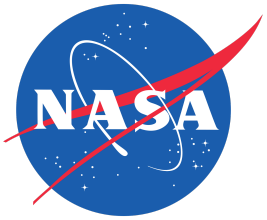
CougSat Simulation Software

Alpha Prototype Report

Mentor

Aaron Crandall

Team Viking

Courtney Snyder
Angie Park
Solomon Egwuonwu

Course: CptS 421 Software Design Project I
Instructor: Aaron Crandall

TABLE OF CONTENTS

I. Introduction

The purpose of this document is to outline the progress we have made this semester, the resulting alpha prototype, and the plans for our beta prototype next semester.

Our client, the club Cougs in Space of Washington State University in Pullman, wants to know that their cubesat, CougSat I, will successfully launch and orbit. To help them feel more confident and help them further develop their design, we will build a testing suite for them. This testing suite will go through all stages of both the launch and orbit to determine if anything could fail and identify the cause of failure. This will enable the club to create a more robust system and more likely make for a successful mission.

Development of the testing suite will be broken up into two phases, the alpha prototype and the beta prototype. This semester, we have been working on the alpha prototype, which is a state machine that looks at all of the top-level behaviors that CougSat I will have, most of which will be represented by various power modes.

The rest of the document contains a description of our alpha prototype, an overview of our alpha prototype demonstration, and future work for our beta prototype.

II. Alpha Prototype Description

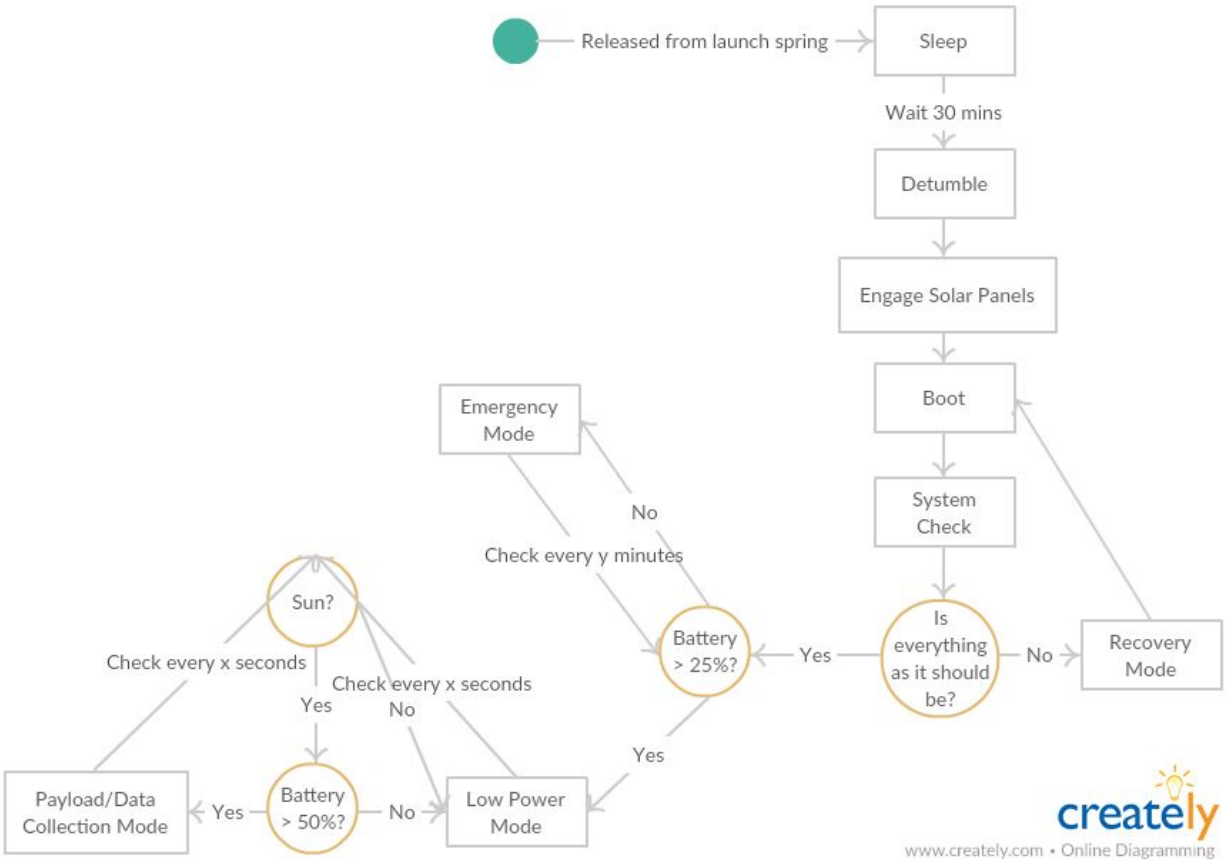


Figure 1. General state machine for CougSat I

For our design, we implemented a state pattern. Since we are more concerned with CougSat I's behavior and our alpha prototype is a state machine, this pattern made the most sense.

As seen in Figure 1, we have determined three power modes: Emergency, Low Power, and Payload/Data Collection. Please note that Recovery mode is not a power mode because it is not dependent on how much battery power is remaining; rather it is dependent on whether or not the satellite successfully passes the system check.

