STAR-Satellite

Report

Author:	Christopher Steffen, Maximilian Roth,
	Joshua Redelbach, Syed Faraz Ali,
	Shreeja Shrijan Shrestha, Sumit Mandal
Course:	FloatSat Praktikum
Working time:	Start 16.10.2023
	Finish 15.02.2024

Julius-Maximilians-Universität Würzburg

Lehrstuhl für Informatik 8

Emil-Fischer-Str. 70, 97074 Würzburg

Contents

1 Introduction **2**
2 Mission **2**
3 Mechanical **3**
3.1 STARSat . . . 3
3.1.1 Design Approach . . . 3
3.1.2 Docking Mechanism . . . 3
3.1.3 Modelling and Assembly . . . 5
3.1.4 Final CAD Model . . . 7
3.2 Mock-Up . . . 8
3.2.1 Design Overview . . . 8
3.2.2 Modeling and Assembly . . . 8
4 Electrical **10**
4.1 STAR-Sat . . . 10
4.1.1 Main Power . . . 10
4.1.2 Power Conversion and Monitoring . . . 11
4.1.3 Logic . . . 12
4.1.4 RPI Zero 2 . . . 12
4.2 Mockup . . . 12
5 Datalink **13**
6 Groundstation **13**
6.1 Introduction . . . 13
6.2 Tools and Libraries . . . 13
6.3 Groundstation windows . . . 14
7 ORPE **15**
7.1 LCCV and Performance . . . 16
7.2 Timing Realignment . . . 16
7.3 Quick LED Identification . . . 17
7.4 RODOS Integration . . . 17
8 Embedded **18**
8.1 Structure . . . 18
8.1.1 Inner Loop . . . 18
8.1.2 Outer Loop . . . 19
8.1.3 Communication . . . 20
8.1.4 Docking . . . 20
8.1.5 Raspberry Pi . . . 21
8.1.6 Electrical Monitoring . . . 21
8.2 Coordinate frames . . . 21
8.3 Modes . . . 22
8.3.1 Startup . . . 23

8.3.2 Standard 24
8.3.3 Mission 25
8.4 Mockup 26
9 Results **27**

List of Figures

Figure 1: FloatSat Rack and Pinion set 4
Figure 2: Visualisation of Magnetic field in normal and Halbach arrangement. Source: here 5
Figure 3: Assembly of Base Plate Level Components 6
Figure 4: Assembly of Middle Plate Level Components 7
Figure 5: Assembly of Top Plate Level Components 7
Figure 6: Fully Assembled STARsat CAD Model 8
Figure 7: Exploded View And Front View of Mockup 9
Figure 8: STAR-Sat electrical structure 11
Figure 9: Mockup electrical structure 12
Figure 10: Summary Window 14
Figure 11: IMU Window 14
Figure 12: Mission Window 14
Figure 13: Telecommand Window 14
Figure 14: Debug Window 14
Figure 15: Embedded code structure 18
Figure 16: Coordinate frames of Mockup 22
Figure 17: Coordinate frames of STAR-Sat 22
Figure 18: Embedded mode flowchart 23

List of Tables

Table 1: Designed Specifications of the Pinion Gear 4

1 Introduction

Recent years have seen a trend towards using cubesats to test and validate experimental technologies. Cubesats are a small, lightweight low cost platform that allows for ride-sharing on launchers to also greatly reduce cost.
A future possibility is large scale structures that can assemble themselves in space and also space junk is becoming an even greater threat to the space industry. To solve both problems we must be able to attach or dock with other satellites or object. In the case of space junk, this must be possible without communication between satellites.
STAR-Sat (Satellite Tracking and Autonomous Retrieval) was developed as a proof-of-concept to test the searching, tracking and autonomous retrieval of non cooperative satellites. For this, a mock-up satellite was build, that is similar to the STAR-Sat, but without communication and the camera is replaced with 12 LEDs to be able to identify and estimate its pose using ORPE.

2 Mission

STAR-Sat's mission is comprised of multiple phases: Searching, Tracking, Arm extension, Docking, Retrieval. In order for the mission to be reliable, STAR-Sat will also do some checks after some phases to ensure they were successful. The following is an overview. The exact implementations are explained in section 8.3.3.
STAR-Sat's initial task is to find the mockup. It does this by rotating and waiting for the camera identify the mockup's LEDs. Once the LEDs have been identified, STAR-Sat will then point towards the mockup center while also tracking its position, correcting for any movements. The docking arm can now be extended shortly before where the mockup's docking port will be after docking, and STAR-Sat will during this time begin estimating the mockup's rotation velocity. STAR-Sat will calculate the exact timing to extend the docking arm so that both the docking arm and mockup's docking port will meet when they arm parallel. Once docked, STAR-Sat will retract the arm pulling the mockup towards STAR-Sat to finalize the docking, thus completing its mission.

3 Mechanical

3.1 STARSat

3.1.1 Design Approach

The design strategy for both, the primary floatsat (STARsat) and the mock-up, depended on how the mission would look like. In particular, the kind of docking planned to achieve between the two models and the docking mechanism used.

To achieve novelty in the mission, the team decided on being ambitious with the design. Conducting docking between two satellites with a docking arm system was a certainty from the beginning. Various kinds of docking approaches and arm designs that could achieve them were considered and then were traded-off by comparing their feasibility in terms of designing, manufacturability and testing. In engineering, while trying to achieve something new, a rapid prototyping strategy works well. In such a strategy, prototypes of the mechanism in question are manufactured, tested and validated quickly. This helps in improving the design iteratively in a fast and efficient way. However, since this is a student project aimed to be realised in a university laboratory with limited resources, rather than in the industry, such a strategy could not be implemented. Hence, a docking mechanism with a low risk of failing, high adaptability and easy manufacturability was to be chosen. A docking arm based on a rack and pinion system with a docking port situated on the mock-up fulfilled this criteria.

3.1.2 Docking Mechanism

A simple rack and pinion mechanism was decided to be used to imitate a docking arm that extends outwards from the primary satellite towards the spinning mock-up. The first step was designing the pinion gear while keeping in mind that the arm needs to extend and retract very precisely. Also for this reason, a stepper motor was chosen to rotate the pinion as it rotates in discreet steps resulting in high position accuracy and produce a good amount of torque at lower speeds, which would later be great for the mission extension of pulling the docked mock-up closer to the STAR-sat.

A spur gear has less sliding friction between the teeth which makes them more efficient at low speed operations. Moreover, it is easier to manufacture with high precision using a 3D printer. Hence, a spur gear with low module was to be designed so as to accommodate more number of teeth in a small sized gear that could fit well on to the satellite along with the racks.

$$Module = \frac{PitchCircleDiameter}{No.ofTeeth} \tag{1}$$

$$Pitch = Module * \pi \tag{2}$$

Implementing an involute gear profile with a standard pressure angle of 20° and using these standard gear design formulae, a pinion with the following specifications was designed:

Table 1: Designed Specifications of the Pinion Gear

Parameter	Value
Pressure Angle	20°
Module	0.85
Number of Teeth	20
Pitch	2.67 mm
Pitch Circle Diameter	17 mm
Pitch Circle Length	53.4 mm
Root Fillet Radius	0.5 mm
Inner Hole Diameter	5 mm
Gear Thickness	12 mm

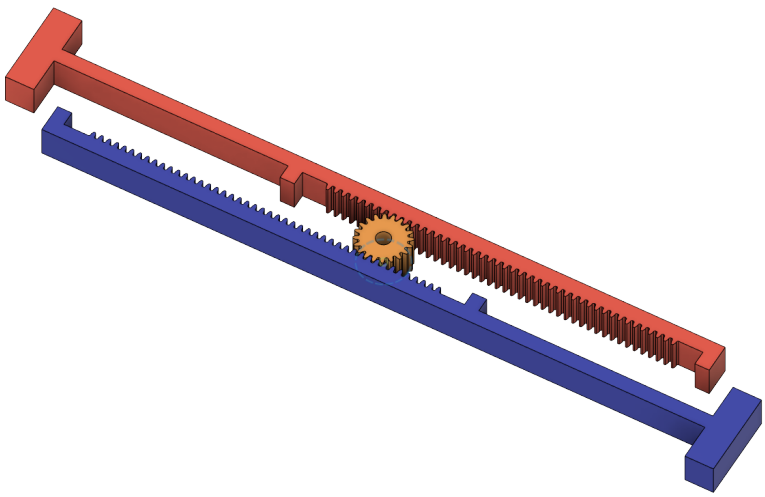

Figure 1: FloatSat Rack and Pinion set

As noticeable, a pitch that small helped in gaining that accuracy in moving a certain distance required by the racks and a pitch circle length of greater than 50 mm meant that the pinion would only require about 2 full rotations to make the rack extend to

a maximum of 10 cm, within which distance the two satellites were supposed to be placed for the mission. The rack was directly designed from the gear to keep the same teeth dimensions. The gear thickness was chosen to be 12 mm such that the rack could also have that height and an appropriate docking object could have enough space to be fixed on to the head of the rack. A second rack was just used to balance the satellite while it floats.

