CubeSat Flight Software for Western CubeSat Project

By
Anthony Teixeira
Hyunjoon Kim
Rakul Janatharan
Stephen Amey

Faculty Advisor: Dr. Ken McIsaac

Software Engineering Design Project (SE4450)

Department of Electrical and Computer Engineering
The University of Western Ontario
London, Ontario, Canada

Abstract

In collaboration with Nunavut Arctic College, Western University is participating in the Canadian CubeSat Project: an initiative funded by the Canadian Space Agency to provide professors and students of post-secondary institutions to design, develop, and launch their own CubeSats in a real space mission in Fall of 2021. In the context of a CubeSat, the flight software is the central data handling and control system, responsible for processing and providing communication between the various subsystems and requires a robust yet effective implementation in order for the mission to succeed. The requirements specific to Western University's CubeSat are that of interfacing with an onboard camera, providing a scheduling system for rotations and image captures, telemetry recording and storage, data transmission, and ability to be controlled from a ground station while in orbit. Many of these requirements must also be able to be met while running concurrently, and for that reason the FreeRTOS middleware was used. The main deliverable of this project is the flight software developed using the Nucleo F429ZI development board using STM32MCU, which will then be adapted for the CubeSat's onboard computer.

Acknowledgements

This project was made possible by the many faculty, staff, and students involved. In particular, Dr. Jayshri Sabarinathan (Principal investigator), Dr. Matthew Cross (Project manager), Dr. Matthew Bourassa (Deputy project manager), and Dr. Ken McIssac (Co-investigator). Alexis Pascual's research and work on the data packeting was of great aid during the development of the software. In addition, the other capstone teams who collaborated in the design decisions, providing their experience and feedback, were essential to the project. Finally, the project would not be possible without the collaboration with Nunavut Arctic College and the Canadian Space Agency, for providing the opportunity for Western University to participate in the Canadian CubeSat Project.

Table of Contents

List of Figures

List of Tables

1. Introduction

Our project is to write flight software for a CubeSat being used in the Canadian CubeSat Project (CCP). The CCP is providing students and professors in 15 selected post-secondary institutions with an opportunity to engage in a real space mission. In this section, you will find the purpose for this document, the scope of the product, the intended audience, an overview of the document, definitions, acronyms and abbreviations used, and a description of the conventions and standards used by this document.

1.1 Document Purpose

The purpose of this document is to describe the product being worked on and describe the scope of the product. The product being worked on is the flight software for a CubeSat. This flight software aims to be an integral part of the CubeSat it is implemented in. The description of the flight software being worked on includes its perspective, functionality, users, characteristics, operating environment, constraints, documentation, assumptions, and dependencies. The scope of the product includes all its interfaces and requirements, functional and non-functional. This document's description of the product and the product's scope aims to provide readers with a detailed understanding of the requirements and a description of user interactions that the software must provide to the user for perfect interaction. Also, this document (via a rigorous assessment) will establish the basis for an agreement between customers on how this product should function.

1.2 Product Scope

The software being specified in this document is the flight software for a CubeSat being used in the Canadian CubeSat Project (CCP). This flight software will accept and transmit data between subsystems and the ground station. The software will do this by accepting commands transmitted from the ground station, and then schedule the execution of tasks based on those commands. These tasks include, but are not limited to, taking images, transmitting telemetry, and rotating. The details of these tasks are outlined in the functional requirements section of this document. The purpose of the product is to aid in achieving the CCP's goals. The CCP's goals are to increase students' interest in STEM, particularly in space domains; develop student's expertise in space domains; give students hands-on experience and prepare them to enter the job market; and advance space science and/or technology [1].

1.3 Objectives

The objective of the software is to transmit data from CubeSat to the ground station. The software is responsible for transmission of images taken from the CubeSat Camera, telemetry logging, command execution and many other functions. Further explanation of the specific functions will be found further in the document. The successful transmission of data and commands are critical to the CubeSat system as it is one of the only ways scientists can maintain stability of the CubeSat. With this critical objective of secure transmission in place, the system is developed to be responsible to execute several commands transmitted from the Ground Station and to be able to re-transmit new data back from the CubeSat to the Ground Station in a secure and efficient process.

1.4 Constraints

When developing the system the following constraints were faced:

- CubeSat project is divided into multiple teams, for each component, such as software, electrical, communications, orbital, etc., so difficulty in testing commands that interface with hardware components as some hardware systems are not complete
- Each component of the hardware requires specific protocols to be used, so the software must be able to follow the protocols to communicate with the hardware parts
- Priorities must be set based on the importance of the action on the CubeSat, so if a low priority action is operating and high priority action is requested, the CubeSat must switch to the high priority action at the next opportunity
- The program must be produced through an IDE called STM32Cube IDE
- The program must also be developed using FreeRTOS in order to implement task scheduling and priorities

1.5 Intended Audience and Document Overview

This document is intended to be used by the schools involved in the Canadian CubeSat Project and Western CubeSat project's PIs. These schools are the readers that will use the product this document specifies the requirements for, and the PIs are the reader that will evaluate this document. The rest of this SRS contains an overall description of the product, the specific requirements of the product, other non-functional requirements, and other requirements, in that order. For any school involved in the Canadian CubeSat Project, we recommend reading the definitions, acronyms, and abbreviations, user documentation, user interfaces, product functionality, and safety and security requirements, in that order. For the project PIs, we recommend reading the product scope, product functionality, product perspective, all external interface requirements, design and implementation constraints, assumptions and dependencies, all other non-functional requirements, document conventions, and references and acknowledgments, in that order.

1.6 Definitions, Acronyms and Abbreviations

Refer to the overall document glossary.

1.7 Document Conventions

We followed IEEE formatting requirements. We used Arial font size 11, single spaced with 1” margins, and italics for comments. Major sections are in large, bold font, and subsections are in a smaller (yet larger than the rest of the document), underlined font (i.e. the template standards).

2. SRS

2.1 Overall Description

2.1.1 Product Perspective

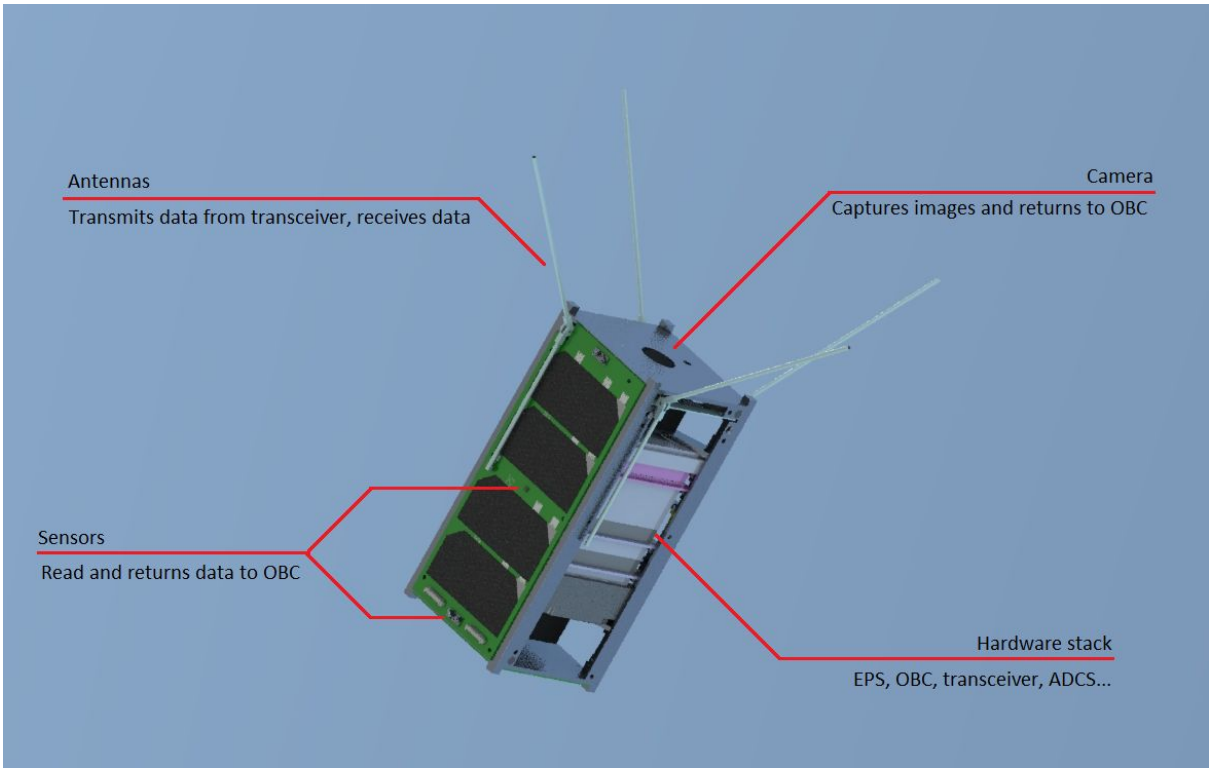

Figure 1: Cubesat structure and solar panels

The CubeSat flight software is one component of a larger collection of subsystems. The software acts as a controller for the various subsystems, interacting with them by message passing through various hardware interfaces. The software reacts to the various subsystems -- performing functions upon receiving communication -- and also initiates functions on its own that will control the other subsystems. As such, the software is the main hub of communication and is a vital component to the overall functioning of the CubeSat and its subsystems.

The flight software is being developed for Western University's CubeSat as part of the Canadian CubeSat Project, demonstrating two novel 180 degree cameras, and is an endeavour that has not been undertaken before by Western University. Previous OBC work was performed in the 2018/2019 capstone projects in [4], which this project partially builds off of.

