlight a desired target. In full-auto mode the CubeSat will recognize target and navigate towards it autonomously.

The feedback the GCS gives the user is a live camera feed from the perspective of the camera on the CubeSat, and information given by all the sensors. The sensory data will be displayed as dashboards and graphs which will be updated in real time. The Ground Control System allows the user the ability to send an immediate stop to the vehicle if that is ever needed.

9.2 Screen Frameworks or Images

![Figure 1: A wireframe mockup of what Ground Control System will look like along with labels to indicate each component of the interface](image)

Key Legend:
1. Camera Feed
2. Data Panels
3. Manual Control Switch
4. Semi-Auto Control Switch
5. Automatic Switch
6. Manual Control Buttons
7. Emergency Stop

Wireframe mockup of what Ground Control should look like on a browser. The panels that can be seen on the left and right side in the mockup will be done in Grafana and put into HTML5 using iFrame. The rest of the key legends are explained in various parts in section 6.1 of this document.

![Temperature Gauge](image)

*Figure 2: An example of what a panel displaying data made by Grafana looks like*

### 9.3 User Interface Flow Model

To be determined.
10. Requirements Validation and Verification

As of the time of this revision (v2), there have been no validation and verification on the specified requirements listed in the SRS document.

11. Glossary

DFD - Data Flow Diagram
SRS - Software Requirements Specification
SDD - Software Design Specification
GCS - Ground Control System
DB - Database

12. References

Software Requirements Specification Document for CubeSat:
https://docs.google.com/document/d/1PlfvhI93fP2tTaxuSNady0hUwHwQ2V82-4koxGMcMdw/edit?usp=sharing