University of Tartu

Faculty of Science and Technology

Institute of Technology

Laur Edvard Lindmaa

Interface development for ESTCube-2 cameras and plasma brake

Bachelor's thesis (12 ECTs)
Computer Engineering

Supervisors: Indrek Sünter, Ph.D

Kristo Allaje, M.Sc.

Tartu 2023

Abstract/Resümee

Interface development for ESTCube-2 cameras and plasma brake Conducting experiments and obtaining data is a crucial component of scientific space missions. To accomplish this, establishing reliable communication with all payloads is necessary. Nevertheless, at the beginning of the thesis, there was no integration in place to interface with the ESTCube-2 payloads. Furthermore, there was limited documentation available regarding communication with both the Earth Observation cameras and the plasma brake. During the thesis, the author created firmware for communicating with said payloads. Additionally, the author conducted thorough tests and verified that camera firmware updating works through satellite interfaces.

CERCS: T120 Systems engineering, computer technology; T171 Microelectronics; T320 Space technology[1]

Keywords: ESTCube-2, CubeSat, On-board computer, Interface, Camera, Plasma Brake

Liidese arendamine ESTCube-2 kaameratele ja plasma pidurile Eksperimentide läbiviimine ja vastavate andmete hankimine on teaduslike kosmosemissioonide üks olulisim komponent. Selle tegemiseks on vaja omada usaldusväärset sidet kõigi eksperimentidega. Enne lõputöö tegemist puudus ESTCube-2 pardaarvutil võimekus suhelda antud eksperimentidega. Lisaks olid nii maavaatluskaamerate kui ja plasma piduriga suhtlemise dokumentatsioonid puudulikud. Lõputöö käigus lõi autor püsivara nimetatud eksperimentidega suhtlemiseks. Lisaks viis autor nende peal läbi erinevaid katseid ja veendus, et kaamera püsivara uuendamine toimib satelliidi liideste kaudu.

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia; T171 Mikroelektroonika; T320 Kosmosetehnoloogia[1]

Märksõnad: ESTCube-2, Kuupsatelliit, Pardaarvuti, Liides, Kaamera, Plasmapidur

Contents

List of figures

Acronyms

CAN Controller Area Network. Serial communications standard originally developed for the automotive industry. 9, 10

COBS Consistent Overhead Byte Stuffing. An algorithm for encoding data which replaces all zero bytes with non-zero values to delimit packets. 11

CRC Cyclic Redundancy Check. An error-detection algorithm used for detecting accidental changes in digital data. 10, 17, 23

EO Earth Observation. 7, 10, 13, 19, 29

EPS Electrical Power System. A submodule responsible for monitoring and providing power to other subsystems. 19

ESA European Space Agency. 9

ESEO European Student Earth Orbiter. European student satellite mission, made by students from eight European countries, launched December 3rd, 2018. 4, 9–11, 14, 24, 25, 31–34

GUI Graphical User Interface. A form of user interface that allows users to interact with software using visual components. 11, 14

HSCOM High Speed Communication. One of the ESTCube-2 subsystems, which is responsible for sending high volume data to Ground Station. 7

I2C Inter-Integrated Circuit. serial communication bus mostly used by lower-speed peripheral integrated circuits. 24

ICP Internal Communication Protocol. ICP is a communication protocol developed by the ESTCube team for subsystem-to-subsystem communications onboard the ESTCube-2 satellite. 7, 10, 12, 21, 23, 27

OBC On-Board Computer. The main processing unit of the satellite. 4, 13, 19–22, 26, 27, 29

UART Universal Asynchronous Receiver/Transmitter. A device that enables communication through an asynchronous serial communication link. 19

UUID Universal Unique Identifier. A unique number, used to identify objects or data. 10, 12

1 Introduction

Over the past hundred years, a lot of progress has been made in space exploration. Starting with the first liquid-fueled rocket engine in 1926 to today's SpaceX Falcon and NASA's Artemis rockets[2]. Space technology has allowed us to see the Earth differently and find solutions that otherwise would not have been considered. Examples include freeze-dried food and photos of deep space[3]. Space technology, meanwhile, has been notorious for its very high cost. As technology has advanced, the creation of smaller and more efficient computer systems has become possible. This makes it possible to build small satellites cheaper than before. Due to that reason, CubeSats have emerged as a popular solution for testing new technologies[4].

1.1 CubeSats

CubeSats are small satellites that follow a specific standard for their size and mass - the unit. One unit (1U) marks a 10 cm cube with a maximum mass of 2 kg[5]. One can combine multiple units together to make larger satellites. The most popular are the 1U, 2U, 3U and 6U configurations with the 3U version being the most widely used[4]. Because the CubeSat standard allows smaller and simpler satellites to be launched, spacecraft construction has become more accessible. An article published by Villela points out that the number of CubeSats being launched has increased more than tenfold over the past ten years while the probability of "infant mortality" has decreased. The latter term is used for satellites that cease operation early in their mission, if not before deployment[4].

1.2 ESTCube missions

Due to the exponential growth in the number of satellites, questions have arisen about space debris. More satellites means more objects to track and a higher chance of two of them colliding with each other. Most CubeSats have a working lifespan of only a few years while remaining in orbit for 20 or more years[4, 6]. Due to their limited size, it is nearly impossible to use propellant to bring the satellite down from orbit. The ESTCube missions are trying to solve this problem by testing plasma brake technology, which only requires atmospheric plasma and electricity.