2.1.2 Product Functionality

- Scheduling system
    - Schedules actions for the cubesat to execute at a later date and manages task execution
- Commands/communication
    - Receives the commands sent from the ground station, and forwards these into the command execution task (e.g. scheduling, transmission, file operations)
- Image capture
    - Captures images using the CubeSat's onboard cameras
- Rotation
    - Provides two states for passive rotation, or scheduled rotation with a specific orientation
- Recording telemetry
    - Records telemetry using the onboard sensors and stores it on the SD card
- Detumble
    - Provides a period to perform detumbling after powering up
- Deploy antennas
    - Deploys the antennas that are used for communication

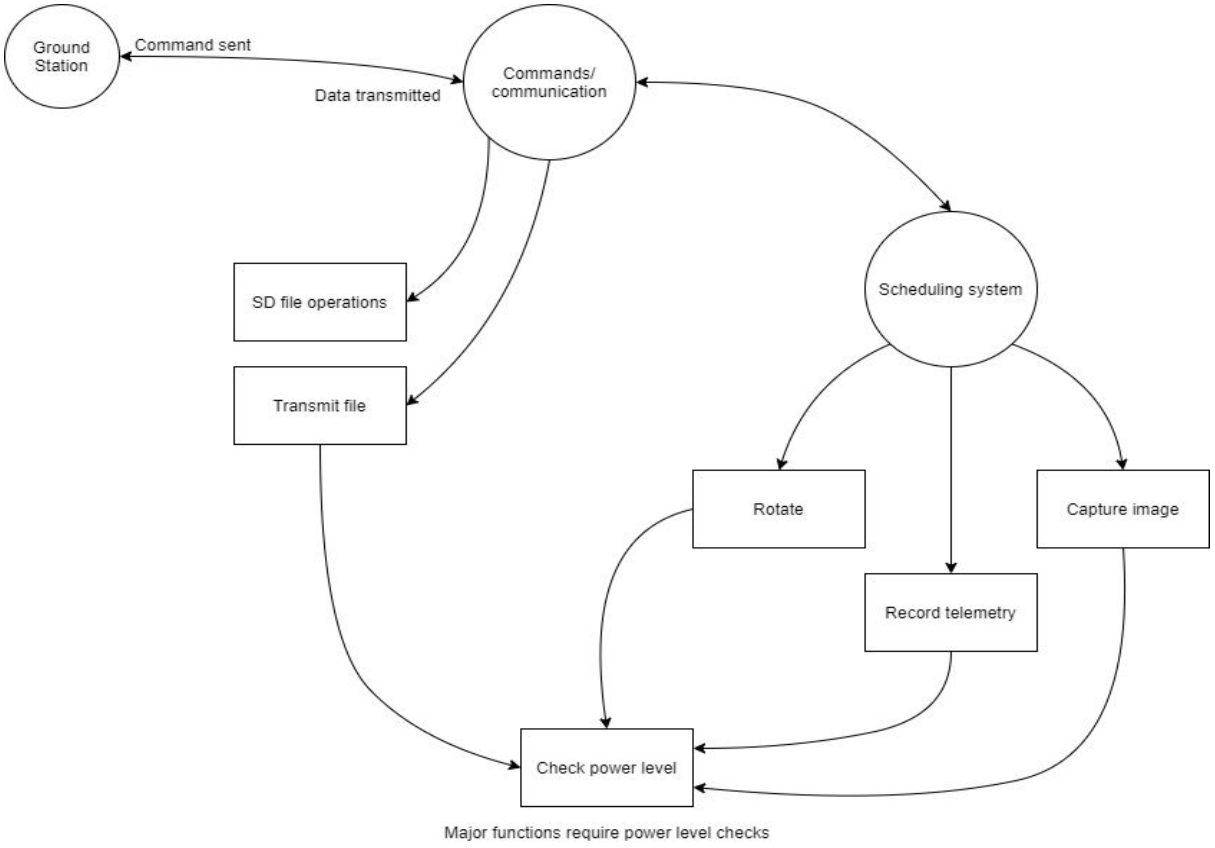

Figure 2: Functionality diagram

2.1.3 Users and Characteristics

There are no direct users of the CubeSat flight software. However, the CubeSat operators will control it via the ground station server, and thus are the sole users by proxy. This single user will have the following characteristics:

- They will operate the software infrequently when issuing commands, but will frequently be monitoring received data and telemetry.
- The user will use all product functions as only the bare minimum, critical functions are to be implemented.
- The user will have vast technical expertise and experience with the CubeSat project. They will be considerably trained in the operation of the CubeSat.
- There is no need for security or privilege levels since there is only one user and a single controlling station.
- As they are involved in the project, the user will have a high educational level and will understand how the software works.
- Since the user is involved in the CubeSat project, they will be very experienced in the mission, requirements, constraints, and operations of the CubeSat.

2.1.4 Operating Environment

The environment in which the flight software will be operating is in-orbit. The software will be running on a NUCLEO F429ZI development board for ground testing, and the EnduroSat OBC when in orbit. The program will also utilize FreeRTOS for its task scheduling and priority abilities. The onboard computer and software must interface with several hardware components: Cameras, EPS, transceiver, ADCS board, and several sensors. It will communicate with a ground station via the transceiver from which it receives commands, and to which it transmits data, telemetry, and images.

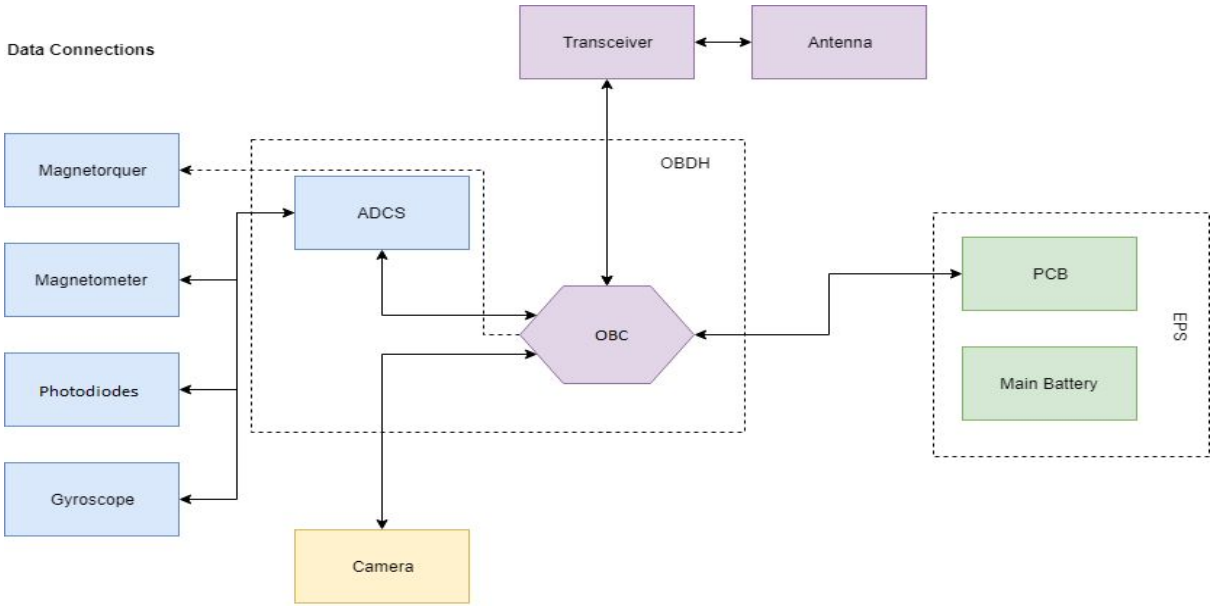

Figure 3: OBC interfaces, [3] Andrew Ning summer 2019 work

2.1.5 Design and Implementation Constraints

- Cannot try the software on the actual CubeSat onboard computer, since it is expensive and if an error in the software causes problems to the board, it may result in damage
- CubeSat project is divided into multiple teams, for each component, such as software, electrical, communications, orbital, etc., so some parts of the CubeSat are currently not decided, or produced to be tested
- Each component of the hardware requires specific protocols to be used, so the software must be able to follow the protocols to communicate with the hardware parts
- Priorities must be set based on the importance of the action on the CubeSat, so if a low priority action is operating and high priority action is requested, the CubeSat must execute the high priority action the next chance it gets
- The program must be produced through the STM32Cube IDE
- The program must be developed using FreeRTOS in order to implement task scheduling and priorities
- CubeSat flight software program must be able to operate on the onboard computer of the CubeSat

2.1.6 User Documentation

Users will not be able to directly access the software since the CubeSat will be in space. However, from the ground station, it will have the ability to communicate with the database through radio signals. Therefore, there are no tutorials or user manuals to help users since the project is only concerned with the flight software that will be operating in outer space. While it is outside the scope of the flight software project, the ground station server will include user manuals and documentation that include the commands needed to operate the CubeSat.

2.1.7 Assumptions and Dependencies

Hardware components of the CubeSat can be changed before the due date of the CubeSat project which is set in Fall of 2021. The CubeSat flight software is dependent on these changes if they occur, since the protocols and methods of operating may require changes which requires alteration of the software.

Another assumption made to the project is that the software constructed will work fully on the actual onboard computer that will be used for the CubeSat. During the construction of the software, testing will be done on a board that imitates the onboard computer of the CubeSat.

2.2. Specific Requirements

2.2.1 External Interface Requirements

2.2.1.1 User Interfaces

The CubeSat operators do not interface directly with the CubeSat software, rather through a ground station server, and are thus outside the scope of the flight software project. Tentatively, the design for the ground station server should likely allow the operator to schedule rotation and image captures, and queue up commands to transmit to the CubeSat on its next pass. It will also likely display CubeSat orbital parameters, estimated next time to communications, and display CubeSat images as it downloads them. However, this is unassociated with the flight software and the ground station team is responsible for it.

2.2.1.2 Hardware Interfaces

2.2.1.2.1 Cameras

Two 180 degree cameras will interface with the OBC via dedicated pins. They will communicate using either the UART protocol (likely) or RS-422, but this is dependent upon which protocol the camera supplier decides to configure the cameras for. The OBC will send commands to the camera to capture an image, return image data, and various other commands yet to be fully determined. The camera itself will return any query data and image data.

2.2.1.2.2 Transceiver

The transceiver will interface with the OBC using dedicated pins (on the Endurosat OBC, these will be in the PC-104 stack). The flight software and transceiver will communicate using the UART protocol at a set clock rate (to be later defined once ground station requirements are determined), and using the AX.25 packet format and behaviour. The data that will be passed are packets to be transmitted (images, telemetry), and packets received (commands).

2.2.1.2.3 Electrical Power System (EPS)

The EPS will be connected to the OBC using dedicated pins (on the Endurosat OBC, these will be in the PC-104 stack). These two components will utilize the UART protocol using the maximum possible reliable baud rate to minimize time spent communicating. The onboard computer (flight software) will query the EPS, and the EPS will return info on its status. The exact format of communication cannot be determined until Endurosat releases the information upon component purchase; a placeholder format will be used with a mock EPS.

2.2.1.2.4 Attitude Determination and Control System board (ADCS board)

The ADCS board will be connected to the OBC via dedicated pins (yet to be determined by the ADCS team). These two components will utilize the UART protocol at the maximum and most reliable baud rate possible. The OBC will send voltage values to the ADCS board which will tell it how to control the onboard magnetorquers. The format of communication is yet to be determined by the ADCS team.