The racks also have a stopper on each end to prevent them from running out of the pinion. Each rack sits on top of a guide that keeps the rack attached to the pinion and helps it move in a straight line when extending out. A limit switch is fixed on to the back of one rack that gets pushed when the racks are fully retracted for calibrating the controller.

For the docking head, it was decided to use neodymium magnets and some kind of small ferromagnetic metal material on the mock-up. The issue with using strong magnets is that their magnetic field interacts with the IMU in the satellite controller and disturbs the readings. But to counter this, the magnets chosen were very small cubes of length 1 mm and fixed in a 3D printed holder in a special kind of arrangement called the Halbach Array. This arrangement augments the magnetic field on one side of the array while bringing the field strength down to almost zero on the other side.

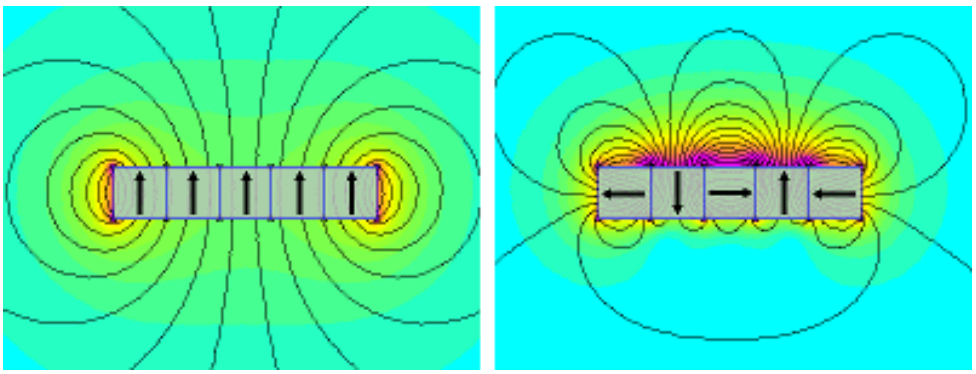

Figure 2: Visualisation of Magnetic field in normal and Halbach arrangement. Source: here

3.1.3 Modelling and Assembly

The floatsat is built in 3 main levels with different kind of components being on different levels. It consists of three kinds of components - laser cut parts, 3D printed parts, and other parts bought from outside. The laser cut parts are made of Polymethyl methacrylate (PMMA), commonly known as acrylic or fibre glass. The 3D printed parts are made of Polylactic Acid (PLA). A brief description of each level of the floatsat is as follows:

- Base Plate Level
  The base plate level consists of the general components required to build a floatsat - Reaction wheel, two batteries and a casing to hold each battery in. The casings were designed in such a way to use less material as possible. Since all the electronics were designed to be placed on the top most level of the satellite, the casings were made a bit longer than the battery's length such that the wires of the batteries and the reaction wheel could pass through them on to the upper levels. The diameter of the base plate is 191.4 mm, such that it could go as deep as possible inside the floatsat hemisphere and keep the CG low. Four spacers are used to support the upper levels of the floatsat on top of the base level. The base plate assembly is shown in the figure below:

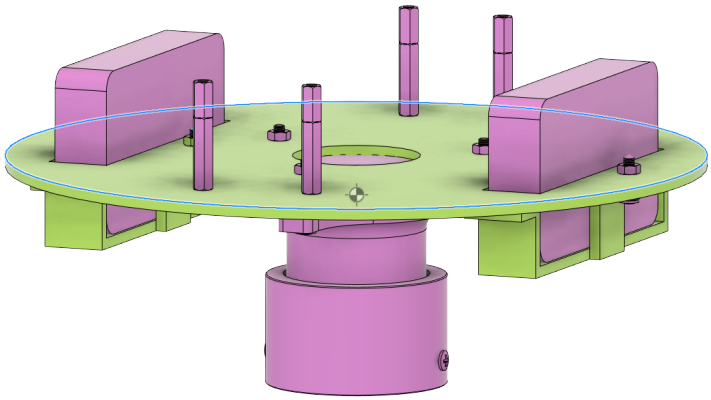

Figure 3: Assembly of Base Plate Level Components

- Middle Plate Level

  This level holds the docking mechanism of the satellite. A stepper motor is fixed on to the plate from beneath. On top of the plate, there is the docking mechanism consisting of a pinion fixed on the motor's shaft, surrounded by two racks sitting on top of their respective rack guides. The pinion, racks and guides are all 3D printed, with the pinion being printed via a high precision nozzle. The figure below shows the assembly of the middle level of the FloatSat.

- Top Plate Level

  The top plate is just a level to fix the electronics mount on top of it. The holes are position in such a way that the wires can easily pass through all levels and

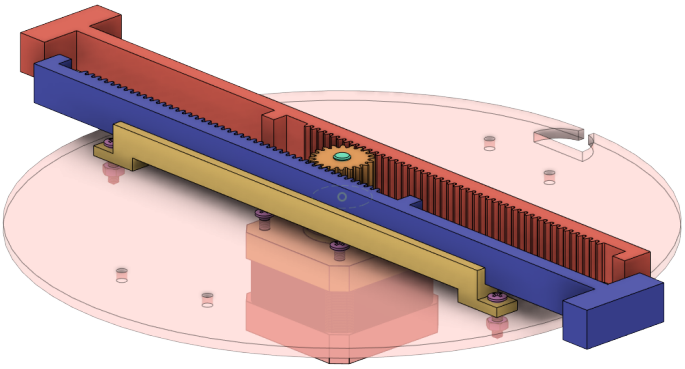

Figure 4: Assembly of Middle Plate Level Components

connect in to the electronics mount. The location of spacers to be used as supports was chosen such that the camera on top could be aligned with the direction in which the docking arms would extend out. The top plate level assembly is shown in the figure below.

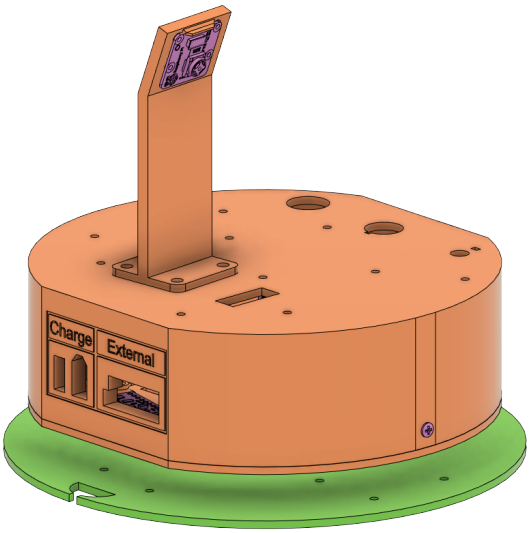

Figure 5: Assembly of Top Plate Level Components

3.1.4 Final CAD Model

The final CAD model of the completely designed and assembled STARsat is shown in the figure below.

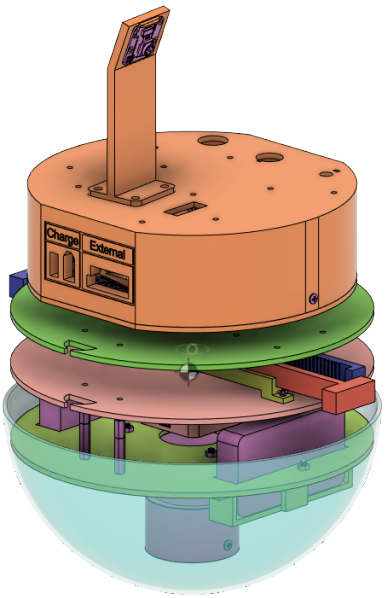

Figure 6: Fully Assembled STARsat CAD Model

3.2 Mock-Up

3.2.1 Design Overview

The main objective of the mockup model is to make a similar design relative to primary StarSat with a magnetic docking system which latches with the extending arm of the StarSat and to make it as lower as possible so that the camera can scan the led pattern for rotational direction and distance between both the models. Prior to fabrication, the entire mockup was meticulously designed using SolidWorks, a Computer Aided Design (CAD) software. This ensured the mockup would meet mission objectives before any physical components were created. The base plates of the Mockup were precisely cut from Polymethyl methacrylate (PMMA), commonly known as acrylic. Meanwhile, the supporting columns and the docking mechanism were fabricated using a 3D printing process, utilizing Polylactic Acid (PLA) filament.