The main goal of the ESTCube missions is to test the new e-sail technology that could reduce orbital velocity by increasing the drag in atmospheric plasma[7]. Furthermore, the goal of the ESTCube team is to develop our spacecraft bus further, to eventually support missions outside of LEO, such as a lunar mission or Multi-Asteroid Touring[8]. Finally, one of the educational goals of ESTCube is to train new space engineers by giving them experience in real-life problem-solving and career growth opportunities.

1.2.1 ESTCube-1

ESTCube-1 was the first Estonian satellite launched from the Guiana Space Center on May 7, 2013. The mission successfully demonstrated that all the satellite's custom-made subsystems worked except for one sun sensor, which was not critical to the mission[7]. There were two science payloads on board - an electric solar wind sail and a camera. The solar wind sail tether on board the satellite was 10 m long. Unfortunately, unravelling the thread failed, foiling the experiment. The exact reason for the failure is unknown, but it is hypothesized that the motor jammed because it was either damaged during the launch vibrations or cold welded in space. The camera, also called the tether end mass imaging system, was used to confirm the failure of the e-sail deployment. As a secondary objective, the camera was used to capture pictures of the Earth. In total, about 300 photos were retrieved from the satellite, capturing color images of different parts of the planet[7].

1.2.2 ESTCube-2

ESTCube-2, a three unit CubeSat, is Estonia's most recent CubeSat for testing new space technologies. It contains more experimental payloads than ESTCube-1, with the primary payload also being the plasma brake. As the e-sail is extruded from the satellite using centrifugal force, cold-gas thrusters were included on board to offload momentum from the reaction wheels[9]. Other payloads on board the satellite are two Earth Observation (EO) cameras and a corrosion experiment to test different coatings that protect against corrosion. Exploded view of the satellite illustrating the payloads is shown in Figure **2**. At the time of writing this thesis, the planned launch window of the satellite is from late summer to early autumn 2023.

The satellite will collect and send its internal housekeeping data every few minutes to report the operational status of all sub-modules[10]. The satellite's subsystems communicate over a redundant set of RS485 interfaces using the proprietary Internal Communication Protocol (ICP)[11]. The spacecraft's payload modules are housed on a separate bus connected to the onboard computer and High-Speed Communication (HSCOM). In addition, each payload has its own protocol, made separately by the payload module teams and thus they are unrelated to the ICP. The connection scheme between mission control and payload bus can be found in Figure **1**.

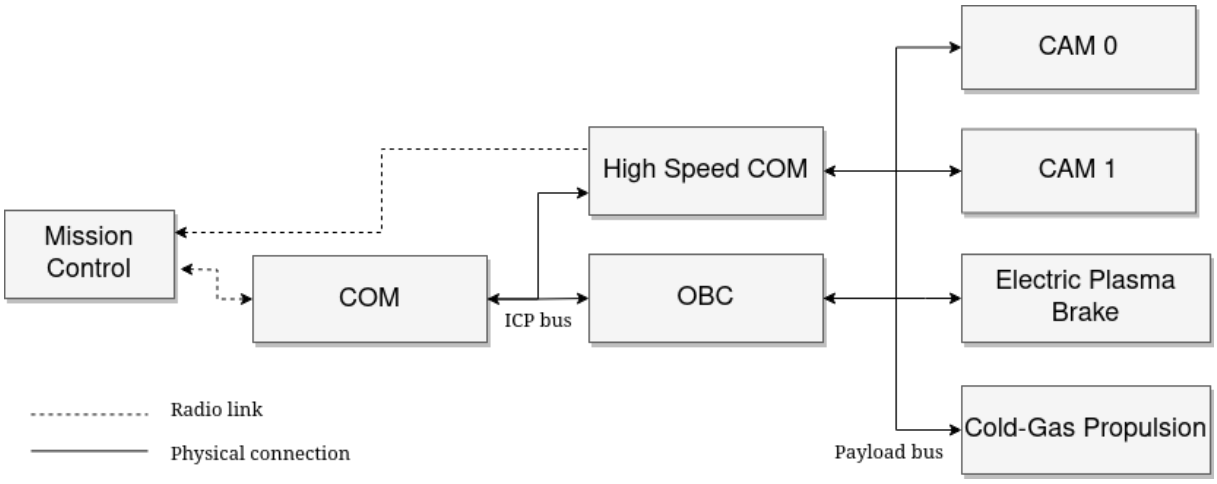

Figure 1: ESTCube-2 mission control to payload bus connection diagram.