2.2.1.2.5 Various sensors

There will be various sensors onboard the CubeSat, each with its own dedicated pins on the OBC. Sensors include: photodiodes, temperature sensors, magnetometers, gyroscopes, release switches for the antennas. These sensors will utilize the SPI protocol in order to allow more sensors to be placed on board. The format of communication is specific to each sensor as given in its respective datasheet.

2.2.1.2.6 Antenna release

The antenna release mechanism will be connected to dedicated pins on the OBC. The pins will be configured as GPIO. The software will use these pins to run a current through connected burn-wire which is holding down the antennas and keeping them in a stowed position.

2.2.1.3 Software Interfaces

The flight software will interface with a ground station server. The server will issue commands, such as scheduling rotations or image captures, managing the onboard file system, and commanding to transmit telemetry and images. The flight software itself will transmit telemetry, general data, and images. The flight software and the server will not be able to communicate most of the time due to the CubeSat being out of range as it orbits, and as such they must make full use of the interface when available.

2.2.1.4 Communications Interfaces

Communication between the ground station and the CubeSat will utilize the AX.25 protocol. This protocol specifies a packet format, as well as communication behaviour in the case of dropped or corrupt packets. This communication will also utilize a specific baud rate, as transmitting at too high a frequency may result in data loss when the signal is attenuated by the Earth’s atmosphere.

This communication will also use some form of encryption and identifications. This is required to prevent the unlikely case that an unauthorized user with their own ground station may track the CubeSat and issue it commands. Additionally, the CubeSat will also be picking

up many signals from Earth based stations as well as other satellites possibly that utilize the same communication frequency, and thus will need to be able to identify Western Universities ground station among them. However this is outside the scope of the capstone project, thus the implementation is left open.

2.2.2 Functional Requirements

The Functional Requirements as explained in section 2.2, explained the two main cores of the system which were the scheduler and the commands/communication systems. These core systems can be broken down into further subsystems/tasks that depend on the core functionality of the scheduler and commands/communication. This further break down can be observed below:

The scheduling system manages the scheduling and execution of the following FreeRTOS tasks based on a predefined priority level.

Core Functionality Name	Task Name	Use Cases
Scheduling System	RecordTlmTask	Get and record Telemetry
	GetSensorsTask	
	CaptureImgTask	Take Image
	RotateTask	Schedule Rotate
		Passive Rotate
	Detumble (during startup of scheduling system)	Detumble
		Deploy Antenna
	CmdExecuteTask	Commands (next table)

The command/communications system manages the commands issues to the CubeSat, as well as the receiving and transmitting of packets.

Core Functionality Name	Command Name	Use Cases
Command/communications	CAPTURE	Schedule Image Capture
	ROTATE	Schedule Rotate
	UPDATE_TLE	Update TLE values
	LIST_FILES	List file
	DELETE_FILE	Delete file

	GET_FILE	Transmit file
	REFORMAT_SD	Repair SD card

Further details of each function can be found in section 3.3.

2.2.3 Behaviour Requirements

2.2.3.1 Use Case View

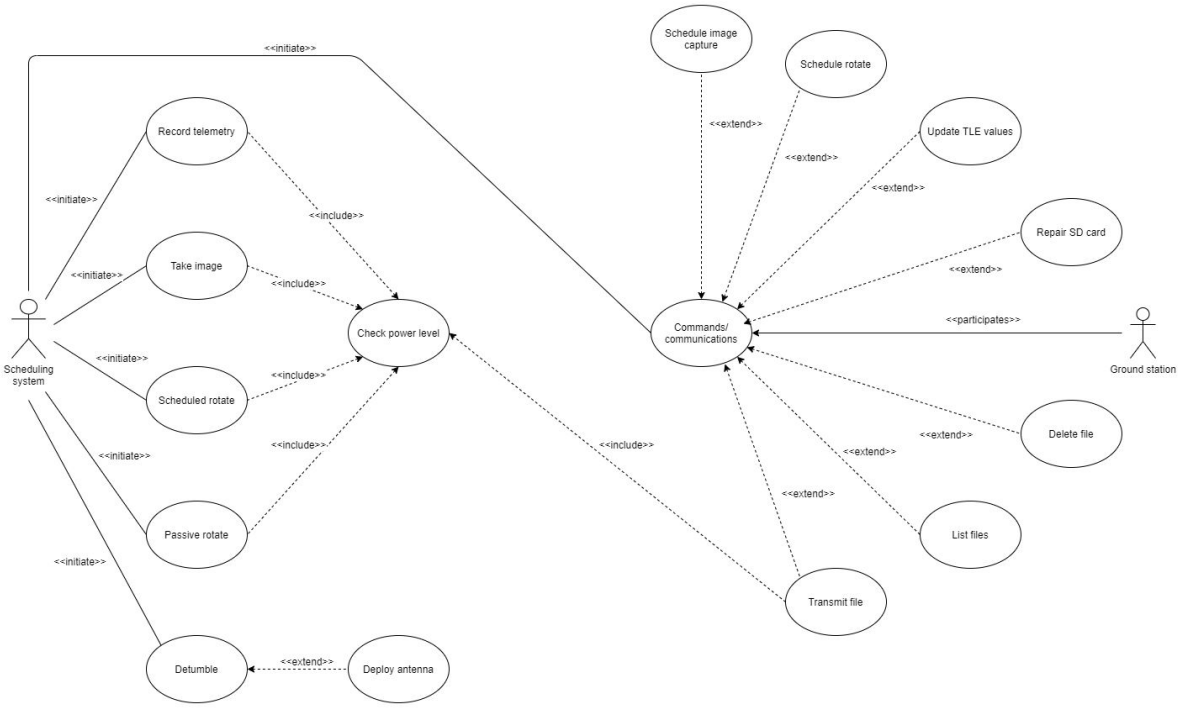

Figure 4: Use-Case diagram

The system will behave in accordance with the use case image that is posted above. The actors that will interact with the system are the scheduling system and the ground station. The ground station will be anyone sending commands to the CubeSat from Earth (in the ground station). The commands will often be to take, transmit and schedule images. The other actor interacting with the system will be the scheduler. The scheduler will be in charge of prioritizing commands received from the ground station in a queue based on a predefined priority index list.

The following is a short description of what each of the use cases will do:

<Record Telemetry>
Queries sensors and subsystems (e.g. power-level from EPS) on set intervals and stores the results in memory for use by telemetry logging and rotation. On separate intervals, parts of this data are selected and written to telemetry files on the SD card.

<Take Image>
If there is an image capture scheduled, performs the image capture process once the timestamp is reached.

<Scheduled rotate>
If there is a rotation scheduled, switches from passive rotation to a specific direction as defined by the scheduled rotation once the timestamp is reached.

<Passive rotate>
If not currently in a scheduled rotation, passively rolls,

<Detumble>
Reads from onboard sensors and attempts to eliminate any rotation (Detumble implementation by ADCS team).

<Deploy antenna>
Runs a current through the burn-wires that are holding down the antennas until the deployment sensor detects that they have been deployed or a specific time interval has elapsed.

<Check Power Level>
Compares the required power level of the function against the CubeSat's current power level (as obtained by Record Telemetry use-case).

<Commands/communications>
Packet data is received from the transceiver, which is forwarded to command execution in the Scheduling system.

<Schedule image capture>
Schedules an image capture at a specific timestamp with a given file name.

<Schedule rotate>
Schedules a rotation at a specific timestamp with a hold-time and direction (Rotation implementation by ADCS team).

<Update TLE values>
Updates the TLE values needed by the CubeSat.

<Repair SD card>
Reformats the SD card to deal with corruption of the file system.

<Delete file>
Deletes a file as specified by its file name.

<List files>
Transmits a list of all files stored on the SD card.

<Transmit file>
Transmits a file as specified by its file name. Specifies a length of transmission time. (Post-capstone implementation will include a bitmask and number of frames per bit for retransmission of missing/corrupt packets).

2.3 Other non-functional requirements

2.3.1 Performance Requirements

The system shall utilize a real-time priority-based operating system, in which each main function as specified in the provided use-cases will have a priority level. This will allow important functions or tasks to run the moment they are needed.

All main functions shall not enter a state where they permanently deny task switching (e.g. deadlock). Some functions may require a longer period in order to complete (e.g. writing an image file to an SD card). This will prevent more important tasks from being run, as well as a task from holding access to a resource for a long period of time.

Functions shall not preempt each other while completing important operations (e.g. writing an image file to the SD card). As some functions must run to completion, and additionally the use of mutexes, task preemption in FreeRTOS must be disabled to ensure deadlocks do not occur.

System functions will record events to an appropriate log file as they execute. As this file will be transmitted during ground-station access, it will allow the operator to ensure that any scheduled commands (e.g. capture image, rotate) actually occurred, and will also allow them to identify any possible errors in execution.

Communication between the flight software and other CubeSat subsystems will utilize the maximum possible reliable clock rate (with the exception of communicating with the transceiver as it does so at a set rate). This will have the effect of minimizing task execution time so that the chance of multiple high-priority tasks needing to be executed at once is lowered.

The system should be capable of queueing up multiple scheduled image captures and rotations. While in orbit, there will be situations where the CubeSat must first rotate to align its cameras with a target as accurately as possible, and then to perform an image capture afterwards. Additionally, the majority of the time it will not be able to contact the ground station, thus it will need the ability to queue up multiple commands at once.

The system should be capable of simultaneously queueing up single image captures and rotations. While in orbit, there will be situations where the CubeSat must first rotate to align its cameras with a target as accurately as possible, and then to perform an image capture afterwards.Due to the long transmission time of images, there is an excess of time in order to schedule another capture and rotation, and thus only a single item of each is needed to be queued.

2.3.2 Safety and Security Requirements

Upon receiving data from the transceiver, the software should be able to check for a type of identifier to verify that the sender is the ground station. The intention of this is to prevent any unauthorized entity from issuing commands to the CubeSat and interfering in its operations. The implementation of user verification is dependent on the ground station and will be left open to fulfill at a later date when the specific implementation is decided.

The CubeSat shall maintain a set of variables containing its orbital TLE values, and will update these when possible during ground station passes. Knowledge of the orbital data is used in global tracking systems to identify possible close passes of satellites and debris. In the unlikely event of a close pass, the other object (if it has the capability) may alter its trajectory to avoid a collision.

Care must be taken when scheduling rotate and image captures in regards to locations on the Earth visible by the cameras. Most satellites are not allowed to capture images of some locations on the Earth's surface (e.g. military facilities), and as such this must be taken into consideration by the CubeSat operator when scheduling the CubeSat to face its cameras towards specific areas and capturing images.