3.2.2 Modeling and Assembly

This section consists of primarily three layers of design stages. This helps us construct the whole design more efficiently and seemingly.

- Bottom layer
  This layer consists reaction wheel and the battery unit. The CAD design of this layer is very similar to main FloatSat to prevent extra workload and errors. The

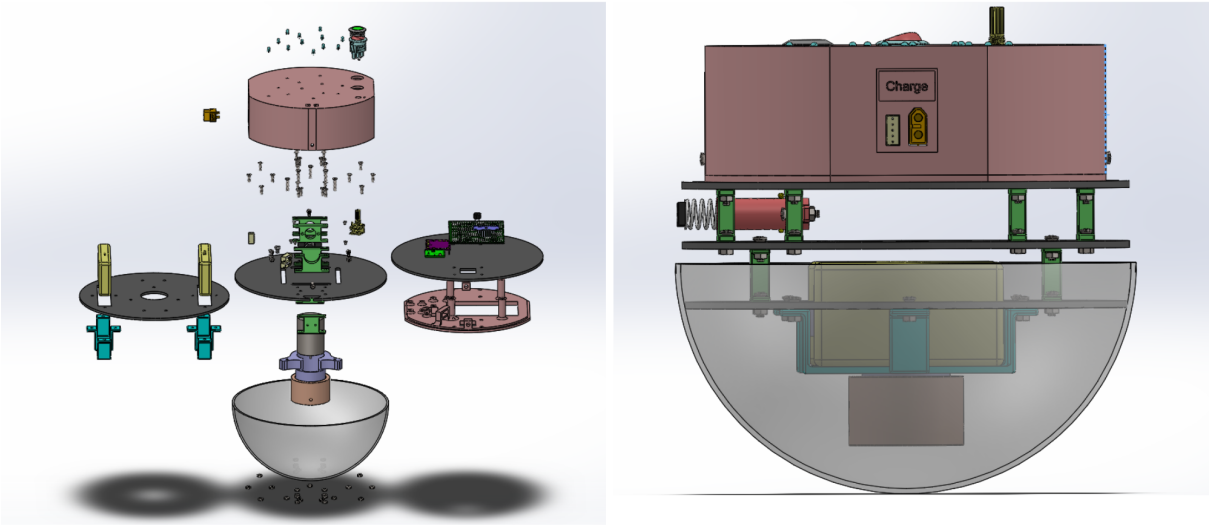

Figure 7: Exploded View And Front View of Mockup

supporting columns were optimized to minimize the overall height of the model. Additionally, the design ensured that the extension arms aligned perfectly with the docking magnets in the mockup.

- Middle layer
  This layer incorporates a crucial component: a docking magnet assembly paired with counterweight columns to absorb rotational momentum. Initially, a simpler magnet design was considered. However, the need to handle high angular speeds during latching necessitated the implementation of linear differential springs within the docking system. This allows for successful latching even at high speeds. Additionally, pre-existing slits in the middle plate enable post-assembly calibration of the center of mass for optimal performance. To achieve a clean design, all electrical components are housed on the top layer. Slits strategically placed opposite the docking system facilitate the routing of wires from the bottom layer to the top, where they can be directly mounted on the electrical

plate.

- Top layer
  The design of the top layer prioritized efficient assembly and integration with the electrical components. This layer needed to seamlessly attach to the middle layer via the four supporting columns, while also facilitating a secure connection to the dedicated electrical mount at the top. This modular approach offered a key advantage: the ability to concurrently work on the electrical assembly and the top layer itself. By housing all electrical components and wiring on the top layer, we gained convenient access for modifications and troubleshooting throughout the development process.

4 Electrical

The electrical system was initially designed for STAR-Sat, but then adapted for the mock-up. Therefore both STAR-Sat and mock-up electrical systems are very similar and the embedded software versions can be run on both STAR-Sat and mock-up interchangeably without damage.
Continuing the modular design of separating the mechanical and electrical as much as possible, both electrical systems for STAR-Sat and the mock-up where designed to have as few external connections as possible. This resulted in the mock-up only having a single connector for the reaction wheel and two connectors, one for each battery. STAR-Sat also had two for the batteries and one for the reaction wheel, but also a single connector for the stepper motor and another for the switch to calibrate the stepper motor steps.
This design decision greatly reduced the risk of damaging wires, made disconnecting devices for testing exceedingly easy and most importantly eliminated connection problems that come from using the typical connecting wires and headers while not being as permanent as direct soldering.

4.1 STAR-Sat

The electrical is split into four main parts. The main power, Logic, Power conversion and monitoring and RPI Zero 2.

4.1.1 Main Power

The main power section allows for the battery to be entirely disconnected for STAR-Sat via the power switch. The power button is used to turn STAR-Sat on, at which point the STM32 switches the relay on if no faults are detected. The relay also gives

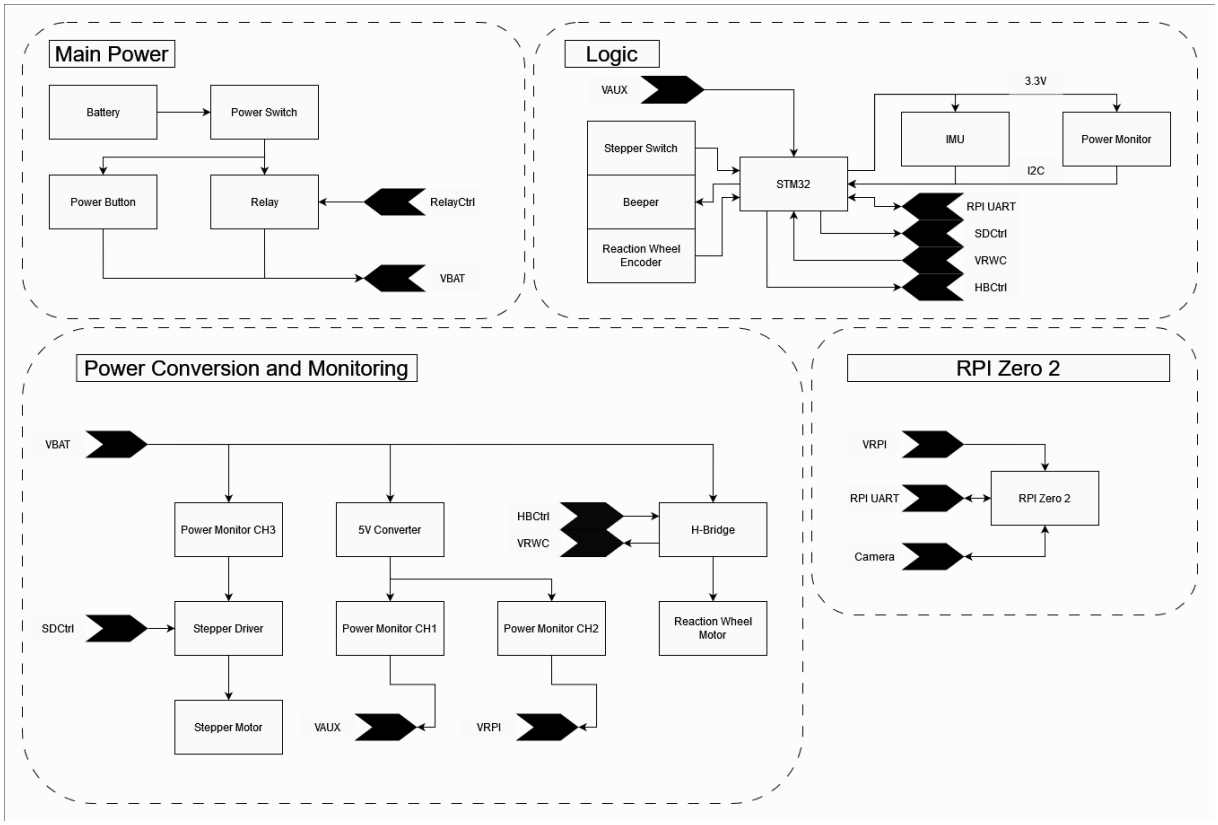

Figure 8: STAR-Sat electrical structure

the STM32 the ability to turn itself off if for example the battery voltage is low or the ground station commands to power off.