Each power mode is determined by how much battery power is remaining; Emergency is for less than 20% battery, Low Power is for between 50% and 20% battery, and Payload/Data Collection mode is for greater than 50% battery. Within each of those power modes, different subsystems will be turned on, and will exhibit different behaviors.

When CougSat I is in Low Power mode, the Attitude, Main, Communications, and Power circuit boards will be on. This is the normal operating behavior of the satellite, as it allows for communication with the ground station, on-board power control, and satellite stabilization. However, if the battery level is too low, having all of these subsystems on will quickly drain the battery.

Thus, when CougSat I is in Emergency mode, only the Main and Power circuit boards will be on. None of the other subsystems will be able to use valuable power at this point because our client would rather have the satellite efficiently budget and collect power than die.

Tentatively, when CougSat I is in Payload mode, the Attitude, Communications, Payload, and Power circuit boards will be on. When the satellite is in Payload mode, it will behave in the same way as it does in Low Power mode, but the payload circuit board will also be on and collecting data to send to the ground station in some capacity.

We further broke our project into five main subsystems because there are currently five circuit boards planned for CougSat I and we wanted to illustrate both how they work individually and how they work together. The general state machine subsystem is the cubesat's behavior as a whole and includes all of the circuit boards. The Attitude, Communication, and Power subsystems illustrate individual circuit board behaviors. Originally, the Communications subsystem contained both inner-satellite communications and ground communications. The inner satellite communications are all through the main circuit board; that is, the main board is connected to each of the other circuit boards and the other boards are not connected to each other. We decided to implement the Main board as its own subsystem, rather than combining it with Communications to lower coupling and increase cohesion.

We used ROS Kinetic to implement each of the circuit boards as a subsystem state machine. We were unable to make a subsystem state machine for the Payload board since it does not yet have a team and is not yet designed. We were able to integrate the individual circuit boards and the general state machine design by implementing the states from the general state machine as a function in the appropriate subsystem state machine.

Currently, the user interface is launching each circuit board ROS node in separate terminals. Specifically, one terminal for the required ROS core process, one terminal per subsystem state machine, and one terminal for the initial sleep mode. As can be seen in Figure 2, there is a total of six terminals.

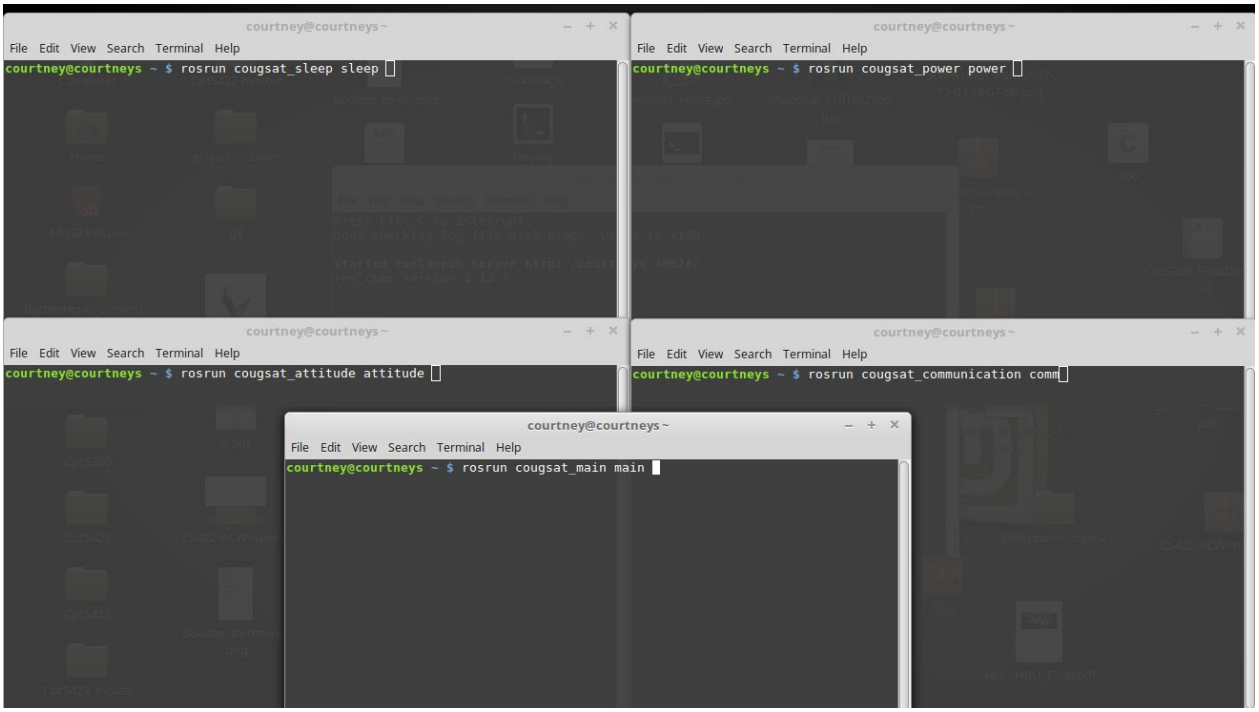


Figure 2. User Interface before running the nodes

The program is launched by running one command in each terminal:

rosrun cougsat_subsystem subsystem

Since the sleep node is the first node to run, we added the “Press enter to run” so the sleep node could subscribe to the attitude topic and vice versa before running.