2.3.3 Software Quality Attributes

2.3.3.1 Availability

The software must give a key focus to availability. This is because it must always be available during ground station passes in order to transmit the stored data (which may take upwards of 2 weeks for a single image), and also so that software patches may be applied if needed. Should the system crash, a non-software related hardware component will detect this and restart it. The software itself should avoid implementations that could result in infinite loops or other long-term blocking behaviour.

2.3.3.2 Correctness

Correctness is a high priority item as the CubeSat software only implements the minimal, highly critical functions needed to complete the missions, while in addition the relatively short life-span of the CubeSat limits the amount of software patches that may be performed, thus it should be as correct as possible the first time. As there are only a few, critical functions, those functions can receive large amounts of focus during their implementation. Additionally, this also allows time for them to be extensively tested to ensure minimal faults remain.

2.3.3.3 Interoperability

Interoperability is a key requirement of the CubeSat's mission. Not only must the software effectively communicate with the ground station, it must also communicate with other components within the CubeSat itself (EPS, transceiver, ADCS board). The UART and SPI protocols will be used in communications unless otherwise required. In the case of ground station communication, the software will adhere to a strict packet format utilizing the AX.25 protocol.

2.3.3.4 Maintainability

The software is required to be maintainable in the case that it must be patched while in orbit. It is possible that faults may be discovered while in orbit, and can be patched via communication with the boot loader (a separate program). Due to the limited lifespan of the CubeSat, it is important that the software be maintainable so that it may be patched quickly. This may be achieved by following a strict coding style and following the design specifications accurately.

2.3.3.5 Reliability

It is important for the software to operate reliably due to the short lifespan of the CubeSat. The functions of the CubeSat itself, e.g. transmitting image data, happen over the span of several weeks. As such, it is important for the software to operate reliably this entire time to make sure the images are transmitted successfully, as the short lifespan means it will be limited in how many images it can attempt to transmit. As time allows, extensive software testing will be performed to ensure that it will operate with as little errors as possible. Following a strict and very readable coding style will also prove helpful in making sure that little or no errors are introduced to the critical functions.

2.3.3.6 Robustness

Robustness of the software should be taken into consideration, as the several main functions are critical to the CubeSat's operation. Faults may occur during the operation of the software, and in particular when crossing over the SAA where memory upsets in software are common. During the implementation of the software, extensive error handling may be used to deal with errors. Additionally in the case of memory upsets, redundant variables will be used (e.g. 3 copies), where all copies are compared and the value that the majority of them agree upon is used (e.g. 2 of 3 variables agree on a value, the third is assumed to be the result of a memory upset, and thus it is fixed). This is to be implemented for use by important variables, e.g. TLE data.

2.3.3.7 Testability

As all of the functions of the CubeSat are critical and it is hard to debug and patch the software while it is in orbit, testability is a concern. Making sure that the software is testable will help in delivering an initially error-free product, and if faults do appear during its operation, it will also aid in quickly developing and testing a successful software patch. Designing the software with a strict and readable coding style will help in making the testability of the software sufficient. Additionally, a batch of tests specifically for the critical functions will speed up the testing of the software in the case of a software patch.

2.4 Other requirements

The software should comply with any requirements or changes that the Canadian Space Agency (CSA), National Aeronautics and Space Administration (NASA), NanoRacks, or Canadensys propose at a later date during Western CubeSat Project's scheduled review sessions.

Appendix A – Data Dictionary

In addition to the below table, the software will also contain a very large list of pin constants that are used by connected hardware. This is configurable and depends on the onboard computer itself before it can be determined.

Variables		
Name	**Type**	**Description**
br bw	Integer	Number of bytes read/written in current SD card operation.
executingBuffer	String	The executing string of commands.
commandReadyFlag	Boolean	Signals if the commands are ready to be executed.
command cmdPtr parameter paramListPtr	String pointers	Used in iterating over the executing buffer, the commands, and their parameters.

epoch1 epoch2 epoch3	Integer	The last updated epoch time. Three copies that are checked against each other to ensure correctness.
powerLevel	Integer	Power level reported by the EPS.
solarPanelCurrent	Integer array	Current produced by each solar panel.
solarPanelVoltage	Integer array	Voltage produced by each solar panel.
pdSensors	Integer array	Readings of photodiode sensors.
magnetometerSensors	Integer array	Readings of magnetometer sensors.
tempSensors	Integer array	Readings of temperature sensors.
logBuffer	String	Buffer used in logging events.
lastLogWrite	Integer	Last time the buffer was written.
lastTlmEPSRecord	Integer	Last time the EPS data was recorded.
lastTlmEPSWrite	Integer	Last time the EPS data was written to the SD card.
lastTlmPD_magnReco rd	Integer	Last time the photodiode/magnetorquer data was recorded.
lastTlmPD_magnWrite	Integer	Last time the photodiode/magnetorquer data was written to the SD card.
lastTlmTempRecord	Integer	Last time the Temperature data was recorded.
lastTlmTempWrite	Integer	Last time the Temperature data was written to the SD card.
rotateTimestamp	Integer	Timestamp to start a scheduled rotation.
rotateHoldTime	Integer	Time to hold the rotation for.
rotAxis1 rotAxis2 rotAxis3	Integer	Axis values representing the desired orientation during rotation.
captureTimestamp	Integer	Timestamp to capture a scheduled image.
captureName	String	Name of the scheduled image file.

Table 1: Variables

Constants		
Name	**Type**	**Description**
CLOCK_RATE	Integer	Clock rate in MHz.
SD_MOUNT_WAIT_INTERVAL	Integer	Wait interval to check if SD card is mounted.
ANT_0_DEP_WAIT_TIME ANT_1_DEP_WAIT_TIME ANT_2_DEP_WAIT_TIME ANT_3_DEP_WAIT_TIME	Integer	Times used in determining if the software should continue past antenna deployment.
CMD_BUFFER_SIZE	Integer	Size of the command recieve buffer.
CMD_END_CHAR	Byte	Character that ends a string of commands.
CMD_DELIMITER	Byte	Character that separates commands.
PARAM_DELIMITER	Byte	Character that separates fields in a command.
WRITE_BUFFER_SIZE	Integer	Size of the buffer used when downloading an image from the camera.
POWER_CAPTURE	Integer	Minimum required power to capture an image.
POWER_ROTATE	Integer	Minimum required power to rotate.
POWER_TRANSMIT	Integer	Minimum required power to transmit.
FAT16_MAX_NAME_LENGTH	Integer	Max length of the name of a file on a FAT16 file system.
FAT16_MAX_TYPE_LENGTH	Integer	Max length of the file type of a file on a FAT16 file system.
SD_ACCESS_DELAY	Integer	Forced delay after accessing of SD resources.
SENSOR_GATHER_INTERVAL	Integer	Delay between sensor gatherings in GetSensorsTask.

EPS_MSG_SIZE	Integer	Expected message size to receive from the EPS when querying.
EPS_QUERY_STRING	Integer	Query message to the EPS.
LOG_BUFFER_SIZE	Integer	Size of the log file buffer.
LOG_FILE_NAME	String	Name of the log file.
LOG_WRITE_DELAY	Integer	Time interval between required log file writes.
LOG_FILL_THRESHOLD	Float	Filled threshold between forced log file writes.
TLM_BUFFER_SIZE	Integer	Buffer size for telemetry buffers.
TLM_EPS_FILE_NAME	String	Name of the EPS log file.
TLM_EPS_RECORD_DELAY	Integer	Delay between recording of EPS data to the buffer.
TLM_EPS_WRITE_DELAY	Integer	Delay between writes of the EPS buffer to the EPS log file.
TLM_PD_MAGN_FILE_NAME	String	Name of the photodiode/magnetorquer log file.
TLM_PD_MAGN_RECORD_DELAY	Integer	Delay between recording of PD_MAGN data to the buffer.
TLM_PD_MAGN_WRITE_DELAY	Integer	Delay between writes of the PD_MAGN buffer to the PD_MAGN log file.
TLM_TEMP_FILE_NAME	String	Name of the temperature log file.
TLM_TEMP_RECORD_DELAY	Integer	Delay between recording of EPS data to the buffer.
TLM_TEMP_WRITE_DELAY	Integer	Delay between writes of the TEMP buffer to the TEMP log file.
IMAGE_FORMAT	String	Format of captures images.
ROTATE_CALC_INTERVAL	Integer	Interval between calculations for rotation (ADCS defined)
MAX_HOLD_TIME	Integer	Maximum possible hold time.

AX25_INFO_SIZE	Integer	Size of the payload field in an AX25 packet.
MAX_TRANSMISSION_TIME	Integer	Maximum time a transmission may occur.

Table 2: Constants

Appendix B - Group Log

Meeting Number	Meeting Date	Meeting Duration (hrs)	Meeting Description
1	09/12/19	0.5	Initial meeting to meet and introduce ourselves to each other.
2	09/19/19	1	Completed the team contract and project selection.
3	09/26/19	1	Broke down the project proposal work into individual tasks, so members would be able to work on it individually.
4	10/03/19	1.5	Combined individual work into one proposal document.
5	10/10/19	0.5	Assigned research roles to break down the system.
6	10/17/19	0.5	Brief meeting to discuss what is to be done in terms of the project.
7	10/24/19	1	Research on how software will work and communicate. Still gaining more insight on the project.
8	10/31/19	0.5	Discussing what to include for the walk through presentation.
9	11/05/19	1	Met with member of the CubeSat communications team to outline the communications between the CubeSat and the systems it interacts with as well as details on how we will keep in touch such as using Slack and GitHu

10	11/13/19	3	Setting up software, and how to allocate each component
11	11/15/19	1	Worked on use cases.
12	11/21/19	1	Walk through presentation
13	11/26/19	0.5	Met to outline plan for the midterm report. Evenly distributed individual work.
14	11/29/19	3	Met to follow through our plan for the midterm report.
15	12/05/19	1	Met to reflect on midterm report work.
16	12/12/19	1	Outlined the requirements for the working prototype.
17	12/19/19	1	Met to outline plan for the working prototype. Evenly distributed individual work.
18	12/26/19	1	Continued work on working prototype.
19	01/03/20	1	Continued work on working prototype.
20	01/16/20	1	Continued work on working prototype.
21	01/23/20	1	Finished work on working prototype.
22	02/06/20	0.5	Outlined the requirements for the class demo.
23	02/13/20	1	Met to outline plan for the class demo. Evenly distributed individual work.
24	02/27/20	1	Continued work on class demo..

25	03/05/20	1	Finished class demo functions.
26	03/19/20	0.5	Met via zoom to outline plan for public presentation.
27	03/23/20	1	Finished work for public presentation.
28	03/31/20	0.5	Outlined plan for final report.
29	04/02/20	2	Finished work for final report.