4.1.2 Power Conversion and Monitoring

This section will convert the battery voltage to 5 Volts, which is fed into the Power monitors channels. The power monitor channels then go into the RPI Zero 2 or logic sections, allowing us to measure the 5 Volt bus voltage and also the section current usage. The last channel of the power monitor is used to measure the battery voltage and also the stepper motor current usage.

The stepper motor driver (TMC2208) is controlled over 3 GPIO. One is to enable the motor, one give the next steps direction and the last GPIO will make the stepper motor move a single step on a rising edge.

The H-Bridge (TB9051FTG) controls the power given to the reaction wheel motor and also measures its current usage. The H-Bridge is controlled by 2 GPIO, each giving a direction and power in this direction.

4.1.3 Logic

This section contains the main logic and control computer of STAR-Sat. Both the IMU (LSM9DS1) and Power monitor (INA3221) are supplied 3 Volts from a GPIO output by the STM32. Their combined current (5mA + 0.35mA) doea not exceed the max GPIO current of the STM32. This gives STAR-Sat the ability to restart the sensors in the case of an I2C lockup. The stepper motor calibration switch is also directly connected to a GPIO of the STM32.

4.1.4 RPI Zero 2

This section simply contains the RPI Zero 2 and its camera. The current usage is separately measured from the rest of the system, which allows us to monitor when the RPI as correctly started and also shutdown.

4.2 Mockup

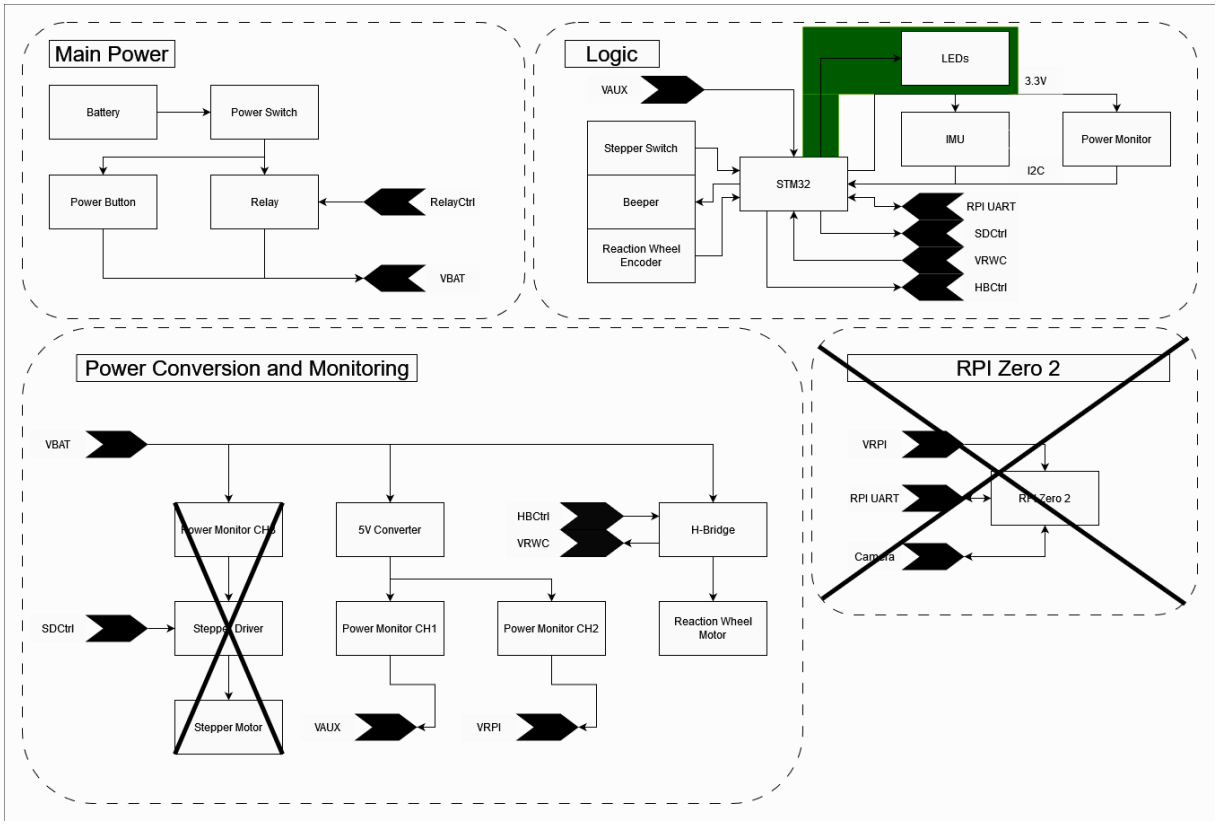

Figure 9: Mockup electrical structure

The only differences of the mockup to STAR-Sat is the entire stepper system and RPI Zero 2 are left out and the addition of 12 LEDs with current limiting resistors. Each of the 12 LEDs are connected to a separate GPIO allowing for individual control of each LED.

5 Datalink

Since STAR-Sat already has the RPI Zero 2 onboard with the ability for WiFi and the STM32 and RPI Zero 2 are connected via UART to communicate with ORPE. We decided to simply use this for the datalink too.
The STM32 sends telemetry data via a RODOS gateway over UART to the RPI Zero 2, where the telemetry data gets forwarded by RODOS gateways over WiFi to the ground station and vice versa for telecommands coming from the ground station to the STM32. Due to the high throughput achievable over WiFi, it should also be possible to stream a video feed from STAR-Sat camera to the Ground Station, but due to time constraints, this was not implemented in STAR-Sat.

6 Groundstation

6.1 Introduction

The ground station for STAR-Sat is a desktop application built to visualize real-time data of STAR-Sat. The application allows users to see sensor data and understand orientation and mission conditions through comprehensible graphs and widgets. It also enables sending commands to the satellite.

6.2 Tools and Libraries

The tools and libraries used by the ground station are:

- Pyside6:
  The ground station is python based QT application built using pyside6, an official Python module for QT.

- Pyqtgraph:
  Pyqtgraph is a GUI library built on PyQT/PySide and numpy. The ground station uses Pyqtgraph for graphical visualization of the satellite's voltage, orientation, motor speed, and angular velocity.

- RODOS:
  RODOS is a real-time operating system for embedded systems. The ground station uses the Python library of RODOS to communicate with the main satellite through WiFi. Groundstation subscribes to five topics with topicID as follows:

  - telemetryContinuous (40)
  - telemetryContinuousExtended (41)

- telemetryCalibIMU (802)
- telemetryControlParam (803)
- telemetryMessage (300)

Groundstation publishes telecommands to topic 51.