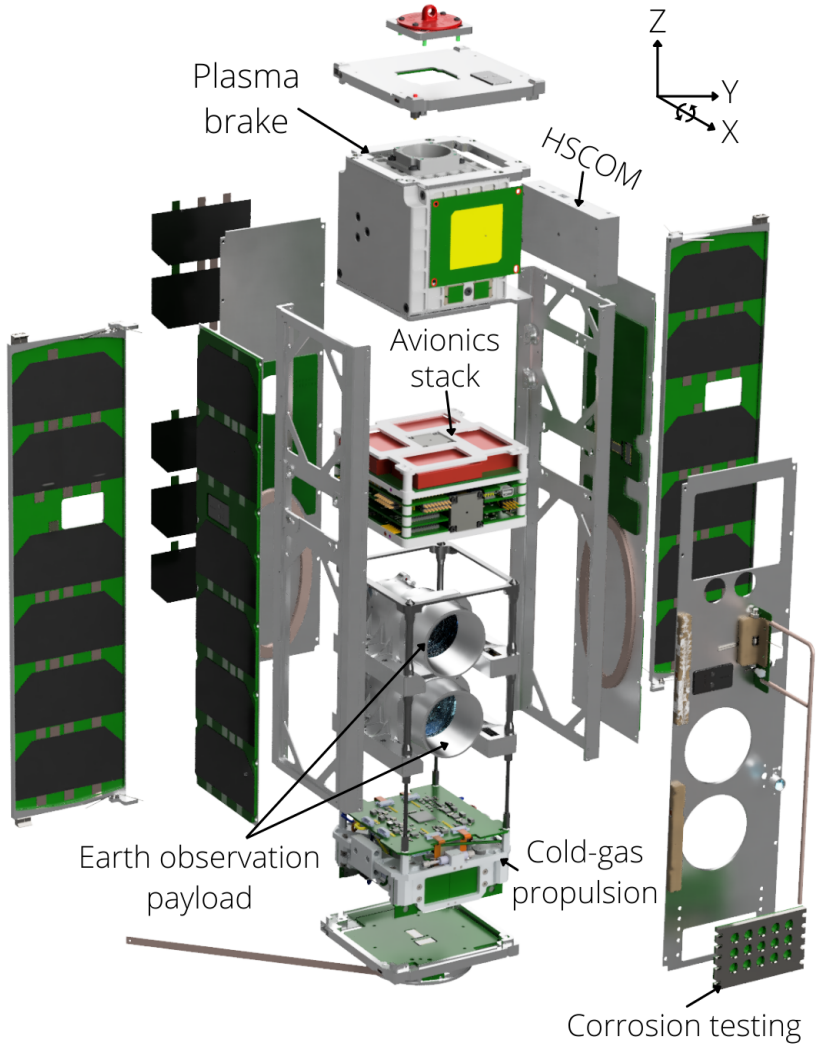

Figure 2: Exploded view of ESTCube-2 payloads, (I. Iakubivskyi)

1.3 ESEO

ESEO was an educational mission developed by the European Space Agency (ESA) in collaboration with European universities and research institutions. The mission aimed to provide a platform for European students to gain hands-on experience in designing, building and operating a satellite. The satellite was launched from Vandenberg Air Force Base in California on December 3, 2018[12].

Originally ESEO was meant to fly with a uCAM payload by DTU Space. However, in 2014 Tartu Observatory was contacted for a replacement camera for the spacecraft [13]. Two cameras were designed to fit into the existing spacecraft bus, called primary and secondary [14]. Both cameras are presented in Figure **3**. The primary camera was connected to the satellite's internal communication bus, the Controller Area Network (CAN) bus, while the secondary was connected to the primary camera through a different interface. Both cameras were designed at Tartu Observatory[15].

ESEO secondary camera design was further developed to be used on board ESTCube-2, with one of the changes being swapping out the CAN bus interface to fit ESTCube-2's RS485 payload bus.

Figure 3: ESEO primary (left) and secondary (right) cameras.

2 Previous work

2.1 ESTCube-2 internal communication protocol

ICP is a proprietary communication protocol developed in-house and used for ESTCube-2's internal communication within the avionics stack and to the side panels. The original ICP packet structure was created during Sander Tammesoo's bachelor's thesis. Since then, it has received significant updates from different volunteers over the years. The packet, shown in Figure **4**, consists of a header, a data portion, and a 16-bit Cyclic Redundancy Check (CRC). The header consists of a destination address, a source address, a command ID, a Universal Unique Identifier (UUID), and a command mode byte[11]. Along with the data connections, the ICP has one access and a shutup connection. The access signal is active low and marks the line as busy and sends ACK and NACK signals. ACK and NACK value depends on for how long the recipient held the signal low. The shutup line is currently not in use, but it is planned to be used in sending priority packets in the future[10].

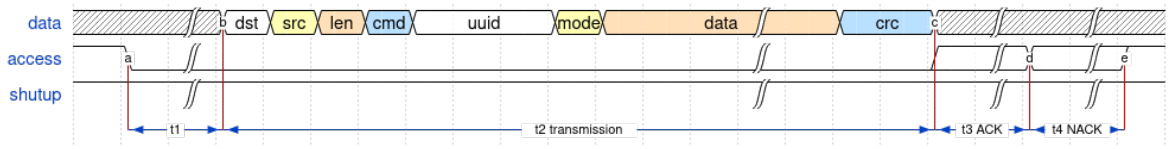

Figure 4: ESTCube-2 ICP packet[10].

2.2 Payload protocols and user interface

2.2.1 Cameras

The EO camera design is based on the European Space Agency's ESEO satellite optical payload that was modified for use in ESTCube-2. The two cameras on board ESEO shared an RS485 connection between them. However, only the primary camera was connected to the rest of the satellite through a CAN bus. In ESTCube-2, both cameras are connected to the RS485 payload bus.

The main reason for changing the communication standard is speed. RS485 provides speeds over 1 Mbit/s while the CAN bus, which was used in ESEO, only achieved 2 KiB/s[16]. A camera image with a size of 2592x1944 pixels means 5,038,848 pixels per image. Each pixel comprises of 12 bits [17], thus the full image size is 7.21 MiB. Sending that amount of data through the 2 KiB/s ESEO CAN bus would take over an hour, while through a 1 Mbit/s RS485 connection, it would take one minute.

2.2.2 CAM protocol

The original inter-camera protocol coordinated commands with the secondary camera and listened to its responses. The protocol structure logic was inspired by ESTCube-1 communications, where multiple commands were chained into a sequence and sent as one RF packet. This ensured that once the RF packet was received on board the spacecraft, all commands in the chain would be performed[14]. Figure **5** shows the original packet structure.