Table 3: Group log

3. SDS

3.1. Introduction of this section

3.1.1 Purpose of this section

The purpose of this section is to provide a written description of our software product, the CubeSat flight software (CFS). This description is written in order to give our development team general guidance to the architecture of the software project. In many ways, this description is needed to coordinate a team as large as ours under a single vision.

3.1.2 Scope of the development project

To write flight software, CFS, used for a CubeSat Project used in Canadian CubeSat Project. CFS will take commands transmitted from the ground station and then schedule execution of tasks based on the commands. The tasks include, taking images, recording telemetry, and rotating. Details will be outlined later on in this document. The objective of the product is to assist achieving the CCP's goals. These goals include providing hands-on education and experience in space domains to students.

3.1.3 Overview of section

The rest of section 3 will consist of sections 2 (logical architecture), 3 (detailed description of components), and 4 (design rational), with their various subsections. Section 2 consists of subsections 1 (overview), and 2 (packages). Section 3 consists of subsections 1 (CubeSat flight software), and 2 (interaction diagram).

In 3.2, we have our logical architecture. This architecture is a top-down view of our system from an architect's perspective. This includes our use-case UML diagram. This diagram is a dynamic or behavior diagram in UML. In 3.2.2, we direct the reader to the detailed documentation for the middlewares we use.

In 3.3.1, we have our flight software's attributes and its states and transitions. Our flight software has many attributes. The attributes of our flight software includes variables pertaining to images taken by the CubeSat, queues to rotate the CubeSat to specific attitudes, references to files that hold interesting data, and references to sensors that can record data in files. The

states and transitions outlined in this document are a little different from most. Our software uses a real-time operating system based on priority task scheduling, so state transitions cannot be defined. Instead, we simply list our states in this section. These states include detumbling in orbit, taking panoramic images, transmitting data back to earth, and more. In 3.3.2, we have our interaction diagrams. This consists of our sequence diagram. This diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

In 3.4, we outline our design rationale. We explain why we opted for a simple approach; this included the reliability that came with it. We also explain the environment that called for this rationale, a team of experts.

3.2 LOGICAL ARCHITECTURE

3.2.1 Overview

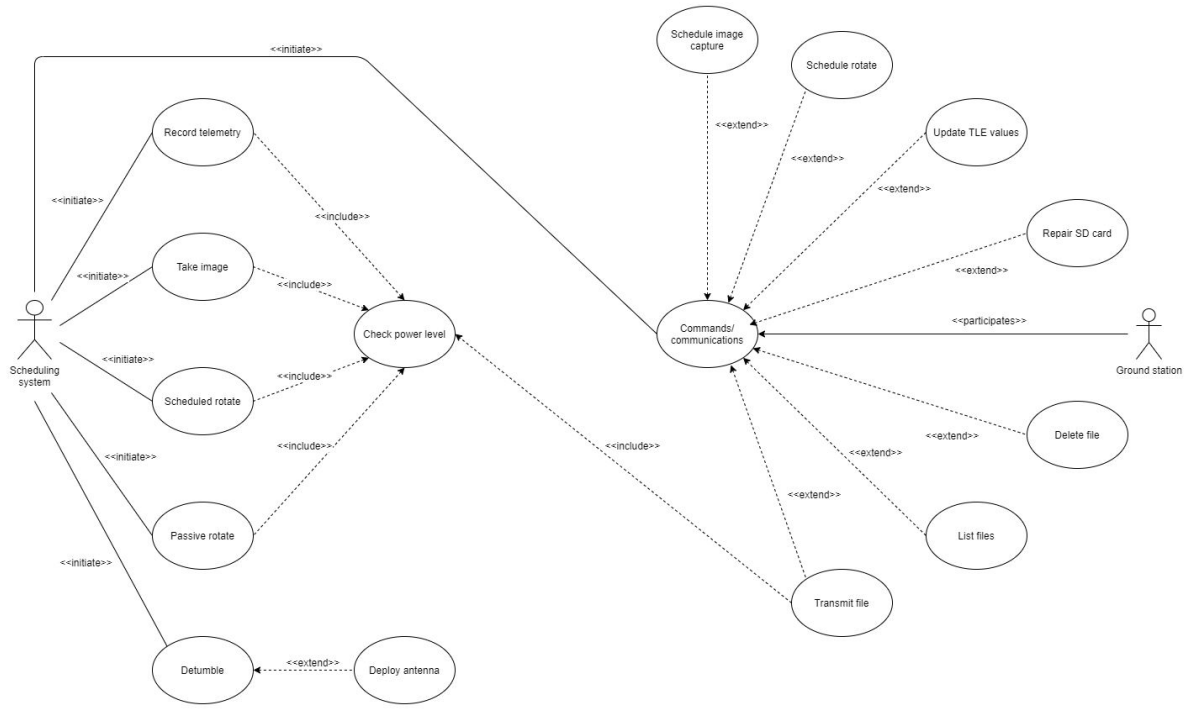

Figure 5: Use-Case diagram

3.2.2 Packages

The CubeSat flight software utilizes two main packages: FreeRTOS and FATFS.

FreeRTOS is the middleware used for priority-based task scheduling, and documentation can be found at [5].

FATFS is the middleware used to communicate with and store data on the onboard SD card. Documentation can be found at [6].

3.3 DETAILED DESCRIPTION OF COMPONENTS

3.3.1 CubeSat Flight software

3.3.1.1 <Main class>

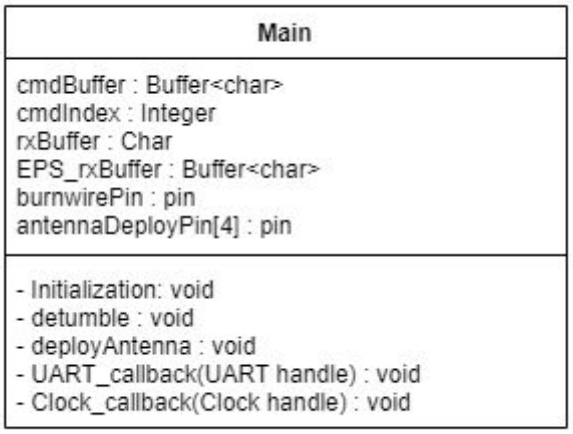

Figure 6: <Main> class

<Main class.Documentation>

3.3.1.1.1 <Main class> Attributes

Name	Type	Description
cmdBuffer	Buffer<char>	Holds the command string as the data is received in the callback.
cmdIndex	Integer	Count of bytes received.
rxBuffer	Char	Holds the next character that is received.
EPS_rxBuffer	Buffer<char>	Holds data received from the EPS.
burnwirePin	pin	Reference to the burn wire pin.

antennaDeployPin	pin[4]	Reference to the antenna switch pins.

3.3.1.1.2 <Main class> Operations

Name	Description
initialization	All startup initialization for pins and FreeRTOS.
detumble	Detumble algorithm upon power up. To be implemented by ADCS.
deployAntennas	Activates the burnwire to deploy the antennas. Continues once all are deployed or a specific amount of time has elapsed.
UART_callback	Interrupt function upon receiving data on one of the UART interfaces. Used for command and EPS UART interfaces.
Clock_callback	Interrupt generated by FreeRTOS timer, used to keep track of the current timestamp.

3.3.1.1.3 <Main class> Design Specification/Constraints

The startup performed in the Main class should not rely on any functionality used by FreeRTOS, and should not take a significant amount of time to complete.

Communication between subsystem components should generally use the UART protocol, and the maximum possible reliable baud rate. Communication with sensors should use the SPI protocol.

3.3.1.1.4 <Main class> States and Transitions

The Main class serves as a ‘startup’ state, after which it transitions to the ‘running’ state where FreeRTOS has been started.

3.3.1.2 <FreeRTOS class>

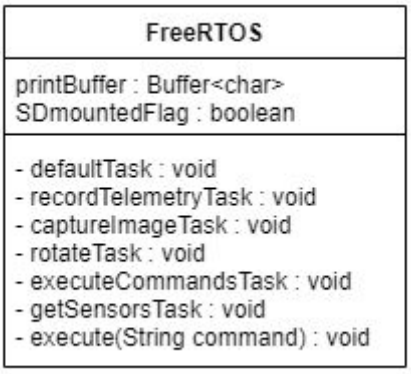

Figure 7: <FreeRTOS> class

<FreeRTOS class.Documentation>

3.3.1.2.1 <FreeRTOS class> Attributes

Name	Type	Description
printBuffer	Buffer<char>	Holds data for manipulation before printing.
SDmountedFlag	boolean	Prevents tasks from executing until the SD card has mounted.

3.3.1.2.2 <FreeRTOS class> Operations

Name	Description
defaultTask	Mounts the SD card and logs the results from startup.
getSensorsTask	Gathers sensor and subsystem data every second and stores it in global variables for use by other tasks.
recordTelemetryTask	Writes the sensor and subsystem data stored in global variables to buffers on set record intervals, and writes the buffers to files on the SD card on writing intervals.
captureImageTask	Performs a scheduled image capture if a timestamp is set and has been reached, and writes the file to the SD card with the given file name.

rotateTask	Performs a scheduled rotation if a timestamp is set and has been reached, and holds it for a specified amount of time. Else performs a passive rotation. The actual rotation will be implemented by ADCS, and will utilize sensor data stored in global variables.
executeCommandsTask	Monitors the command buffer and flag, and once the flag is set it copies the command string to a new buffer and executes the commands via the execute function.
execute	Handles the execution of commands by parsing the command string, and then calling the associated command function.

3.3.1.2.3 <FreeRTOS class> Design Specification/Constraints

As preemption is disabled in the flight software's usage of FreeRTOS, no task should cause permanent blocking, and allow opportunities for a higher priority task to take over.

Each iteration of a task should not take a significant amount of time, with the exception of the captureImageTask and executeCommandsTask.

Tasks that access the SD card should utilize a mutex in order to prevent it from being accessed by multiple tasks at the same time.

Each task should have its own priority level which will be utilized by the scheduling system to determine which task to run at the next switching opportunity.

3.3.1.2.4 <FreeRTOS class> States and Transitions

Each task contained in FreeRTOS can be considered its own state. Transitions can be considered to be the task switching that occurs. Task switching occurs when the currently running task explicitly provides an opportunity to switch. The next task chosen is then based on the priority levels of tasks currently waiting to run.