6.3 Groundstation windows

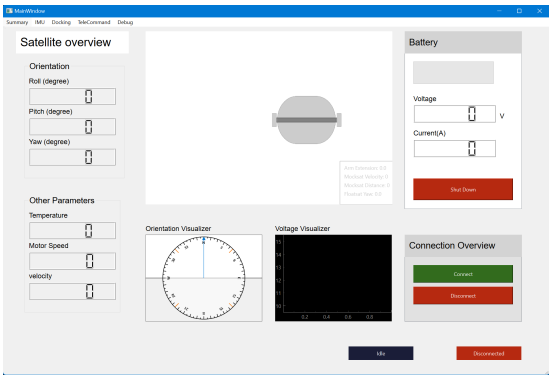

Figure 10: Summary Window

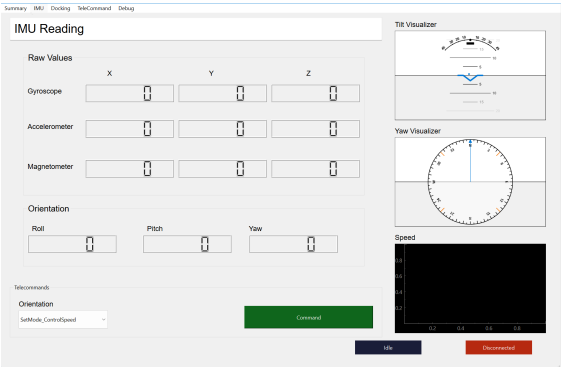

Figure 11: IMU Window

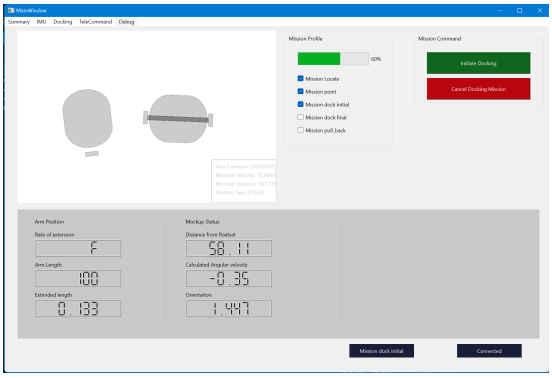

Figure 12: Mission Window

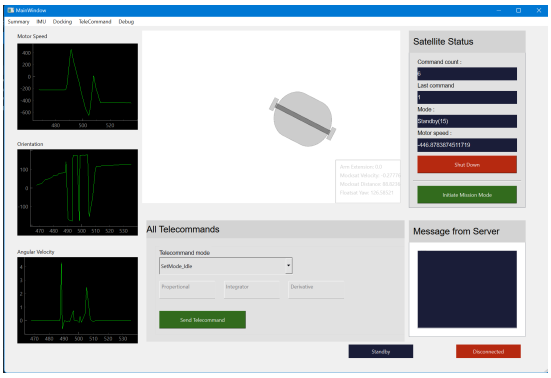

Figure 13: Telecommand Window

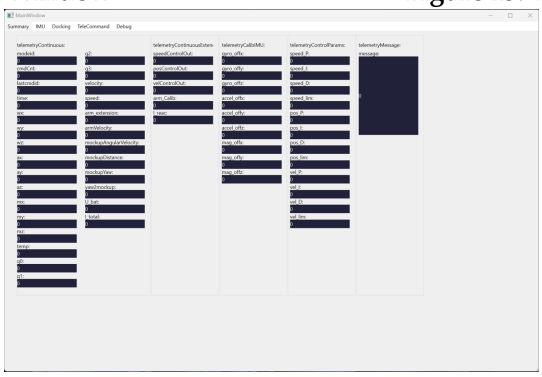

Figure 14: Debug Window

The ground station is divided into 5 windows:

- Summary:
  The summary window (fig 10) contains important information gained from STAR-Sat like the orientation, battery status, temperature, and velocity. The user can connect or disconnect the satellite communication from this window. The user can also send *shutdown* telecommand if necessary.

- IMU:
  IMU window (fig 11) contains raw values sent by gyroscope, accelerometer and magnetometer along with roll, pitch, and yaw calculated from quaternions. The roll, pitch, and yaw visualizer along with the motor speed visualization graph is also present. Users are also allowed to set *position*, *angular velocity*, and *motor speed* of satellite in this window.

- Mission:
  Since the STAR-Sat's mission is to dock to the mock satellite, the docking window (fig 12) guides the user through the steps of docking using a progress bar and a checklist. The user can also *initiate the mission* or *end the mission* through this window. Ending the mission sets the state of the satellite to standby.

- Telecommand:
  The telecommand window (fig 13) is a fully functional window that deals with sending commands to the satellite. The window contains a dynamic form allowing the user to choose between various telecommands. For basic missions like setting position, velocity, and motor speed, the window also contains three graphical interfaces that assist in analyzing the performance of the satellite based on parameters like settling time, overshoot, and others.

- Debug:
  The debug window (fig 14) contains all the data being sent from the satellite as it is without any manipulation.

7 ORPE

STAR-Sat uses ORPE (Optical Relative Pose Estimation) to find and track the mockup's relative position and orientation. Using a camera on STAR-Sat and 12 LEDs on the mockup, ORPE estimates the relative Pose by using the IPPE implementation for solving the Perspective-N-Point problem. ORPE also uses an LED encoding and decoding scheme to identify each LED ID individually and therefore can differentiate between LEDs and other light sources.

Since ORPE was not developed with STAR-Sat and is rather a product of a Bachelor Thesis. The following section specifically goes over the changes made over the course of STAR-Sat's development and testing to improve tracking, reliability and integration into RODOS[2].

7.1 LCCV and Performance

The original implementation of ORPE used the OpenCV VideoCapture system. This was great for simply reading data from cameras or video files, but didn't give much control over the Raspberry Pi Camera V2 camera settings, as it used the legacy camera stack and not the new libcamera system. This resulted in images what were not dark enough and too many other lightsources were found.
Thankfully recently the work was made by a 3rd-Party to use OpenCV with libcamera. LCCV[1] gives control over the camera ISO and shutter speed.
The RPI camera module V2.1 camera ISO is set to -0.1, Shutter speed to 1us. As the Raspberry Pi Zero 2 is unable to meat the processing requirements for a 1920x1080 and 30FPS video stream, it was reduced to 1280x720 and 20FPS, this proved to be more than sufficient for STAR-Sat as the distanced were not more than 50cm and the velocities were fairly constant, but further testing showed that ORPE could still work up-to 13 Meters distance with these settings.

7.2 Timing Realignment

During the development of STAR-Sat we often had the issue where the ORPE was unable to correctly decode the LEDs. The cause was due to an inherent issue with ORPE's coding design as the LED transition became aligned with the camera image timing. RODOS thread timing also made this more sensitive as timing accuracy made the system for sensitive to the alignment issue.
A quick solution was giving the LED coding thread the highest priority. This was not an issue for the rest of the system as updating the LEDs is a fast process. Although this only makes it more unlikely for the coding and encoding to become aligned.
A proper solution is to detect and delay the camera or LED timing by half the frame interval. Detection is done by utilizing the coding parity bits to detect errors, if many errors are detected in a short time frame, ORPE will assume there is alignment issues. The current implementation will simply restart the camera to delay the next frames.

7.3 Quick LED Identification

As the LED identification can take a substantial amount of time, we can use the knowledge of the LED-Model and current pose estimation to quickly identify the LED IDs of points in the image.
This is done by projecting the LED-Model points onto the image plane, and then calculating the standard deviation of distances of the already identified points to characterize the error of the projection. Then we find the best matching unidentified points in the image to the projected LED-Model points. If the distance is within the standard deviation, we can update the image point ID with the matching projected LED-Model point.

7.4 RODOS Integration

Since ORPE is running on the Raspberry Pi Zero 2 in Linux, separately from the actual control software on the STM32 in RODOS, we required some way of communicating the ORPE pose estimations to the control software. The simplest solutions would be using the already available datalink system that utilises the RODOS topic middleware. Initially after simply integrating RODOS into ORPE's build process and running the ORPE loop inside a RODOS thread. There appeared to be a memory leak somewhere inside the program. This was not the case when running ORPE separately. With some further investigation using the analysis tool Valgrind[3], showed the memory leak possibly coming from the libcamera library.
The solution for this problem was to compile and run ORPE and RODOS separately and then have them communicate over a simple IPC (Inter-Process-Communication) protocol. As the total latency from estimation to the control software receiving the estimate must be reduced, the IPC method must also have a low latency. UDP was chosen due to its thread safety, simplicity and throughput. As the communication is running inside the same system, UDP latency should also be extremely low.
RODOS uses port 12314 to send commands to ORPE. Currently this is only in the form of a boolean, where a value of 1 triggers ORPE to startup and begin estimating and a value of 0 will stop ORPE from estimating and reset. ORPE uses Port 41523 to send the pose estimations to RODOS. This also includes a 32bit frame counter that increments when a new image is taken, a boolean indicating of the estimation is valid, an 8bit value containing the number of LEDs that have been identified and the number of points currently seen in the image including the identified LEDs.

8 Embedded

The embedded software was implemented using RODOS on a STM32F4 Discovery Development Board. The structure of the code as well as the logic behind it is illustrated in Figures 15 and 18 and is presented in detail in the following. Sections 8.1 and 8.3 deal with STAR-Sat embedded code and section 8.4 describes the changes made to this code to fit the Mockup.

8.1 Structure

The software can be grouped in four main parts, each consisting of one or two threads which are running the different functionalities. The communications between the different threads and classes is implemented using RODOS topics and sometimes by using semaphores. An overview is provided in Figure 15.