At the start of the packet, there are 2 zero bytes, with one zero byte at the end. The lengths of the delimiters at the beginning and end of the packet are different because in this way they can be determined more reliably on a noisy line. Otherwise, the noise could be considered to be a packet and the actual packets as noise. In the middle of these zeroes is the packet data, encoded using Consistent Overhead Byte Stuffing (COBS). COBS reformats the data to remove all zeroes, and the zero bytes can be used to delimit the data.

The COBS encoded data includes a header that contains a 4-byte long checksum and 4 bytes that mark the total length of the command sequence and the final command sequence. Checksum is calculated using the CRC32 MPEG2 standard with 0x04C11DB7 as the polynomial. The length field is 4 bytes long and shows the complete length of the command sequence. The command sequence consists of one or more commands, each having a one-byte long command ID field and a byte-long field to show the length of arguments, followed by the said arguments.

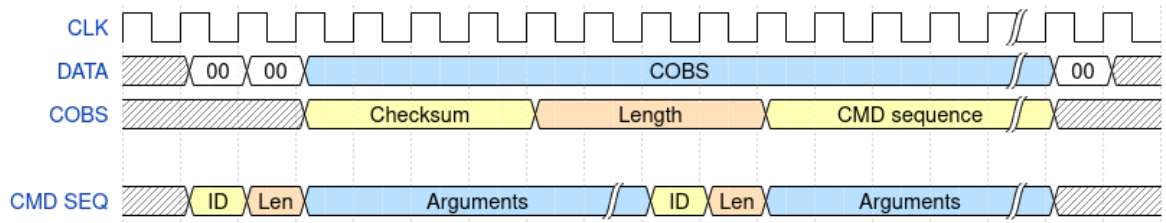

Figure 5: ESEO inter-camera protocol.

2.2.3 CANTerminal

ESEO cameras came with a Graphical User Interface (GUI) called CANTerminal that helped users to control and debug the cameras through a USB interface. CANTerminal was originally made by Indrek Sünter, Henri Kuuste, and Martin Valgur for use in the ESTCube-1 and ESEO missions. The program was written in Python 2 and used PySide for the graphical interface. Since the program was created several years ago, it was outdated and did not start. Python 2 development had been discontinued, and the libraries used had become unavailable or only been updated for Python 3.

The CANTerminal incorporates various higher-layer programs, referred to as extensions, which enhance functionality and enable automation. Instead of manually typing different commands into the terminal, the extensions would do it for the user with just the click of a button. More detailed descriptions of the extensions that were used can be found in section **4.3**.

2.2.4 Plasma brake

The plasma brake is also connected to the ESTCube-2 RS485 payload bus and its communication protocol can be seen in Figure **6**. The packet begins with two sync bytes to mark the start of the packet. Sync bytes are meant to reduce the effect of noise, so the module would ignore random fluctuations on the data lines and not mark them as valid data. Plasma brake protocol uses CRC16 Modbus standard with 0x8005 as the polynomial for checksum calculation [18]. Otherwise, the packet is similar to the ICP packet, except for the lack of UUID and command mode fields.

Figure 6: Plasma brake protocol.

3 Thesis goals

An essential part of any scientific space mission is conducting experiments and receiving the resulting data. For that, it is necessary to communicate with all the payloads reliably. However, interfacing with the ESTCube-2 payloads had not been implemented at the start of the thesis. In addition, there was minimal documentation on communicating with both the EO cameras and the plasma brake.

This Bachelor's thesis aims to develop software to enable communication between the OBC and the aforementioned payloads. To achieve that, the task was divided into smaller sub-goals:

- Update CANTerminal and its extensions to Python 3 to enable their use on modern computers to control the cameras.
- Create an interface for communicating with the plasma brake and cameras and use all of their capabilities.
- Test the plasma brake and camera commands and functionality through the payload bus.

4 Development

4.1 Camera user interface

The camera firmware and CANTerminal were initially located in a private ESEO code repository. A separate Git project was created for them in the ESTCube Bitbucket.

To execute and debug CANTerminal, the camera libraries had to be compiled. The author reformatted the files and directories to match the ESTCube-2 naming system. The exact change also had to be done in the build system so the compiler could find the correct directories.

4.2 CANTerminal development

First, all of the compiled files and build folders were removed from the cloned repository because they are unique to each user and can be created based on source files. After that author started to update CANTerminal to use Python 3. Due to complications with Pyside 2, it was decided to switch to Qt 5 instead for the QT graphical framework. Initially, Qt and Pyside were thought to be very similar because they were created by the same company, The Qt Company, and used the same back-end[19]. While porting to Qt 5, numerous unforeseen problems emerged related to threading and communication between them, and in the process, a lot of code would have had to be rewritten. Because of this, it was decided to revert the Qt 5 changes and update to PySide 2 instead.

When porting PySide and Python to a newer version, the most significant problems were the newer syntax and different data types of Python 3. For example, in Python 2, there was no "bytes" data type. Instead, the "string" data type was used.

After fixing the bugs in the GUI, the author got CANTerminal successfully ported to the new version and working. The graphical interface of the program can be seen in Figure **7**. As the next step, the author studied the existing terminal extensions and compiled a list of modules that needed updating to test the cameras' full functionality.