3.3.1.3 <Commands class>

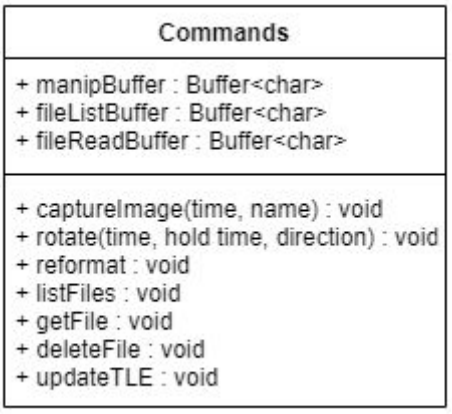

Figure 8: <Commands> class

<Commands class.Documentation>

3.3.1.3.1 <Commands class> Attributes

Name	Type	Description
manipBuffer	Buffer<char>	Holds data for manipulation before printing.
fileListBuffer	Buffer<char>	Holds a string representing the files and their sizes that are stored on the SD card. Used by the listFiles function.
fileReadBuffer	Buffer<char>	Data read from a file in the getFile function is stored in this buffer.

3.3.1.3.2 <Commands class> Operations

Name	Description
captureImage	Sets the capture timestamp and file name based on the provided parameters. The FreeRTOS task captureImageTask will later capture an image using the timestamp and file name.
rotate	Sets the rotate timestamp, hold time, and direction (Axis values) based on the provided parameters. The FreeRTOS task rotateTask will later rotate using the timestamp, hold time, and direction.

reformat	Reformats the SD card.
listFiles	Fills the fileListBuffer with a list of files, and then transmits it to the ground station.
getFile	Continuously fills the fileReadBuffer and transmits it, repeating this until the file is fully transmitted or the transmission time is reached.
deleteFile	Deletes a file from the SD card.
updateTLE	Updates the global TLE variables with the given parameters.

3.3.1.3.3 <Commands class> Design Specification/Constraints

As commands often involve use of the SD card, they should utilize mutexes in order to prevent the same resource being accessed by the current command and another task.

Commands should not execute unless all parameters have been given appropriately.

3.3.1.3.4 <Commands class> States and Transitions

The commandExecuteTask passes control to the command function currently executing, after which it then returns control back to the task.

3.3.1.4 <Utility class>

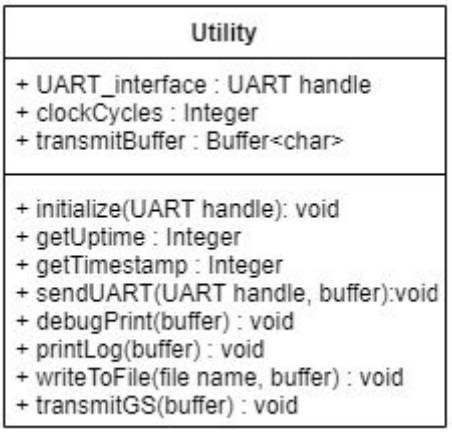

Figure 9: <Utility> class

<Utility class.Documentation>

3.3.1.4.1 <Utility class> Attributes

Name	Type	Description
UART_interface	UART handle	Reference to the UART interface used for transmission.
clockCycles	Integer	Holds the number of total clock cycles. Updated by the timer callback in Main class. Used with the clock rate to determine up time.
transmitBuffer	Buffer<char>	Holds packetized data to transmit.

3.3.1.4.2 <Utility class> Operations

Name	Description
initialize	Sets the UART interface.
getUptime	Returns the amount of time in seconds that the software has been running. Calculates via clockCycles and the CLOCK_RATE constant.
getTimestamp	Returns the current timestamp, using the TLE value epoch and the upTime.
sendUART	Sends the given buffer data to the specified UART interface.
debugPrint	Prints to the debug UART interface to view through serial viewer program, e.g. PuTTy.
printLog	Writes a string to the event log buffer.
writeToFile	Writes a given buffer to the specified file. Wipes the buffer afterwards.
transmitGS	Transmits the given buffer to the ground station via the transmission UART interface.

3.3.1.4.3 <Utility class> Design Specification/Constraints

None, the functions of the class are meant to provide functionality that is used in many places in the software, and remove repeated code.

3.3.1.4.4 <Utility class> States and Transitions

None.

3.3.1.5 <Globals class>

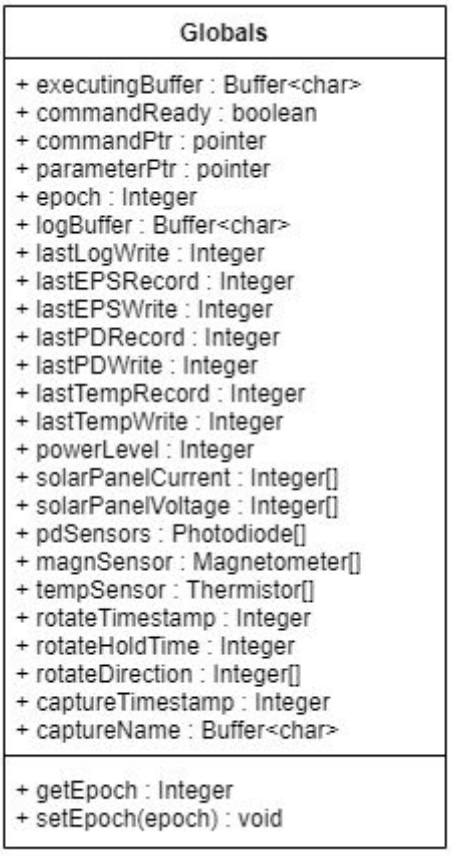

Figure 10: <Globals> class

<Globals class.Documentation>

3.3.1.5.1 <Globals class> Attributes

Name	Type	Description
executingBuffer	Buffer<char>	Holds the command string that is currently being executed.
commandReady	boolean	Signals when the command string received in the callback is ready to be copied to the executingBuffer and executed.

commandPtr	pointer	Points to the current command executingBuffer.
parameterPtr	pointer	Points to the parameters in the current command.
epoch	Integer	Holds the current TLE epoch value.
logBuffer	Buffer<char>	Holds the log events to write to the event log file.
lastLogWrite	Integer	Last timestamp the log was written.
lastEPSRecord	Integer	Last timestamp the EPS data was written to the buffer.
lastEPSWrite	Integer	Last time the EPS telemetry buffer was written to the SD card.
lastPDRecord	Integer	Last timestamp the photodiode data was written to the buffer.
lastPDWrite	Integer	Last time the photodiode telemetry buffer was written to the SD card.
lastTempRecord	Integer	Last timestamp the temperature data was written to the buffer.
lastTempWrite	Integer	Last time the temperature telemetry buffer was written to the SD card.
powerLevel	Integer	Current power level of the CubeSat as obtained from the EPS.
solarPanelCurrent	Integer[]	Currents of the solar panels as obtained from the EPS.
solarPanelVoltage	Integer[]	Voltages of the solar panels as obtained from the EPS.
pdSensors	Photodiode[]	Readings from the photodiodes.
magnSensors	Magnetometer[]	Readings from the magnetometers.
tempSensors	Thermistor[]	Readings from the thermistors.
rotateTimestamp	Integer	Timestamp as set by the rotate function in the Commands class. Specifies the time to perform the next rotation.

rotateHoldTime	Integer	Specifies the time since rotateTimestamp to dedicate to the rotating/holding of the rotation.
rotateDirection	Integer[]	Three axis values that specify the orientation to hold. Specific implementation by ADCS.
captureTimestamp	Integer	Timestamp as set by the captureImage function in the Commands class. Specifies the time to perform the next image capture.
captureName	Buffer<char>	The name of the image file to write to the SD card. Follows FAT16 file naming conventions.

3.3.1.5.2 <Globals class> Operations

Name	Description
getEpoch	Checks the three copies of the epoch variable against each others, performs any needed corrections, and returns the value.
setEpoch	Sets all three copies of the epoch variable to the given value.

3.3.1.5.3 <Globals class> Design Specification/Constraints

All variables defined in this class should be public and accessible by any class that includes it.

3.3.1.5.4 <Globals class> States and Transitions

None.

3.3.2 Interaction Diagrams

3.3.2.1 Sequence diagram

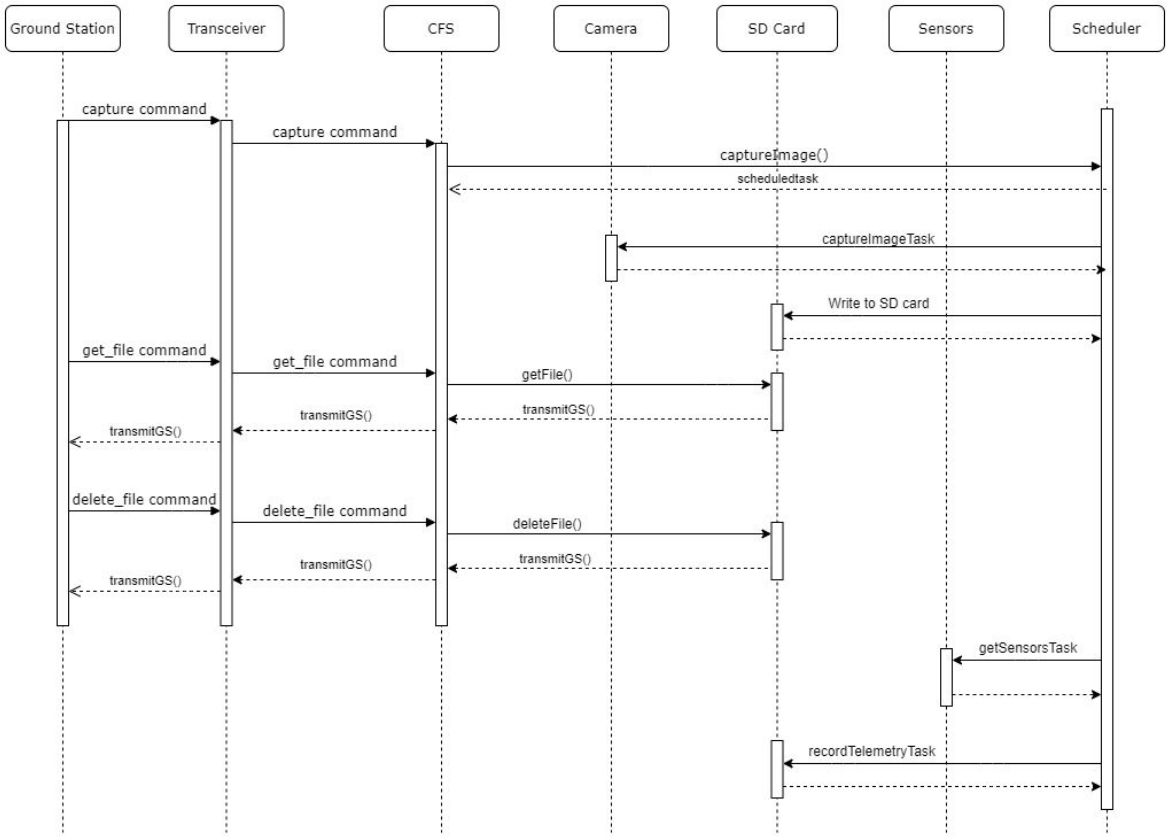

Figure 11: sequence diagram

In this sequence diagram example, the process of capturing an image is shown. First a capture is scheduled from the ground station, and when the time of the capture arrived, the cameras will take the picture. The picture is then saved to the SD card. The ground station then commands to transmit the image, and the CubeSat begins transmission. Once the image has been fully downloaded, the ground station commands that the CubeSat delete its copy of the image from the SD card to free up space. After this, the next getSensorsTask interval is reached and it gathers data from the sensors. Some time after this, the recordTelemetryTask reaches its interval and records this data to the SD card

Most commands that the ground station may issue result in immediate effects (e.g. transmission of data), but may also schedule tasks in the future (e.g. schedule rotate, schedule capture).