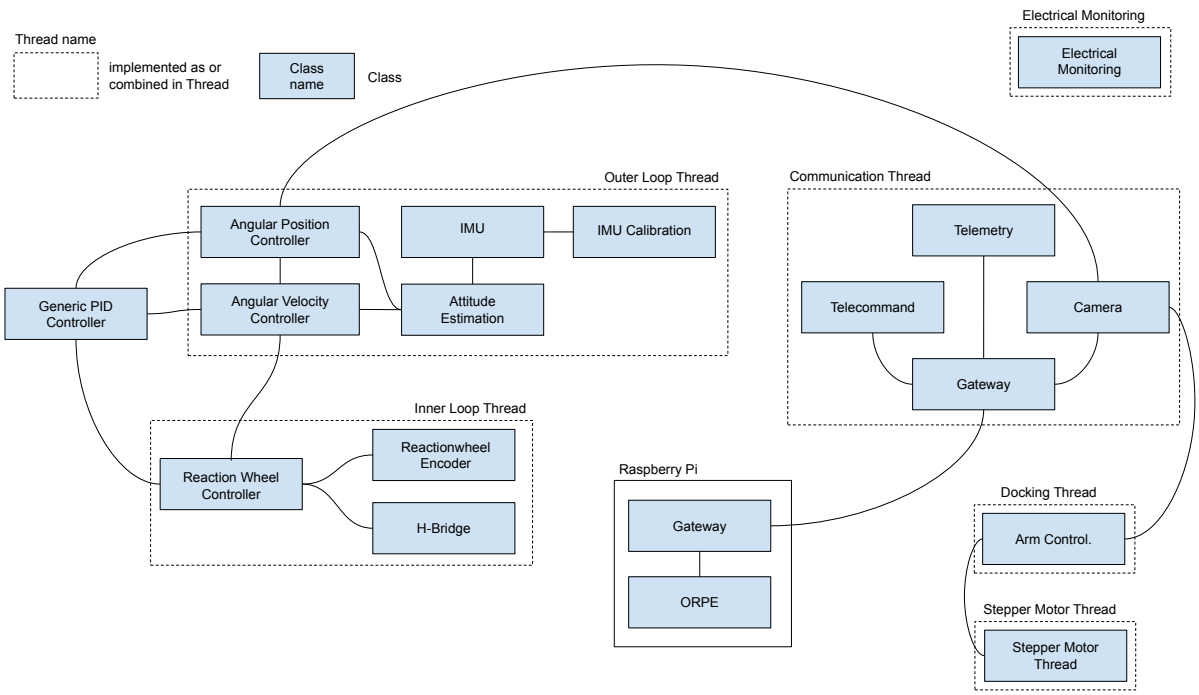

Figure 15: Embedded code structure

8.1.1 Inner Loop

The first part is the *Inner Loop Thread*, including everything for controlling the reaction-wheel. The thread is running at a frequency of 50 Hz.

- *HBridge*: A class implementing the driver for sending the reaction wheel a Pulse-Width-Modulation (PWM) signal corresponding to the desired speed and also limiting the current to the motor.

- *Reactionwheel Encoder*: A class implementing the driver for reading the speed of the reaction wheel.
- *Reactionwheel Controller*: A class implementing the PID controller for controlling the reaction wheel speed by using the data of the *Reactionwheel Encoder* and commanding the desired voltage to the *HBridge*. During testing, it has turned out that a PI controller is sufficient, leading to following control gains: $K_P = 0.3$ and $K_I = 0.05$, $K_D = 0$. Additional to the basic structure of a PID controller, an anti-windup control was added.

8.1.2 Outer Loop

The second part is the *Outer Loop Thread*, including everything for attitude determination and control. The thread is running at a frequency of 5 Hz.

- *IMU*: A class implementing the driver for reading the raw data (using I2C), converting the raw data and applying the calibration for each sensor of the IMU (Inertial Measurement Unit): gyroscope, accelerometer, magnetometer and temperature sensor.
- *IMU Calibration*: A class implementing the calibration routine for the gyroscope, accelerometer and magnetometer. For the gyro and accelerometer the offset on all axis is being calibrated and for the magnetometer the offset in the X-Y plane is being calibrated.
- *Attitude Estimation*: A class implementing a *QEKF* (Quaternion Extended Kalman Filter) for estimating the attitude of STAR-Sat (3 DOF orientation and 3 DOF angular velocity) and the gyro drift using the calibrated data provided by the *IMU*.
- *Angular Velocity Controller*: A class implementing the PID controller for controlling the angular velocity of STAR-Sat by using the estimated attitude from the *QEKF* and commanding the required speed of the reactionwheel to the *Reactionwheel Controller*. During testing, it has turned out that a PD controller is sufficient, leading to following control gains: $K_P = -20$ and $K_I = -200$, $K_D = 0$. Additional to the basic structure of a PID controller, an anti-windup control was added.
- *Angular Position Controller*: A class implementing the PID controller for controlling the orientation about the z-axis (Yaw angle) of STAR-Sat by using the estimated attitude from the *QEKF* and commanding the required angular velocity of STAR-Sat to the *Angular Velocity Controller*. During testing thefollowing control

gains were determined: $K_P = 0.8$ and $K_I = 0$, $K_D = 0.05$. Additional to the basic structure of a PID controller, an anti-windup control was added and the derivative part only takes the measurement, not the error.

8.1.3 Communication

The third part is the *Communication Thread*, including everything for the data link between the STM32F4 board and the ground station via the Raspberry Pi. The thread is running at a frequency of 2 Hz.

- *Telemetry*: A class for collecting and preparing all telemetry data that needs to be sent to the ground station via RODOS topics.
- *Telecommand*: A class for processing the received telecommand topic from the ground station.
- *Gateway*: RODOS gateway for sending the telemetry data to and receiving the telecommands as well as the camera data from the Raspberry Pi using UART.
- *Camera*: Implements transformation between the various coordinate frames of STAR-Sat to process the received camera data: determine the orientation (Yaw angle) of the Mockup as well as the distance from the docking arm to the docking port on the Mockup and the angle from STAR-Sat to the Mockup. It can also be checked if the camera data is a valid estimation. The coordinate frames are elaborated upon in section 8.2.

8.1.4 Docking

The fourth part consists of the *Stepper Motor Thread* and of the *Docking Thread*. The *Stepper Motor Thread* implements the driver for controlling the stepper motor which is used to extend and retract the docking arm. Two GPIO pins are used, one for indicating the direction of the step and one for indicating the step by creating a rising edge. Furthermore, a calibration routine for the docking arm is implemented. The frequency at which the thread is running depends on the period between two consecutive steps. This period is controlled by the *Docking Thread* which implements the docking procedure. It uses the *Arm Controller* class, which determines the required steps, velocity and acceleration of the docking arm based on the processed camera data. It also calculates the angluar velocity of the Mockup to determine to optimal docking time. The *Docking Thread* is running at a frequency of 2 Hz.

8.1.5 Raspberry Pi

The software running on the Raspberry Pi is running in two separate process'. This is due to the previously mentioned issues ORPE had when running inside of RODOS.

- *Datalink*:
  The Datalink section is primarily used for the communication between all three systems (Ground station, Raspberry Pi and STM32). It transfers data between ORPE and the STM32 and also data between ground station and STM32. This communication uses the RODOS gateways for everything except between itself and ORPE, as ORPE does not support the RODOS topic system.

- *ORPE*:
  ORPE runs in its own process inside the Raspberry Pi and is started immediately after boot and communicates with the Gateway datalink via UDP.

8.1.6 Electrical Monitoring

This thread abstracts the connection with the electrical section away. It also has its own logic for starting up and shutting STAR-Sat down when commanded or the battery becoming critically low. The Electrical monitoring also uses the beeper to notify when the system is booting up and also gives a warning when the battery voltage is becoming low. The electrical monitoring runs inside its own thread to make sure it will always run and can protect STAR-Sat.