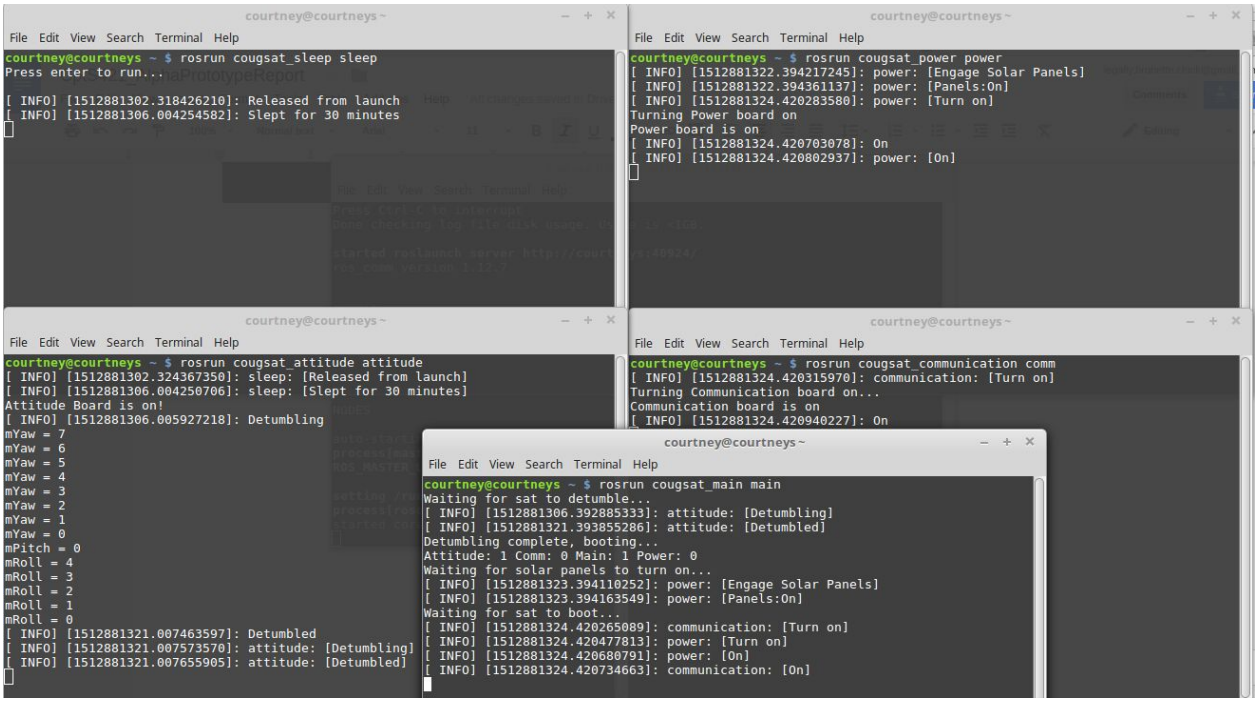


Figure 3. User Interface after running the nodes

We are able to see that the nodes are communicating appropriately through the topics since the actions were happening in the order that they were supposed to happen. We also used a tool called ROS RQT, which dynamically creates a graph of running ROS nodes. This tool shows what nodes are running, and what topics each node is publishing to and what topics each node subscribes to.