3.3.3 Relational Database Schema

3.3.3.1 <DatabaseSchema>Diagram

The CubeSat flight software does not make use of a database.

3.4 Design rationale

The design of the CubeSat flight software opted for a simple approach. While this will result in less ‘automatic’ functionality and is harder for a user to operate, it was thought that this would be safer. The CubeSat is a rare opportunity for a project, and thus this was determined to be the most failsafe way of ensuring it operates correctly. Additionally, operators will be extensively trained and knowledgeable about the CubeSat, thus operation difficulty is not a relevant concern. Additionally, team members have had past experience in the operation of a remote device, and difficulty was encountered when attempting to offer a high degree of autonomy of the device while offering manual control.

It was opted to implement some measures of redundancy on the flight software. Multiple copies of important variables may be used to check if one was changed e.g. during a memory upset by a high energy particle. There are several uncontrollable conditions in which the CubeSat flight software may suffer memory corruption or restarts, thus a measure of redundancy is required.

With regards to scheduling rotations and image captures, it was opted to use a single Timestamp. While multiple captures or rotations may not be lined up in a row, this was determined to not be a required feature, and thus the simpler and less complex option was chosen.

While this may result in issues if scheduled events are inserted in the wrong order (e.g. an event 5 seconds from now is placed after an event 10 seconds from now), the use of a queue simplifies much of the process. This measure of possible error is acceptable as the

Any errors that occur and persist due to incorrect operation may be resolved by a restart (which may occur automatically if the program hangs, via another onboard component called a watchdog timer).

4. Implementation

The following pseudocode is intended to represent a high-level description of what occurs in the various tasks, commands, and functions of the CubeSat flight software. In the following sections the major functionality and code are covered, however minor functionality and code is omitted.

4.1 Main and FreeRTOS tasks

4.1.1 Main method

The main method runs first on power up, and handles the initialization of the peripherals, detumbling, deploying antennas, and then gives control to the FreeRTOS tasks. The main also sets up the UART callback used for receiving commands and data.

main

```
Initialize peripherals;
Enable clock counter;
Initialize util and camera;

Begin listening on UART interfaces;

detumble();
deployAntenna();

Start FreeRTOS tasks;

UART callback{
    If transceiver interface{
        Add to command buffer;
        If end character found{
            Copy the buffer data to the command execution buffer;
            Wipe the buffer;
            Set the commandReadyFlag;
        }
    }
    If EPS interface{
        Write the data received to the global variables;
    }
}
```

4.1.2 defaultTask

The default task is the task first run after the main method completes. It mounts the SD card and sets the flag to signal other tasks that they may run.

defaultTask

```
Mount SD card;
Set mounted flag;

Print startup results to log;

Infinite loop{

}
```

4.1.3 getSensorsTask

The getSensorsTask is responsible for querying all subsystems and sensors every second, and setting the global variables to the received data.

getSensorsTask

```
Wait for SD card to mount;
Infinite loop{
    Send queries to subsystems to tell them to send data (e.g. EPS);

    Read photodiodes;
    Read temperature sensors;
    Read magnetometers;
    //Read any other sensors

    Delay 1 second;
}
```

4.1.4 recordTelemetryTask

The getSensorsTask is responsible for querying all subsystems and sensors every second, and setting the global variables to the received data.

recordTelemetryTask

```
Wait for SD card to mount;
Infinite loop{
    //EPS
    If time elapsed > EPS record delay{
        Set the lastRecordedEPS time;
        Format and add the current EPS data to the data in the EPS buffer;
        If EPS data buffer full, write it to the SD card, and wipe it;
    }
    //Repeat for photodiodes and magnetometers…..
```

```
    //EPS
    If time elapsed > EPS write delay{
        Write the EPS data buffer to the EPS file on the SD card, and wipe it;
    }
    //Repeat for photodiodes, magnetometers, and log/events;

    Delay 1 second;
}
```

4.1.5 captureImageTask

The captureImageTask waits until a capture timestamp is set and the time reached, then captures an image. It then writes the image to the SD card.

captureImageTask

```
Wait for SD card to mount;
Infinite loop{
    If timestamp != 0 and current time > timestamp and sufficient power level{
        Capture the image;
        Take the mutex and disable interrupts;
        Download the image from the camera and write it to the SD card;
        Set timestamp = 0;
        Return the mutex and re-enable interrupts;
    }

    Delay 1 second;
}
```

4.1.6 rotateTask

The rotateTask waits until a rotate timestamp is set and the time reached, then switches to a specific rotation mode and holds it for a given time. Otherwise it is in a passive rotation mode.

rotateTask

```
Wait for SD card to mount;
Infinite loop{

    If timestamp outside scheduled rotation period{
        Wipe the scheduled rotation variables;
        Return to passive rotation mode;
    }
```

```
    If just entering a scheduled rotation period{
        If power level sufficient, enter specific rotation mode;
        Else wipe scheduled rotation variables;
    }

    If power level sufficient{
        If specific rotation mode, calculate voltages for desired orientation;
        Else, calculate voltages for passive rotation;

        Feed voltage data to magnetorquers;
    }
    Else if within scheduled rotation period, wipe variables and return to passive rotation;

    Delay;
}
```

4.1.7 executeCommandsTask

The executeCommandsTask waits until the commandReadyFlag is set. It then iterates over the execution buffer and executes each command with the execute() function.

executeCommandsTask

```
Wait for SD card to mount;
Infinite loop{

    If commandReadyFlag set{
        Unset commandReadyFlag;
        For each command in the buffer, execute(pointer to command);
    }

    Delay 1 second;
}
```

4.1.8 execute method

Execute reads the command string given and if it matches a command, sets the parameter pointer and calls the command function.

execute

```
Read name of command;

If command name matches a command{
    Set parameter pointer to command parameters;
    Call the command function;
}
```

4.2 Commands

4.2.1 captureImage

CaptureImage sets the capture timestamp and filename after verifying the parameters.

captureImage

```
Extract the timestamp parameter and verify;
Extract the file name parameter and verify;

If valid{
    Set the capture timestamp and file name global variables;
}
```

4.2.2 rotate

Rotate sets the rotate timestamp, hold time, and direction variables after verifying the parameters.

rotate

```
Extract the timestamp parameter and verify;
Extract the hold time parameter and verify;
Extract the direction parameters and verify;

If valid{
    Set the rotate timestamp, hold time, and direction global variables;
}
```

4.2.3 updateTLE

UpdateTLE updates the global TLE variables.

updateTLE

```
Extract the epoch parameter and verify;
//Other TLE values as needed

If valid{
    Set the epoch global variable;
    //Set other TLE global variables as needed;

    Reset the number of clock cycles to 0;

   Set lastRecord and lastWrite time variables for EPS, PD, Temp telemetry records to 0;
}
```

4.2.4 listFiles

Transmits a list of all files stored on the SD card to the ground station.

listFiles
Take the SD mutex. Open the file directory. Iterate over each file and add its name and file size to a buffer. Close the file directory; Return the SD mutex; Transmit the list to the ground station;

4.2.5 deleteFile

Deletes a file on the SD card given a file name.

deleteFile

Extract the file name parameter and verify;
 	
If valid and the file exists{
    Delete the file;
    Check if the file was deleted;
}

4.2.6 getFile

Transmits a file stored on the SD card to the ground station, for a specified amount of time.

getFile

Extract the file name parameter and verify;
Extract the transmission time parameter and verify;
 	
If valid and the file exists{
  Take the SD mutex;
 	
  Open the file;
  Read through the file and transmit segments of a set size;
  Close the file;
 	
  Return the SD mutex;
}

4.2.7 reformatSD

Reformats the SD card with a new FAT16 file system.

reformatSD

Extract the file name parameter and verify;
Extract the transmission time parameter and verify;
 	
If valid and the file exists{
  Take the SD mutex;
 	
  Reformat the SD card;
 	
  Return the SD mutex;
 	
  Check for errors;
}

4.3 Utility

4.3.1 writeFile

Writes a buffer to a file given by a file name.

writeFile

Take the SD mutex; Open the file (create if necessary); Write the buffer to the file; Close the file; Return the SD mutex; Check for errors;

4.3.2 transmitGS

Transmits a given buffer to the ground station.

transmitGS
Break the buffer data into smaller segments if necessary. Packetize the data in the buffer; Transmit each byte in the buffer;

4.3.3 packetize

Packetizes buffer data into an AX25 packet.

transmitGS
Adds data for callsigns, frame ID, other leading header fields to the transmission buffer; Adds the data to transmit to the packet; Adds frame status, time, checksum, and end flag to the transmission buffer; Returns the transmission buffer and size of the packet;

5. Test plan

While the FreeRTOS functionality and commands have been tested, the software will require much more testing once the final implementation of ADCS code, sensors, cameras, etc. are chosen and interfaced. Additionally, the porting of the software to the Endurosat OBC will likely require changes to the software, and thus require additional testing. The test cases specified below are recommended to be used when testing changes to the CubeSat hardware and their associated software changes.

The following test cases are a high-level description of what each case covers. The short-term stress tests involve a large number of commands being sent to the OBC over a short time period. The long-term stress tests involve a smaller number of commands being sent to the OBC intermittently over a long time period.

5.1 Startup

5.1.1 Detumble

On startup:

- Sufficient power, not sufficient power, detumble takes a short time, detumble takes a long time, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

5.1.2 Deploy antenna

On startup:

- No antennas deployed, one antenna deployed, two antennas deployed, three antennas deployed, four antennas deployed, and all possible combinations of which antennas are deployed.