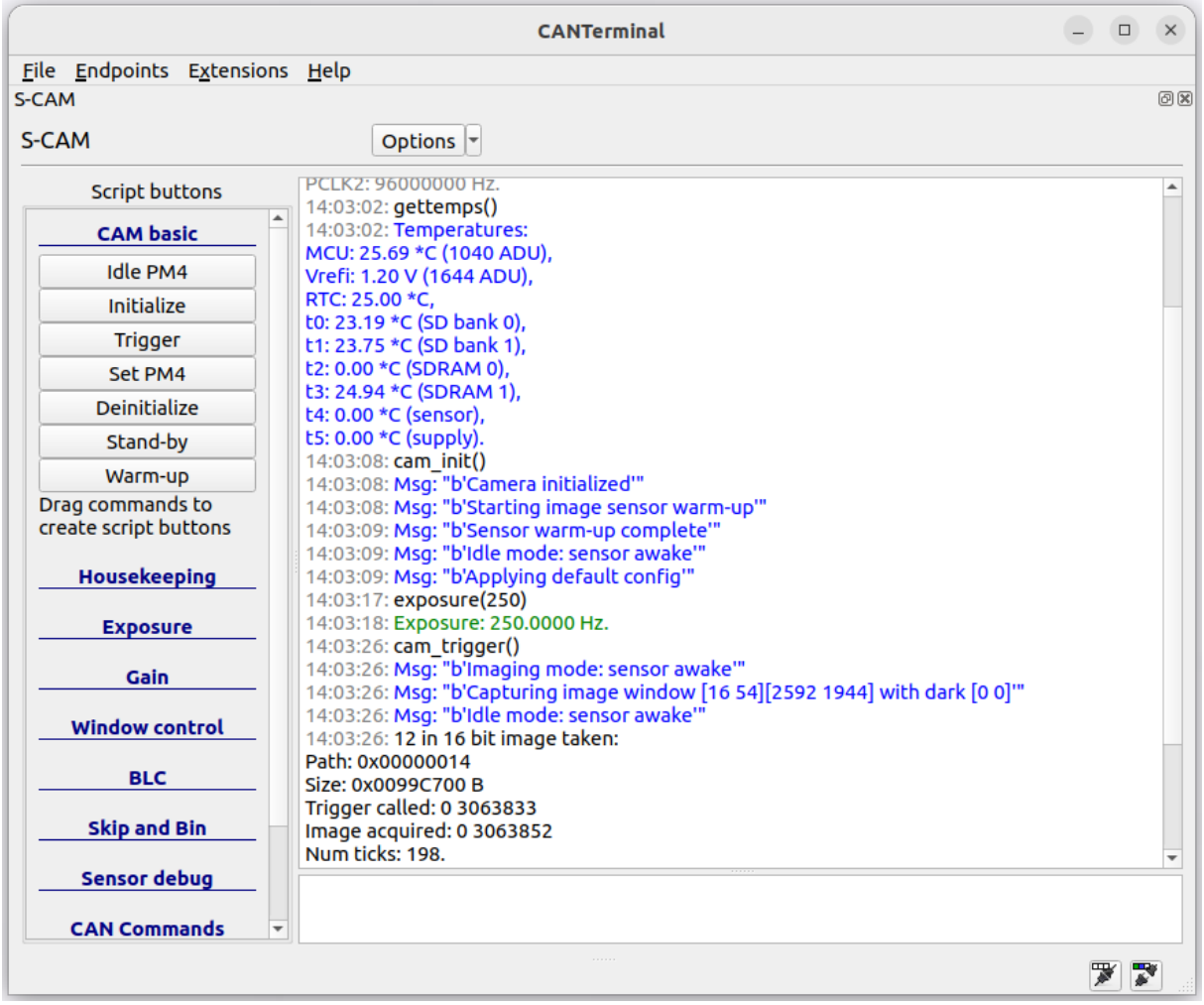

Figure 7: CANTerminal.

4.3 CANTerminal extensions

CANTerminal has several extensions, making it easier to operate and debug the cameras. Since there are a lot of extensions, the amount of work to update them all would be too large. A list was compiled containing extensions that are required to achieve the thesis goals:

- Configuration manager
- Device table
- PyCAM - an extension that makes it easy to take and process pictures
- File transfer
- Firmware updater

4.3.1 Configuration manager and device table

The device table is an extension that lists all the peripherals on the cameras with their states. Through it, the user can manage existing peripherals and see their current status. It also allows the user to see if the selected device reports any error. Using this extension, one could also enable or disable selected peripherals. These features were helpful during the testing of the cameras because they made it possible to see which peripherals were actually working and if any of them as recently reported an error.

In addition to the device table, the configuration manager extension was selected to be updated, through which the settings of each peripheral could be specified in greater detail. For example, if the device table can enable the RS485 line, then in the configuration manager one can change the RS485 communication baud rate and internal buffer size.

4.3.2 PyCAM

The PyCAM extension allows users to take pictures, download and save them to the computer. It uses the file transfer extension in the background to load images from the camera. Another feature of this program is that it can remember the metadata of the pictures already taken along with their location in the camera storage. This is useful if the user takes several pictures in a row and wishes to retrieve them later without going through all the storage devices to find the locations of the pictures.

PyCAM made it possible to preview the image immediately after capturing it. Seeing the pictures shortly after capturing them made debugging a lot easier. The extension also offered the option to view each color channel (red, green, and blue) separately in the image, which allowed the author to see if all channels on the image sensor were working correctly. More about debugging the cameras and pictures taken by them is written later, in section **5.1**.