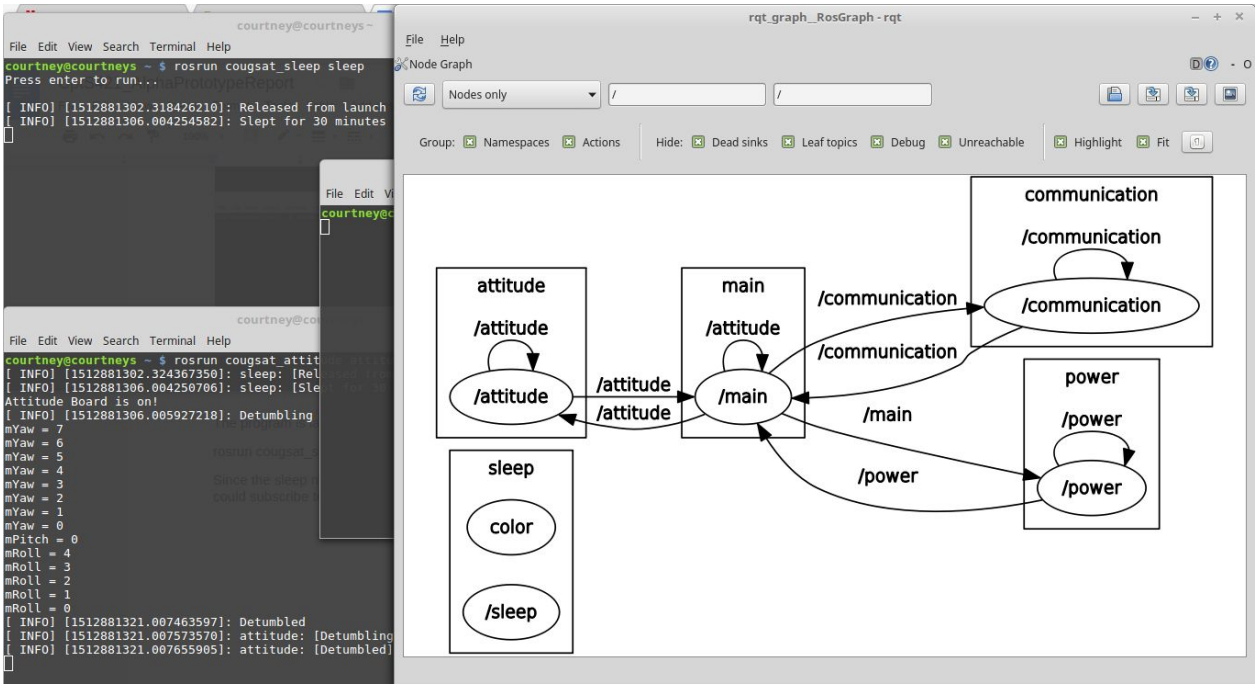


Figure 4. RQT Graph of the simulator

II.1. Attitude

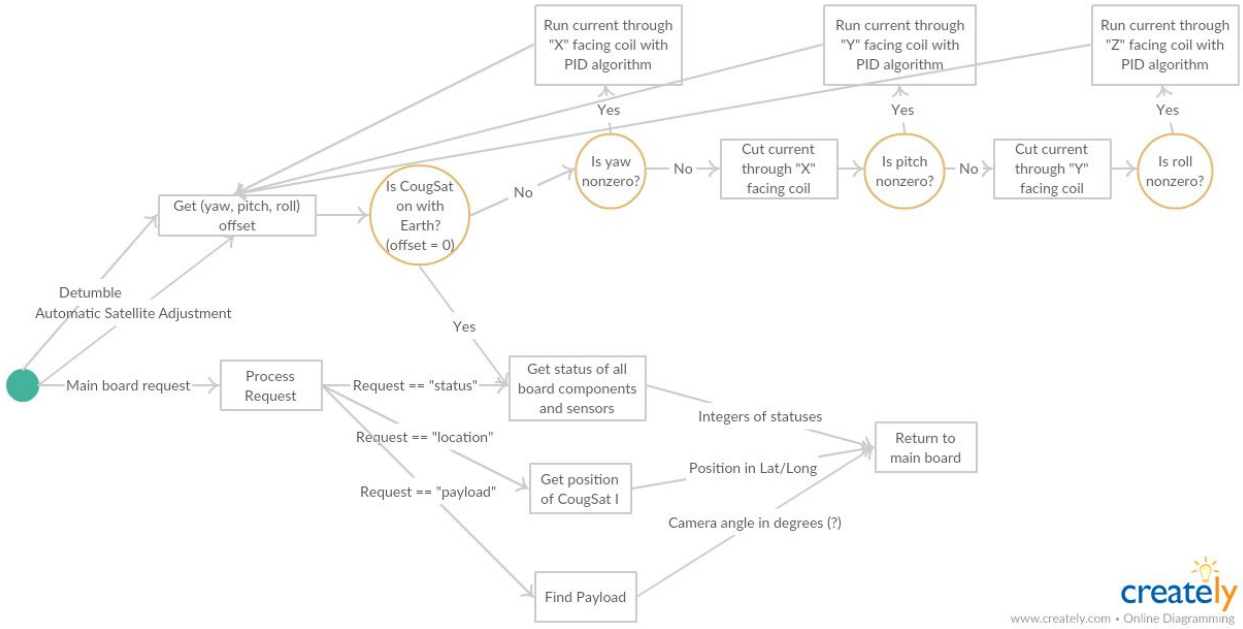


Figure 5. State machine for the Attitude subsystem

II.1.1. Functions and Interfaces Implemented

II.1.1.1. Detumble

The attitude board will be connected to an isolated timer that begins counting as soon as the satellite is launched; once the timer reaches 30 minutes, the attitude board will turn on and begin to detumble the satellite. Typically, the attitude board will measure yaw, pitch, and roll offsets using the gyro; right after the launch, the cubesat will be moving so fast that the offsets will all be zero. Instead, we will measure the angular velocity in each direction. The attitude subsystem will use a proportional-integral-derivative (PID) algorithm to efficiently bring the angular velocities to zero, then for each direction we will turn on the coil in the perpendicular face to align it to the initial coordinate system that points at Earth. Once the satellite is no longer tumbling and is pointing at Earth appropriately, it is considered detumbled. This function is almost fully implemented, it just needs to be testing with more accurate angular velocities acting on it.

II.1.1.2. Automatic Satellite Adjustment

After the satellite is detumbled, the attitude board will use the gyro to get the yaw, pitch, and roll offsets, and will turn on the coil in the perpendicular face of the satellite to correct the offset as needed. This function is not yet implemented, as the satellite does not change its position at all and has no physical forces acting on it. After the physics library is integrated, we will be able to get the offset of the coordinate system and adjust the satellite appropriately.