Reliability:

- Short term stress test, long term stress test.

5.2 FreeRTOS tasks

5.2.1 Recording telemetry

On FreeRTOS task:

- SD card available, SD card not available, when SD is full, when SD is not full, when power is above threshold, when power is below threshold, when subsystem(s)/sensors(s) are are available, when subsystems(s)/sensors(s) are not available, when log file is deleted, when log file is not deleted, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

5.2.2 Scheduled image capture

On FreeRTOS task:

- SD card available, SD card not available, when SD is full, when SD is not full, when power is above threshold, when power is below threshold, camera connected, camera disconnected, camera working (sends correct data), camera not working (sends incorrect data), SD card write success, SD card write fail, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

5.2.3 Scheduled rotation

On FreeRTOS task (rotation scheduled):

- When power is above threshold, when power is below threshold, very short time specified, very long time specified, moderate time specified, voltage data in range, voltage data below range, voltage data above range, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

5.2.4 Passive rotation

On FreeRTOS task (no rotation scheduled):

- All entry/exit conditions, when power is above threshold, when power is below threshold, voltage data in range, voltage data below range, voltage data above range, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

5.3 Commands

5.3.1 Receiving commands

On command:

- One command, many commands, missing parameters, exact number of parameters, extra parameters, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

5.3.2 List images

On command:

- Not receiving data interrupts, receiving data interrupts, extra parameters, SD card available, SD card not available, few files, many files, files with small

size, files with large size, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

5.3.3 Schedule image capture

On a correct command:

- Not receiving data interrupts, receiving data interrupts, missing parameters, invalid parameters, extra parameters, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

5.3.4 Schedule rotation

On a correct command:

- Not receiving data interrupts, receiving data interrupts, missing parameters, invalid parameters, extra parameters, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

5.3.5 Update TLE values

On a correct command:

- Not receiving data interrupts, receiving data interrupts, missing parameters, invalid parameters, extra parameters, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

5.3.6 Delete file

On a correct command:

- Not receiving data interrupts, receiving data interrupts, missing parameters, invalid parameters, extra parameters, SD card available, SD card not available, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

5.3.7 Get file

On a correct command:

- Not receiving data interrupts, receiving data interrupts, missing parameters, invalid parameters, extra parameters, SD card available, SD card not available, SD card full, SD card not full, and all possible combinations of the above conditions.

Reliability:

- Short term stress test, long term stress test.

6. Conclusions/Recommendations

Overall, the software successfully performs the operations of data transmission, scheduling tasks, communicating, and interfacing with other subsystems as per the initial objectives of the project. The software that is implemented is working in accordance with the initially chosen subsystem interfaces. However, the current software is not in its final form because many of the interfaces have not yet been finalized and have the possibility of changing. Despite this, the software is a complete software platform as per the exact and current implementation required by other subsystems. Therefore, the current software platform can successfully write and read data to the SD card, communicate via interfaces, and transmit data.

It is recommended that all users should review the user manual in order to understand the various tasks/modules of the program. Furthermore, the software is built such that other teams may add onto existing functions. Therefore, new commands, sensors, or algorithms may be added onto the software as needed by the project. Finally, to test the software the team used fake sensor data and a custom camera because the final camera and sensors were not made available. As a result, the final release of the software will instead utilize the actual camera and sensors that will be used in orbit.

7. Appendices

[1] Anon, (2019). *What is a CubeSat*. [online] Available at: http://www.asc-csa.gc.ca/eng/satellites/cubesat/what-is-a-cubesat.asp [Accessed 6 Dec. 2019].

[2] C. Lizotte and S. Amey, "Western University's CubeSat structure and solar panels," Aug. 2019.

[3] A. Ning, "Data interfaces between Western University's CubeSat subsystems," Aug. 2019.

[4] C. Bechard, J. Lee, and S. Terryberry, "ONBOARD DATA HANDLING FOR CUBE SATELLITE," Apr. 2019.

[5] "RTOS Implementation," *FreeRTOS context switch implementation*. [Online]. Available: https://freertos.org/implementation/main.html. [Accessed: 05-Dec-2019].

[6] "FatFs - Generic FAT Filesystem Module," *FatFs - Generic FAT Filesystem Module*. [Online]. Available: http://elm-chan.org/fsw/ff/00index_e.html .

[7] F. George, "Telemetry and Telecommand Transfer Frames Format," *Telemetry and Telecommand Transfer Frames Format*, 19-Mar-2007. [Online]. Available: http://www.goldstem.org/Swiss Cube/05 - Flight software/S3-BC-SE-1-1b-AX.25 Transfer Frames Format.pdf. [Accessed: 03-Apr-2020].

[8] B. Newhall, "The Cyclic Redundancy Check (CRC) for AX.25," *The Cyclic Redundancy Check (CRC) for AX.25*. [Online]. Available: https://www.scribd.com/document/401927729/AX25-CRC. [Accessed: 03-Apr-2020].

[9] "STM32 Nucleo-144 boards data brief," *STM32 Nucleo-144 boards data brief*, 24-Mar-2020. [Online]. Available: https://www.st.com/resource/en/data_brief/nucleo-f429zi.pdf. [Accessed: 03-Apr-2020].

[10] "FreeRTOS API," *FreeRTOS*. [Online]. Available: https://www.freertos.org/a00106.html. [Accessed: 03-Apr-2020].

8. User manual

This section is intended to explain how to interface and control the flight software with an accompanying ground station, as the flight software itself is intended to operate without direct user interaction.

8.1 Interfacing

This section is not concerned with the physical hardware components between the software interfaces of the flight software and the ground station (e.g. transceivers, antennas). Instead, it is only concerned with the format and rate of communications.

Data communication is performed using the AX25 packet format, for both the sending and receiving of data.

Flag	AX.25 Transfer Frame Header (128 bits)				Information Field	Frame-Check Sequence	Flag
	Destination Address	Source Address	Control Bits	Protocol Identifier			
8	56	56	8	8	32-2048	16	8

Figure 12: AX25 format

For packets being transmitted TO the CubeSat, the payload of the packet is intended to be filled only with commands. For packets being transmitted FROM the CubeSat, the payload of the packet is either text data from telemetry files, or data from image files. Packet fields should be filled with the appropriate values (e.g. callsign, to be determined at a later date). The flight software communications operate at a baud rate of 9600, however this is to be properly determined during interface with the groundstation.

8.2 Commands

The following table provides a comprehensive list of all commands the flight software recognizes, providing a description of their parameters and usage.

Command name	Parameters	Usage
CAPTURE	Timestamp File name	Schedules an image capture at the given Timestamp, and saves the file to the SD card with the given File name once captured.
ROTATE	Timestamp Hold time Axis1 Axis2 Axis3	Schedules a rotation at the given Timestamp, and holds it for an amount of time as specified by Hold time. Axis1, Axis2, and Axis3 specify the direction (Proper

		implementation by ADCS).
UPDATE_TLE	Epoch (others as required by ADCS)	Updates TLE values stored on the CubeSat (values required by ADCS will be implemented when determined).
LIST_FILES	None	Transmits a list of all files stored on the SD card and their file size.
DELETE_FILE	File name	Deletes a file with the given File name.
GET_FILE	File name Transmit time	Transmits the file with the given File name for an amount of seconds as specified by Transmit time. (Post-capstone additions will implement missing packet retransmission)
REFORMAT_SD	None	Reformats the SD card by creating a new FAT16 file system, to correct any errors in the current file system.

Table 4: Commands

When transmitting commands to the CubeSat, the command string is placed in the payload section of the AX25 packet. One or more commands may be transmitted in a single overall command string. The command interpretation in the flight software uses three special characters when interpreting the command string:

Character	Usage
,	Separates fields (command name, parameters) within a single command.
|	Separates commands in the overall command string.
^	Signals the end of the command string.

Table 5: Command special characters

A command string consisting of one command is formatted as:

Command_name,parameter1,parameter2….,parameter n^

Ex.
LIST_FILES^

Ex.
CAPTURE,1585743500,fname^

A command string consisting of multiple commands is formatted as:

cmd_1_and_parameters|*cmd_2_and_parameters*|*.....cmd_n_and_parameters*^

Ex.
LIST_FILES|*GET_FILE,fname*|*DELETE_FILE,fname*^

8.3 Good practises

In order for the software to run most efficiently and reduce any unneeded work, there are several recommended practises to follow when interfacing and controlling the flight software:

- Before attempting to download a file, first issue a LIST_FILES command to ensure it exists and to see how large the file is.
- When a file has been fully downloaded from the CubeSat to the ground station, the file should be deleted from the CubeSat's SD card.
- The spelling of all commands and parameters in the command string to be sent should be verified before sending, e.g. via the ground station software.
- Scheduled image captures or rotations should not be scheduled far into the future, and rather as close as possible to the desired time. This is to avoid any issues that may arise during the wait time due to possible memory upsets, e.g. SAA.

8.4 What to avoid?

While the flight software is designed and tested to be fault tolerant, loss of performance and SD issues may arise if improperly used, and as such some behaviour should be avoided:

- Allowing the SD card memory to be filled up with data will result in longer processing times, as well as making it more difficult to transmit the desired parts of a file.
- Avoid violating the AX25 packet format in order to e.g. gain more 'command space'.
- Unnecessary repetition of commands.
- Reformatting of the SD card via REFORMAT_SD command unless absolutely necessary.

9. Glossary

ADCS	Attitude Determination and Control System
AX25	A communication protocol that defines packet format
EPS	Electrical Power System
CCP	Canadian CubeSat Project
CFS	CubeSat Flight Software
OBC	Onboard computer
FatFS	Middleware used for SD card interfacing
FreeRTOS	Middleware used for task priority and scheduling
NASA	National Aeronautics and Space Administration
PC-104	A connection scheme commonly used in CubeSats
PI	Project investigator
SAA	South Atlantic Anomaly, an area of the Earth's magnetic field where it dips close to the Earth's surface. Spacecraft passing through this area commonly suffer memory upsets.
SPI	A communication protocol used with sensors. It allows the same pin connections to be used for multiple sensors
TLE	Two-line element set, used to encode orbital parameters and define the position of the CubeSat in orbit
UART	Universal Asynchronous Receiver/Transmitter, a protocol used for serial communication between components
UML	Unified Modeling Language

10. Vitae

Stephen Amey, 4th year Software Engineering, Astrophysics
samey3@uwo.ca

Anthony Texeira, 4th year Software Engineering
ateixei4@uwo.ca

Hyunjoon Kim, 4th year Software Engineering
hkim778@uwo.ca

Rakul Janatharan, 4th year Software Engineering
rjanatha@uwo.ca