8.2 Coordinate frames

For the docking mission various coordinate frames are used. In Figure 16 the frames of the mockup are shown and in Figure 17 the coordinate frames of STAR-Sat are shown. The Mockup up frame (M frame in Figure 16) is centered in the middle of the LEDs and fixed to the Mockup, it is used as a reference frame for the camera. The docking port frame (P frame in Figure 16) is not fixed to the docking port. Its placed on the line connecting the Mockup center and STAR-Sat center so that, if the docking port is on that line, the docking port is in the center of the frame.
The camera frame (C frame in Figure 17) is used by the camera, the camera estimates the distance vector to the Mockup frame in the camera frame and the orientation is given by the rotation of the camera frame to the Mockup frame. The docking frame (D frame in Figure 17) is fixed at the docking arm in the calibrated position, it rotates with STAR-Sat but does not move if the arm is extended (keeping track of the arm extension is handled in the *Arm Controller* class instead). The IMU frame (IMU frame in Figure

17) is fixed in the center of STAR-Sat. Additionally STAR-Sat's attitude is measured in reference to a north, east, down system.
The transformation from IMU frame to docking frame was determined from the CAD model. The combined transformations from docking frame to camera frame plus Mock-up frame to docking port frame were measured. For this STAR-Sat, with its arm in the calibrated position, was docked witch the Mockup. Now the docking frame and docking port frame coincide. Thus the distance vector measured by the camera is the negated translation of the aforementioned transformation. And because the docking port frame is just a shifted version of the Mockup frame, the measured rotation is the inverted rotation of the aforementioned transformation. Thus it is now possible, with the measurement of the camera, to determine the full transformation from docking frame to docking port frame or from IMU frame to docking port frame.

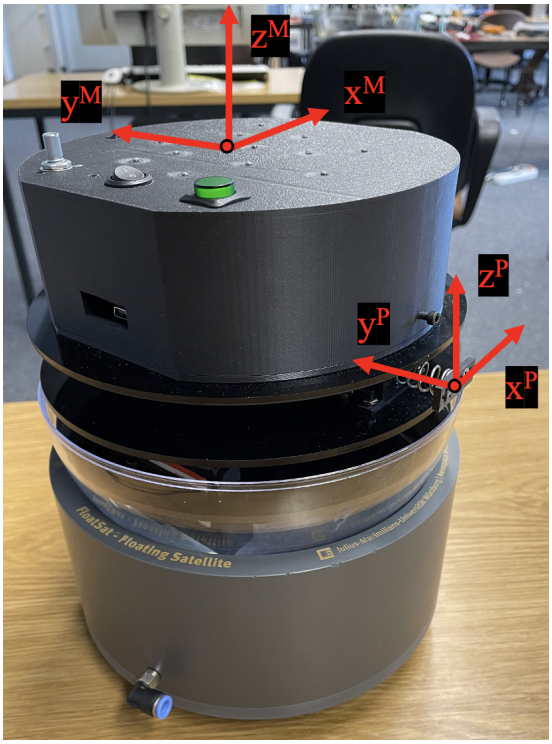

Figure 16: Coordinate frames of Mockup

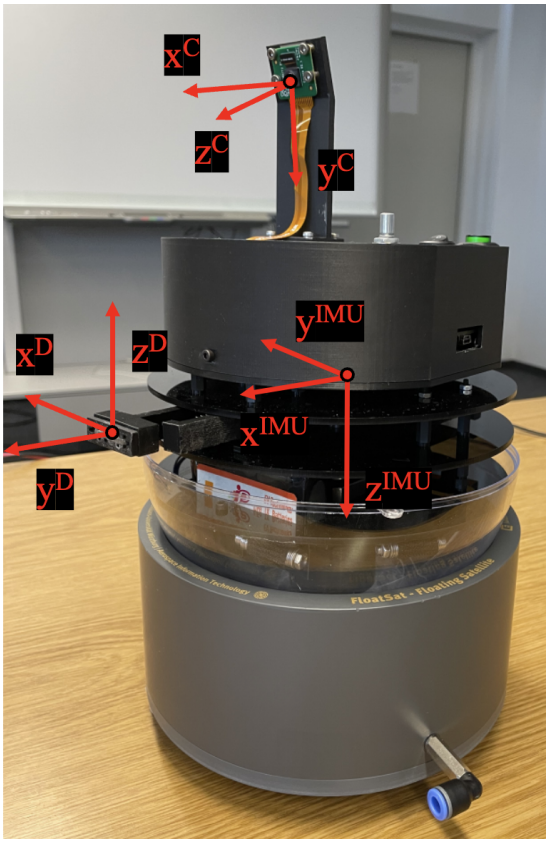

Figure 17: Coordinate frames of STAR-Sat

8.3 Modes

STAR-Sat has various operating modes, they can be grouped into the startup routine, the mission modes and the standard modes. An overview of the modes and the flow of modes can be seen in Figure 18. A deeper explanation of each mode follows.

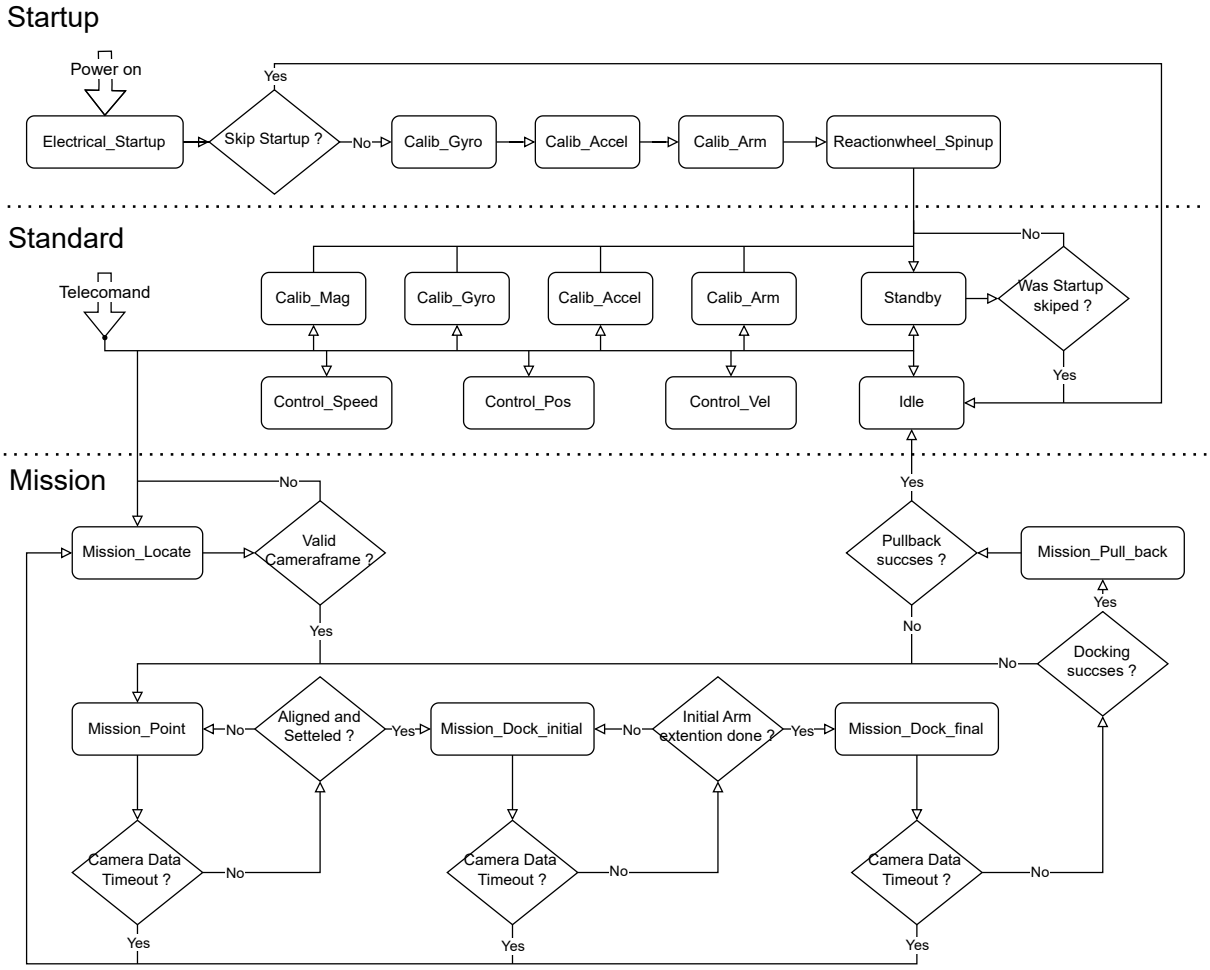

Figure 18: Embedded mode flowchart

8.3.1 Startup

After being switched on, STAR-Sat will be in *Electrical_Startup* mode and, if enabled, will go through a startup routine afterwards. The routine is supposed to be executed while the air cushion bowl is not yet supplied with air. The routine consists of the modes (in order): *Calib_Gyro*, *Calib_Accel*, *Calib_Arm*, *Reactionwheel_Spinup* and ending with *Standby*. Skipping the startup will result in the mode *Idle* and is only for debugging and testing purposes and can be enabled/disabled in a config file.