II.1.1.3. Main Board Request

The main board can turn the attitude board on and off, and the ground station can ask the satellite different things about itself using a set of commands. The main board will receive the commands, decrypt them, and request information from the appropriate. The main board can get status and location from the attitude board. If the command is status, the attitude board will get the status of all of its components and sensors and tell the main board if everything is working as it should be. If the command is location, the attitude board will tell the main board its location in latitude and longitude. This function is not yet implemented.

II.1.2. Preliminary Tests

I first made sure that the attitude node was subscribed to the “sleep” topic so that it could begin detumbling after the satellite slept. I confirmed this by running only roscore, the sleep, and the attitude nodes and printing out the published messages from the sleep node in the sleep callback function of the attitude node.
I tested the Detumble function by checking that each angular velocity was decreasing when I called its fix function. I first checked for yaw; fixYaw would decrease yaw by 1 degree per second and stopped decreasing when yaw == 0. Then I checked if fixPitch decreased pitch by 1 degree per second and stopped decreasing when pitch == 0. Finally, I checked if fixRoll decreased roll by 1 degree per second and stopped decreasing when roll == 0, and publish that detumbling is complete on the “attitude” topic. I then put each function call into the detumble function and tested it using random values under 20 for yaw, pitch, and roll.

II.2 Communication

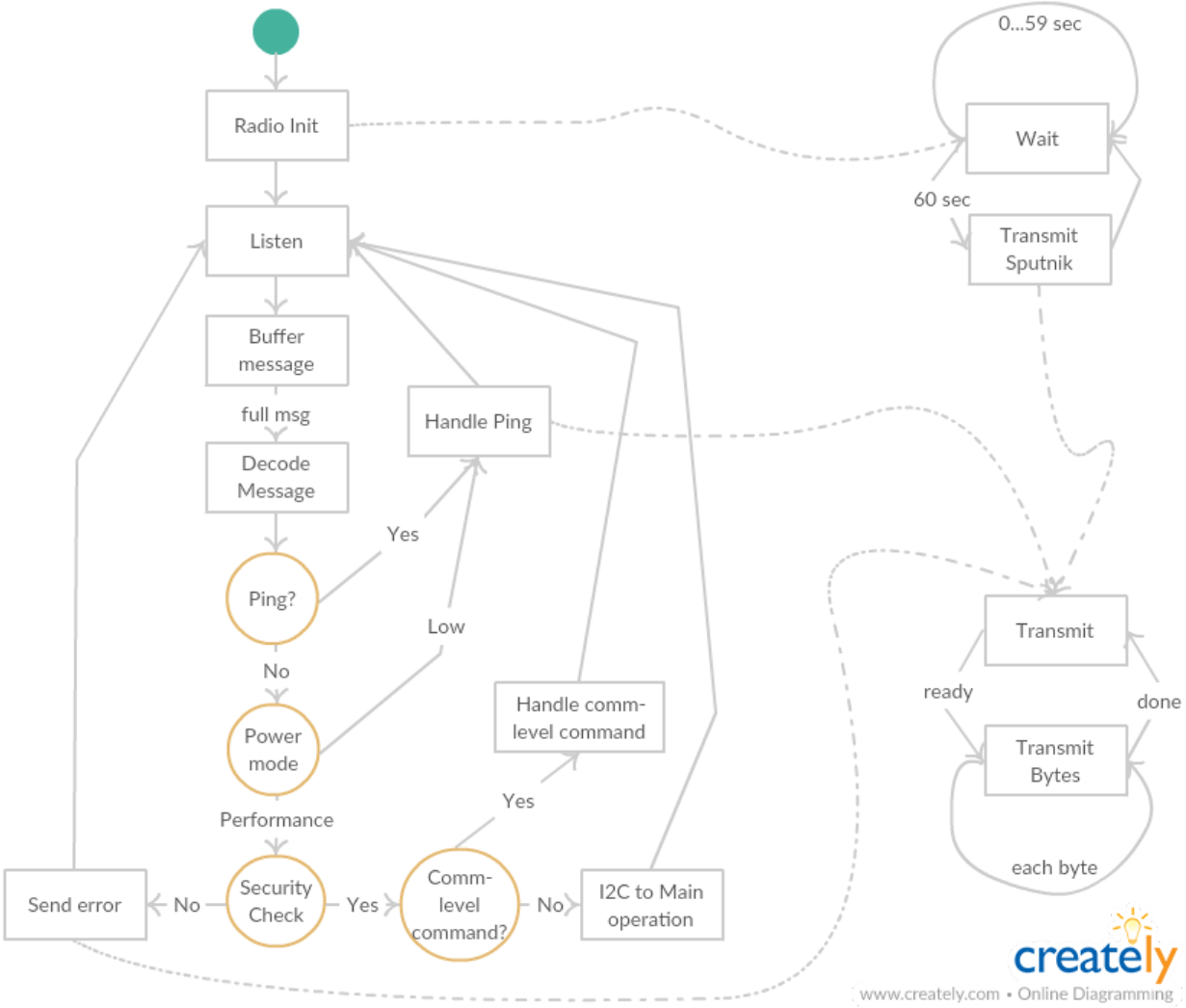


Figure 6. State machine for the Communication subsystem

II.2.I Functions and Interfaces Implemented

II.2.I.I Ping Test

This is the most basic interface of the communication system. The Communication board will be requested 'ping test' from the ground control occasionally. The ping test is one of the ICMP used for checking the current networking status. The test will let the ground control be convinced about the network connection with the satellite. It assures of the connection if it successfully receives a data packet back from the ground server when it sends one to there. The ping test service will be running all the time when the board is on.