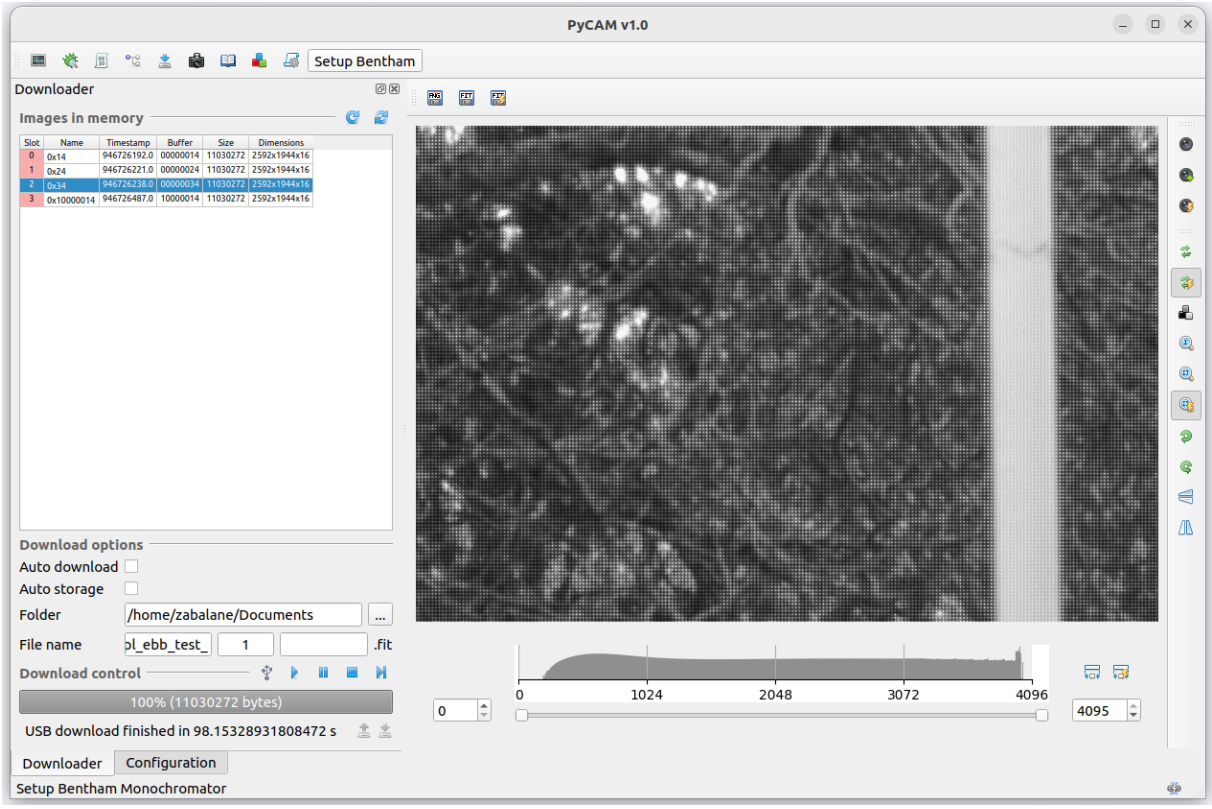

Figure 8: PyCAM extension graphical user interface.

4.3.3 Firmware updater and File transfer

The firmware update extension helps the user to test the camera firmware update functionality through the selected communication line. The user can choose which firmware slot the new software will go into and check it before sending it. When selecting an image, the program shows whether it fits in that slot by checking the length and verifies its version number and CRC. Before sending, the firmware updater divides the image into pages of up to 255 bytes, which can be sent to the camera using the CAM protocol. In addition, the user can choose how frequently firmware pages are sent. The firmware update extension user interface is shown in Figure **9**.

After uploading the new firmware, it is possible to download the map of pages and CRC of the whole image to see if all the data arrived correctly. Faulty pages are retransmitted and rechecked. After all the pages have been sent, the user can change the boot order and start using the new software.