- Electrical_Startup:
  While this mode is active every thread, except Electrical, is waiting for the electrical system to be set up. They do nothing but check the mode five times a second. The electrical thread then changes the mode once it is done.

- Calib_Gyro, Calib_Accel, Calib_Arm:
  The modes *Calib_Gyro*, *Calib_Accel* and *Calib_Arm* call the gyro-, accelrometer-

and arm-calibration routine respectively. They are meant to be run while STAR-Sat is not yet levitating on the air cushion, so they don't have any controllers active while calibrating. Once the calibration is done the mode gets changed to the next in the startup routine.

- Reactionwheel_Spinup:
  In this mode the reaction wheel is brought to its predefined base speed. This mode is also meant the be run while not yet levitating on the air cushion, this allows the reaction wheel to get to its base speed without rotating STAR-Sat. Once the reaction wheel speed controller has settled the mode gets changed to *Standby*.

8.3.2 Standard

All the Standard modes can be accessed at any time through telecommands.

- Idle, Standby:
  The *Standby* and *Idle* modes serve as the default modes for STAR-Sat, most modes will return to these once they have completed. In Standby mode the Velocity controller is keeping STAR-Sat at standstill. In Idle mode no controller is active and the voltage to the reaction wheel is set to $0V$. If the startup was skipped, the *Standby* mode will transition into the *Idle* mode. This is to avoid operating the reaction wheel around $0RPM$, where the encoder cant measure the speed properly.

- Calib_Gyro, Calib_Mag, Calib_Accel, Calib_Arm:
  The modes *Calib_Gyro*, *Calib_Accel* and *Calib_Arm* call the gyro-, accelerometer- and arm-calibration routine respectively. They should not be accessed during normal operation, since during calibration there is no controller active. They are accessible only for debug and testing purposes. The mode *Calib_Mag* calls the magnetometer calibration routine and the velocity controller spins STAR-Sat at $\frac{\pi}{16}\frac{rad}{s}$. After completion of the calibration all modes go back to Standby mode.

- Control_Speed, Control_Vel, Control_Pos:
  These modes hand the control to their respective Controllers. Control_Speed refers to the reaction wheel speed control, Control_Vel to the angular velocity control of STAR-Sat and Control_Pos to the angular position control of STAR-Sat. The set point of the controllers is not being changed, therefore the last set point or default set point is being used.

8.3.3 Mission

The mission can be started through telecommand, by changing the mode to Mission_Locate, all other mission modes cant directly be accessed by telecommand. After successfully completing the mission STAR-Sat will return to Idle mode.

- Mission_Locate:
  In this mode STAR-Sat is being controlled by the velocity controller. At first the velocity is set to $\frac{\pi}{6}\frac{rad}{s}$. But as soon as the camera picks up at least one light point the speed get reduced to $\frac{\pi}{32}\frac{rad}{s}$ and if the camera identifies at least one LED the speed get reduced even more to $\frac{\pi}{64}\frac{rad}{s}$. This is done to give the camera more time to identify the LEDs once the camera suspects the Mockup is in the FOV. We do not want to stop rotating, as sometimes other light sources can be seen or not all LEDs might be visible. It is also checked that the arm is in the calibrated position and calibrates if its not the case, additionally any previous measurements of the Mockup, such as distance and angular velocity get reset. This is done because this mode might get accessed if steps further ahead in the mission fail. Once the camera has identified the Mockup the Mode gets changed to *Mission_Point*.

- Mission_Point:
  In this mode STAR-Sat is being controlled by the position controller. As a set point for the controller the current position plus the angle to the Mockup, as measured by the camera, is chosen, this effectively treats the camera measurement as the error for the control, since the current position is being subtracted again as part of the error calculation inside the controller. To compensate the camera sometimes loosing the Mockup due to the movement or sending an inaccurate initial estimation a timeout is implemented. This timeout will change the mode back to *Mission_Locate* if there was no valid estimation from the camera for more than two seconds. Otherwise the mode get changed to *Mission_Dock_initial* if the position controller has settled ($|error_{yaw}| < 0.1rad$ and $|w_z| < 0.2\frac{rad}{s}$).

- Mission_Dock_initial:
  In this mode the position control is still in control in the same way as in *Mission_Point*. The same timeout as before is also still being checked. Additionally the distance to the docking port and the angular velocity of the Mockup are constantly measured. This measurements are saved in a ringbuffer of size five each. By median filtering single values to be used by the arm control are being calculated. Before starting to extend the arm it is awaited until five measurements of distance to the docking port and angular velocity of the Mockup are made (the

respective buffers are both full). Then the arm is extended to the docking port distance minus $10mm$. It has to be noted that the distance for the arm to be extended is not being updated once the arm is moving, despite the new measurements. Once the Arm has extended the mode is changed to *Mission_Dock_final*.

- Mission_Dock_final:
  In this mode the control, camera timeout are the same as in *Mission_Dock_initial*. And the constant update of distance to the docking port and angular velocity of the Mockup continues. The main objective of this mode is to calculate the optimal time to dock. For this the time for the Mockup to arrive in the right orientation Δt_{mockup} and the time for the arm to extend Δt_{arm} are being calculated with the latest measurements. Then, if $\Delta t = \Delta t_{mockup} - \Delta t_{arm}$ is smaller than one second, the thread is suspended until $t+\Delta t$. Once resumed the arm is extended to the distance of the docking port plus $3mm$. After the arm extension is done it is checked if the docking was successful by firstly waiting three seconds and then checking if the orientation of the Mockup differs more than 15 degrees from the expected docked orientation. If the Mockup is still within these margins it is assumed to be docked and the mode is changed to *Mission_Pull_back*. Otherwise the arm is calibrated, Mockup measurements reset and the mode is set to *Mission_Point*.

- Mission_Pull_back:
  In the final mission mode no controller is active, because of the rough nature of the pullback. The arm is retracted to the initial position. Afterwards a similar check as at the end of Mission_Dock_final is performed, but this time by only waiting one second. This check is to make sure the Mockup has not disconnected while being pulled back. If successful the mode is set to *Idle* and the mission is complete. If failed the arm is calibrated, Mockup measurements reset and the mode is set to *Mission_Point*.

8.4 Mockup

The Mockup uses mostly the same code and structure as STAR-Sat. The docking and communication thread are not needed, since there is no active docking mechanism and there is no outwards communication at all. The attitude control and determination remains mostly the same. It uses a similar startup routine to STAR-Sat, simply skipping the arm calibration, and remains in velocity control mode afterwards. Two threads are added, one controlling the flashing of the LEDs and another converting the potentiometer input into the set point for the velocity control, allowing the change of rotation speed during operations.

9 Results

STAR-Sat is capable of receiving telecommands and sending telemetry back to the ground-station. We can also command STAR-Sat to go into different modes. E.g. velocity control mode where it rotates at a constant velocity, position control mode where it points in a given angle in reference to magnetic north, calibrate the magnetometer and many more modes.
STAR-Sat is also fully capable of completing its mission of docking with the mock-up. This was fully tested at different positions, distances and also different rotation velocities of the mock-up. Docking success rate is roughly 90 percent reliable, but this mostly depends on how much both STAR-Sat and the mock-up are wobbling, as this greatly effects the docking port vertical alignment at final arm extension and how consistent the velocity estimation of the mock-up is.
The limits are a max distance of 98 millimeters between the docking ports. This is a result of the maximum length of the arm extension. Another limit is the max rotation velocity of the mock-up. This is due to inconsistency of the rotation velocity estimation and the mechanics of docking. When the mock-up is rotating with a velocity of $1.2\frac{rad}{s}$ the docking becomes unreliable and only after multiple automatic retries, is STAR-Sat able to successfully dock with the mock-up.

References

[1] kbarni. Libcamera wrapper library for opencv. https://github.com/kbarni/LCCV, 2023.

[2] Sergio Montenegro and Frank Dannemann. Rodos - real time kernel design for dependability. *DASIA 2009 - Data Systems in Aerospace*, 669:66, 2009.

[3] N. Nethercote and J. Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. https://doi.org/10.1145/1250734.1250746, 2007.