II.2.I.II Sputnik

Once the Communication board is on, the Sputnik would be always on. It will send a simple signal in every 60 seconds. The signal of Sputnik works similarly to the ping as it is more likely a

ghost signal. The implementation is to let the function wait for 59 seconds and transmit when it reaches the 60th second.

II.2.I.III Security Check

When the satellite is available to handle the commands other than ping test, the security check is necessary, as the commands can directly affect to the satellite. It will check who is a sender of the command. Processing the commands from unknown sender could be dangerous to CougSat I because the commands can directly affect to the satellite. If the command is from unauthorized sender, the satellite would not handle the command and it will let us know about the error event.

II.2.I.IV Communication-level Commands Processing

There are some commands that are classified as low-level commands. The communication board could digest the commands by itself. So, these commands is called as communication-level commands mostly. The commands do not require the complicated process to be operated. The commands are about getting the status of CougSat I, like getting battery level, power mode, and location, at the moment that the board gets the commands. This service might be correlated to the main board request of Attitude board.

II.2.I.V Transmission to Ground Station

As being a board for the communication, the transmission is the most important service of the CougSat I. It serves as sending data from the satellite to the ground station. It works as transmitting each bytes of the buffer per one time. This service is provided when it handles ping and it sends error message about the command from unauthorized senders who could not pass the security check. For this process, CougSat I will carry AX.25 protocol which is suitable communication protocol for amateur radio operation.

II.2.2 Preliminary Tests

The first thing to test the communication node is checking if the board is on. The main node is in charge of “boot” and turns the all off nodes on. After the main node publishes to turn on to the communication node, the I let the comm node gets the turning on sign by subscribing the main node using the callback function and then turns itself on.

I implemented the code for the communication node to test some services it is supposed to support. Sputnik keeps transmitting “Alive” after waiting 59 seconds. Ping function is called when the input command is “ping” and simply transmits the message saying “Ping”. The communication node also has an implementation of the function that determines if the command is communication-level command. If the command is same as “Battery Level”, “Power Mode”, or “Location”, it calls the commLevelHandler function to process the commands inside of the node. Otherwise, the commands should be sent to the main node. However, as the implementation could not contain the radio simulator and does not have the ground station node yet, the function for transmit was not able to be tested.

II.3 Main

Unfortunately, we were unable to make a state machine diagram for the main board because we ran out of time this term. As aforementioned, we decided to separate inner and outer satellite communications in the simulator because Cougs in Space eventually decided to have them on separate circuit boards. Basically, the Main board is the main point of inner-board communication such that all of the subsystems are connected to the main board and not to each other.

II.3.1 Functions and Interfaces Implemented

II.3.1.1. Boot

The main board will turn on all boards. This function is fully implemented, so it just needs to be tested when the satellite switches between power modes.

II.3.1.2. System Check

After the satellite is booted, the main board will make sure that each subsystem circuit board is behaving as it should. The system check first checks that all boards are on, then it checks that there is no freezing, overheating, or bit flips from radiation. This function checks that all boards are on before proceeding, but the check itself it not yet implemented because not all functions are implemented.

II.3.1.3. Main Board Requests

The main board will listen to the power board. If the power board switches power modes, the main board will turn off the appropriate board or boards to put the satellite in that power mode. The main board will listen to the communication board as well. If the ground station contacts the satellite and asks for the satellite's location, the communication board will relay the message from the ground station the main board, and the main board will relay that message to the attitude board. The main board will then wait for the attitude board to give its location, and then it will send the message back to the communication board. This function is partially implemented; it communicates with each board, but all commands have not yet been determined.

II.3.2 Preliminary Tests

I first made sure the main node was subscribed to the "attitude" topic so that it could start running after detumbling.
For Boot, I turned on each node by publishing "Turn on" to each subsystem topic.
For System Check, I confirmed that Boot was successful by checking that each board was on. I had the main board publish a message to each board's topic and waited until each board confirmed that it was on before proceeding.
For Main Board Requests, I listen for messages on the "attitude", "communication", and "power" topics by making sure the messages are printed in the main node's terminal, inside of each topic's callback function.