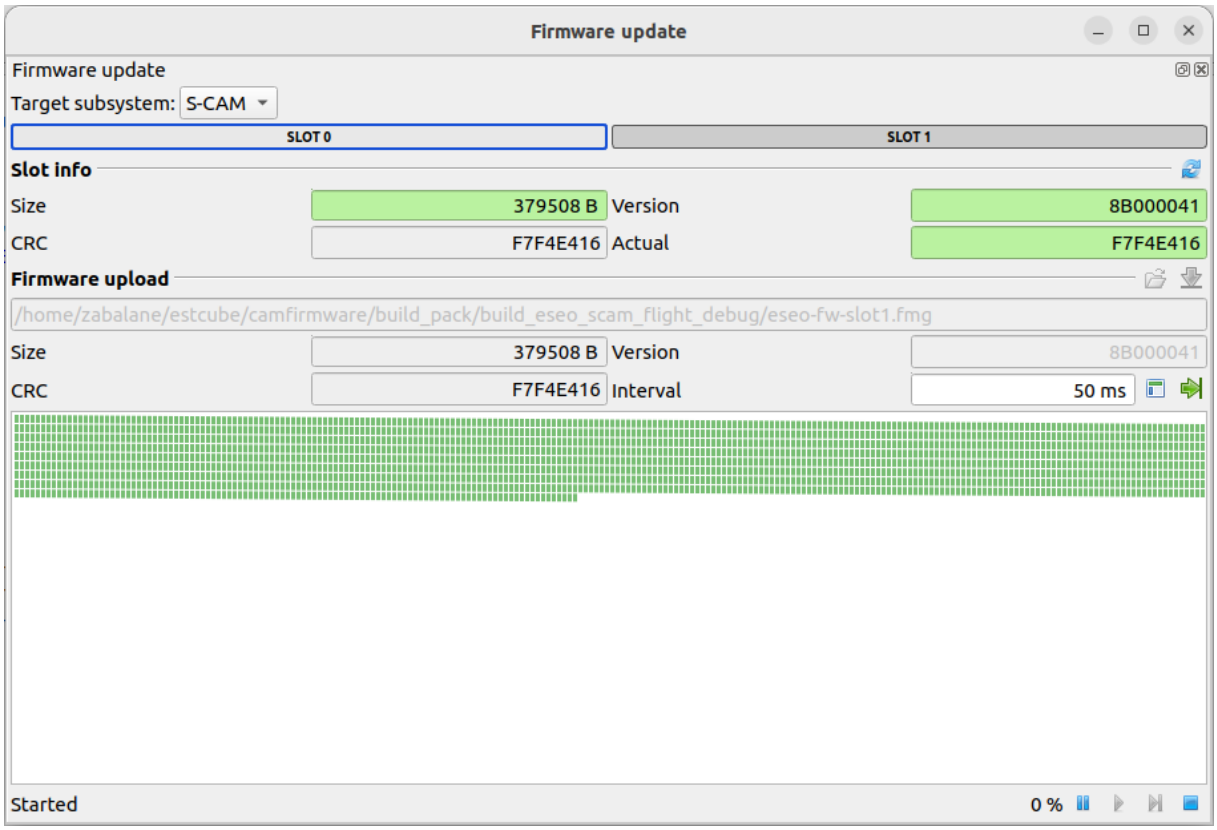

Figure 9: Firmware updater extension user interface.

The file transfer extension allows users to download and upload files from the camera. It also allows the user to load only some parts of the file by specifying the offset from the start and the length of the data. Besides the possibility of loading files, the user can bookmark the locations and sizes of the files to update them by only referring to the bookmark. File transfer is mainly used by other extensions, not directly by the user. Of the restored extensions, the file transfer extension is used by firmware update to send firmware pages and by PyCAM to download pictures.

4.4 Payload connections

The payload bus uses the RS485 standard for communication and it has the following signals:

- VIN (power input) pins connected to the main power bus
- Ground connections
- RS485 A and B connections
- Interrupt to and from the payload

Figure **10** shows the payload bus connector on the camera board; the plasma brake has the same pinout but a different connector. The power input pins are directly connected to

the battery through a power switch on the Electrical Power System (EPS). The interrupt pins are used to coordinate actions between the payload and OBC. For example, the interrupt can be used to make a payload listen again after sending a deafen command. While pins were reserved for interrupts from payloads, in the final configuration they don't generate any.

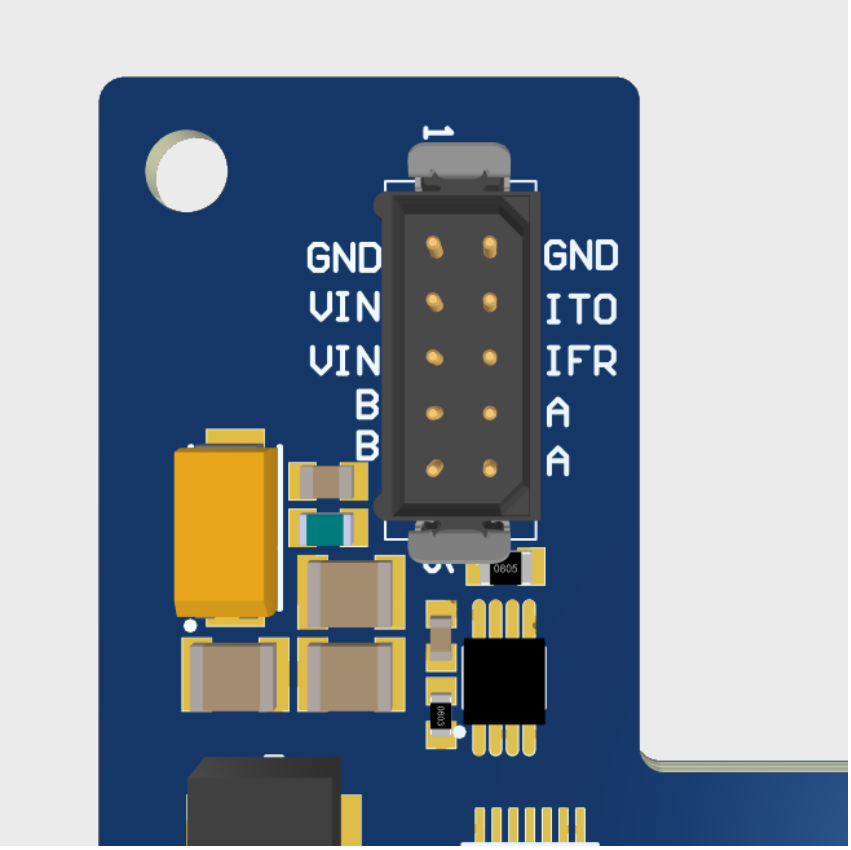

Figure 10: Camera payload bus connector.