II.4 Power

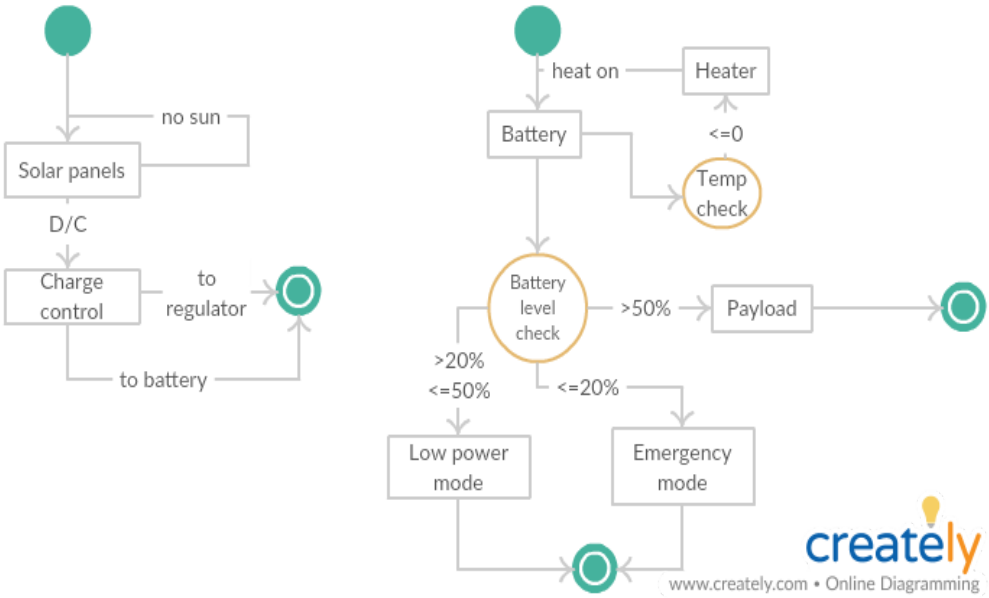


Figure 7. State machine for the Power subsystem

II.4.1 Functions and Interfaces Implemented

II.4.1.1. Engage Solar Panels

This function engages the solar panels as soon as the CougSat I is done detumbling and providing sufficient power to the electrical subsystems, minimizing power drain from the batteries, and ensuring efficient recharging of the batteries.

II.4.1.2. Charge and Discharge

Generally, the battery charger is connected directly to the battery, which in turn is connected to the other electrical subsystems. That is, the battery begins to gain charge when the battery is less than 100% charged and there's no eclipse. The battery begins to discharge once there's eclipse and the battery is fully engaged, supplying power to all other subsystems.

II.4.1.3 Switch Power Modes

The switch power modes function switches between modes depending on the battery level percentage. If the battery percentage is greater than 50%, the payload mode is activated, meaning that all subsystems is fully activated. If the battery percentage gets to or goes below 50% but greater than 25%, the low power mode is activated, turning of all subsystems that are not needed. And once the battery percentage gets to 25% of below, the emergency mode is activated, shorting off all subsystems.

II.4.2 Preliminary Tests

We tested the power board by waiting for a command from the main board to turn on the power board. After receiving the message to turn on, the power board turns on and engages solar panels. As of now, this just means all N solar panels turn on. The power board then checks the battery level and determines which mode the satellite should be in, and reports that back to the main board.

II.5 Sleep

Sleep is not a board and it only does one thing, so we did not make it its own state machine. It is the first state after “Release from Launch” in the general state machine in Figure 1.

II.5.1 Functions and Interfaces Implemented

II.5.1.1. Sleep

The sleep function gets the time that the satellite is released from launch and waits 10 seconds before announcing that it is done sleeping. Theoretically, the sleep function would behave the same for 1800 seconds or 30 minutes, but we do not want to make the user wait 30 minutes for sleep mode to execute in a simulation.

II.5.2 Preliminary Tests

I first made sure the sleep node was able to publish to the “attitude” topic, since the satellite detumbles after it is done sleeping.
For sleep, I made sure that the initial release time was stored by getting the variable and printing it to the screen. I then printed out “Sleeping…” once every second for ten seconds. I then tested different amounts of time under 30 seconds. After confirming that sleep worked for a specified number of seconds, I made sure that it publish that it was done sleeping on the “attitude” topic so that the satellite could begin detumbling.

III. Alpha Prototype Demonstration

III.1. Summary of the Demonstration

We began by showing the audience the general state machine from Figure 1. We then explained the individual subsystem state machines (Figures 5-7), and compared the general state machine with the ROS RQT graph generated in Figure 4. Finally, we demonstrated the simulator as seen in Figures 2 and 3.

III.2. Your mentor’s comments/suggestions on the prototype

After the demonstration, we got some suggestions that could be useful for improving the alpha prototype and the further developments from our mentor, Dr. Aaron Crandall. The most basic comments implied us to be more specific about the purpose of the project. We have focused on the behaviors of the satellite for the alpha prototype. However, in the demonstration, our client, Cougs in Space, seemed to be confused about the reason of the existence of the project. A lot of this comes from the fact that we are developing our simulator from the top down and most, if

not all, subteams are developing all of their software from the bottom up and have not yet thought about some of the high-level behaviors our simulator will exhibit.

Crandall also suggested that we be more clear about the softwares we will use in the future; that is, we should choose between building our own simulation software with our ROS state machine or if we want to utilize a tool like STK or something like it. We will consider all possible approaches and choose one as the most usable option for the beta prototype.

The last suggestion he had for us was to spend more time understanding how state machines work and why we use them. He found some inconsistencies in our subsystem state machines, such as using different terms for the same states, and he thinks that some of us are confused about the fundamental differences between the states and variables.

III.3. Your mentor's questions to your team and your responses to those questions.

We were fortunate enough to get written feedback from Dr. Crandall. He had three main questions and comments for us:

1) Ensure that you continue to develop the documentation and data delivery requirements for the various other groups you need information from. Basically, the Cougs in Space engineering groups will need to be delivering you data such as CAD drawing files, cricut designs, power requirements, code examples, etc. What formats do these documents need to be for the CougSat I simulations to be more effective?

Since we are still deciding between using STK and ROS Kinetic with Gazebo, any format will work and we will translate the information once we make our decision. There is no easy way for Cougs in Space team members to add their designs to ROS without collaborating with us, in part because we are still developing the underlying simulation software. On the other hand, STK supports MATLAB integration so MATLAB files would work best with that simulator.

2) Can your system be integrated into the CougSat I engineering as a testing system?

That is the goal of our beta prototype. As of now, our alpha prototype state machine behaves the exact way that each subsystem board is designed, and environmental factors are not taken into account. The behaviors are accurate, but the hardware representation and component temperatures are not taken into account. This is due to the fact that we were not given individual circuit board or sensor specifications and also because we did not have time to integrate our state machine into a physics library. As mentioned above, the decision to proceed with our beta prototype design using STK or ROS depends on what resources we can utilize in a timely manner and what can give us the most accurate simulation.

3) There's a tradeoff in expanding your own state machine out versus using another tool like STK. Please make sure to note these differences and how you expect to progress with the project. Building your own simulator is possible, but getting the full integration of many systems

(especially mechanical ones) into it will be tough. Using STK or something like it might make the project start to flow better. There's still room for your own simulations and working with ROS because it should map onto the behaviors of the satellite state machines, but which approach will be more usable over time?

In the long run, STK would be more usable because that software has detailed documentation, up to date tutorials, customer service, and is actively maintained. Our simulator would have decent documentation, maybe a tutorial, no maintenance, and customer service would not be a guarantee. We implemented our alpha prototype state machine in ROS with the idea that we would like to create a model of each subsystem circuit board, encapsulate all of the smaller models into one larger model, and apply physics to the larger model. This would allow Cougs in Space to run multiple, customized tests on CougSat I. But as mentioned above, it may be more realistic to find a way to integrate our state machine into STK or translate it into a MATLAB model, which we know STK supports.

IV. Future Work

Our beta prototype will graphically simulate all of CougSat I's states and behaviors from launch to orbit, including communication and control systems and resource management. Having the simulation model will help us and the members of Cougs in Space discover errors and confront problems that might happen in order to improve the satellite and prevent failure in space.

Our alpha prototype established the inner satellite behaviors. In order to accomplish our goals for the beta prototype, we will finish the remaining states in the subsystem state machines, as well as allow for communication with a simulated ground station. We hesitated to develop the ground station as one of our alpha prototype, but we decided to do it for the beta prototype to test the communication node's behavior. Also, having the ground station would make the simulation of each commands from the beginning step to the processing. The rest of the unfinished functions were specified in the subsections of II.1, II.2, II.3, and II.4.

After all of those functions are implemented, we will make sure our data and designs from the Cougs in Space engineering groups are up to date in order to develop a digital model of CougSat I. We still need to determine if it is in our clients' best interest for us to use our state machine in STK or implement it as a model using in a simulation engine called Gazebo. Our decision of which tool to use will be based on which tool case best represent the physical behaviors of the satellite, both inside and outside of it. The model will physically represent the designs of each on-board component, will behave in accordance with the state machine from our alpha prototype, and will follow the laws of physics.

After we determine whether to use STK or Gazebo, we will add physics to the CougSat I model. Once we are able to simulate physics on the satellite, we will be able to improve the existing states so that they more accurately behave when being acted upon by forces such as gravity. We will also be able to add temperatures to each board and determine when components freeze and overheat. We will be able to implement each ROS node as a model inside of Gazebo, and

then we can connect all of the smaller models together to form the satellite. We will need to further investigate how to integrate our ROS nodes into STK.

We will also improve our user interface. Currently, we have a launch file that runs all five ROS nodes and the ROS core node if it is not already running. The reason we did not use this launch file in the alpha prototype demonstration is because it launches all of the nodes, but does not display the messages in each topic as they are published and received like the individual terminals do. Watching the messages in each topic becomes less necessary when the graphics are integrated, as we will be able to see what behavior(s) the satellite is not exhibiting. We hope to allow the user to enter in various launch conditions such as initial battery level, and possibly launch location.

V. Glossary

AX.25: Amateur X.25; a data link layer that is designed for use on amateur radio packet network which has the nearly global coverage.

Detumble: Stop uncontrollable tumbling of a spacecraft.

Gazebo: A simulation software that supports ROS and includes Physics and Graphics libraries.

GUI: Graphical User Interface; an intuitive way for users to interact with the software.

ROS: Robot Operating System; an operating system that is useful for creating state machines and simulations.

RQT: A GUI that displays the network of ROS processes that are processing data together. For our purposes, this includes nodes, messages, and topics.

Sputnik I: The first satellite launched by the Soviet Union in 1957. It transmitted electronic signals to the Earth.

STK: Systems Tool Kit. A satellite simulation tool made by AGI that allows users to customize the size and launch location of the spacecraft.

ICMP: Internet Control Message Protocol. It is a protocol allows to determine if the current network is connected properly.

VI. References

Rouse, Margaret. “What Is Ping? - Definition from WhatIs.com.” *SearchNetworking*, Feb. 2009, searchnetworking.techtarget.com/definition/ping.