4.5 Helper board

As part of the thesis goal to test the EO cameras, a tool was created for testing the cameras through the payload bus. CANTerminal used a USB connection to control the cameras; however, testing the RS485 interface in the same way was impossible. The author created a USB and Universal Asynchronous Receiver/Transmitter (UART) to RS485 converter for that use case. The logic diagram of the board is shown in Fig. **11**. The pictures and layers of the helper printed circuit board can be found in appendix B.

It allowed the user to communicate with the cameras through the RS485 interface, simulating a typical payload bus transmission. In addition, the board can connect another device, for example, a development board, which can send messages using the UART interface. To choose which input to use, interface selection jumpers were added before the RS485 converter. To test the camera interrupts, separate pin headers were added to the

board. During the thesis, interrupts were implemented in the camera firmware to make the cameras listen again after the OBC has sent the deafen command.

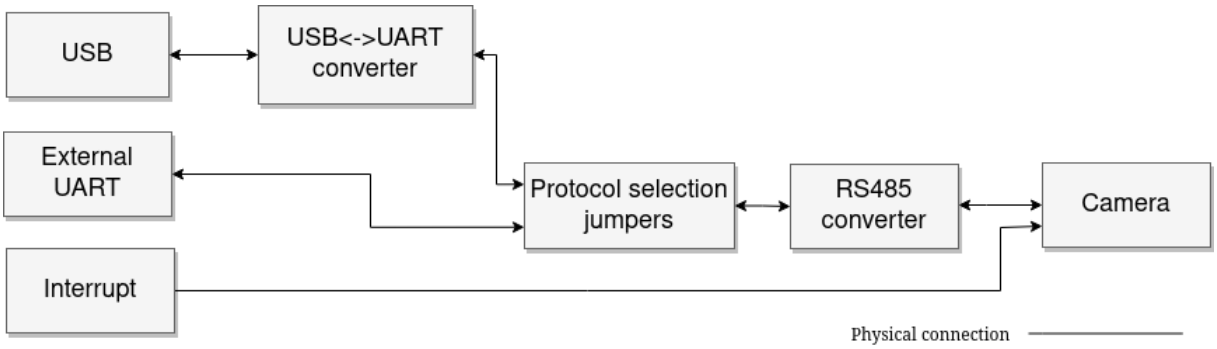

Figure 11: Helper board communication diagram.

4.6 Changes to the CAM protocol

The disadvantage of the original camera protocol was that only two nodes could be on the bus simultaneously. This was because there was no destination nor source defined in the packet. With more nodes, it would have been impossible to determine exactly where the packet originated from and where it should go. In addition, the cameras were located on the payload bus containing other experiments that use different protocols, such as the plasma brake. Because of this, it was necessary to create a way to deafen the camera and make it start listening again. Otherwise, it would read invalid data when listening to the wrong protocol whose structure and baud rate are likely different. The updated CAM protocol is shown in Fig **12**. The author changed the camera protocol by adding a camera ID field and created a deafen command for the cameras. After deafening the cameras can be made to listen again by sending an interrupt through the payload interface. It was decided not to add the source field as the payload bus works in a master-slave configuration. That means the OBC expects a response only from the payload it currently communicates with.

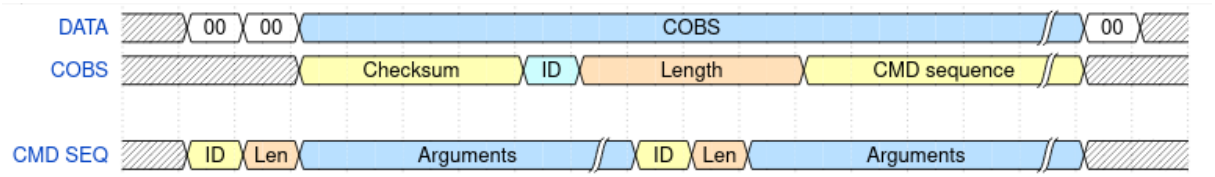

Figure 12: Camera protocol.

4.7 Payload interface

At the start of doing the thesis, a minimal interface on the OBC existed, which could only ask for housekeeping data from the Plasma brake module. The interface was hard-coded, and it couldn't execute any other commands. Due to its lack of features, only the OBC could execute the housekeeping command making it impossible to control the payload module through mission control. The housekeeping command was crucial during the initial testing because it was used to determine if the communication with the module works and how it works internally.

To fully control the payloads, many more commands are required. Cameras must receive at least the commands to warm up the sensor to configure the exposure and gain before taking a picture. They must have the ability to receive and process firmware update commands as well. In addition, it is necessary to change and calibrate other camera values through the payload interface, for example, by configuring communication timeouts. Similarly, the plasma brake must receive the commands to control the stepper motor and a high-voltage amplifier to be fully functional. To use all the features of the cameras and the plasma brake, the author completely remade the existing payload interface.

The payload interface can now be called through the ICP bus by setting the destination MAC address to that of the payload. It assumes that the ICP logic is working correctly and is given a pre-verified ICP packet. The OBC retrieves all packets intended for payloads and transfers them to the payload handler for subsequent processing. The payload handler checks where the packet should actually go in the end and restructures it accordingly. The translated packet is then sent to the payload bus. If the previous message on the payload bus used another protocol, then before sending a new message, the deafen command is sent to previous listeners and the listeners of the current protocol are woken up. If the payload packet contains more than one command, the payload handler cuts it into multiple packets because the ICP protocol allows sending only one command at a time. The logic diagram of the payload handler is shown in Figure **13**. In the same way, the cameras and the plasma brake are attached to the payload handler, other payloads could also be connected. Adding support for other payloads is beyond the scope of the thesis.

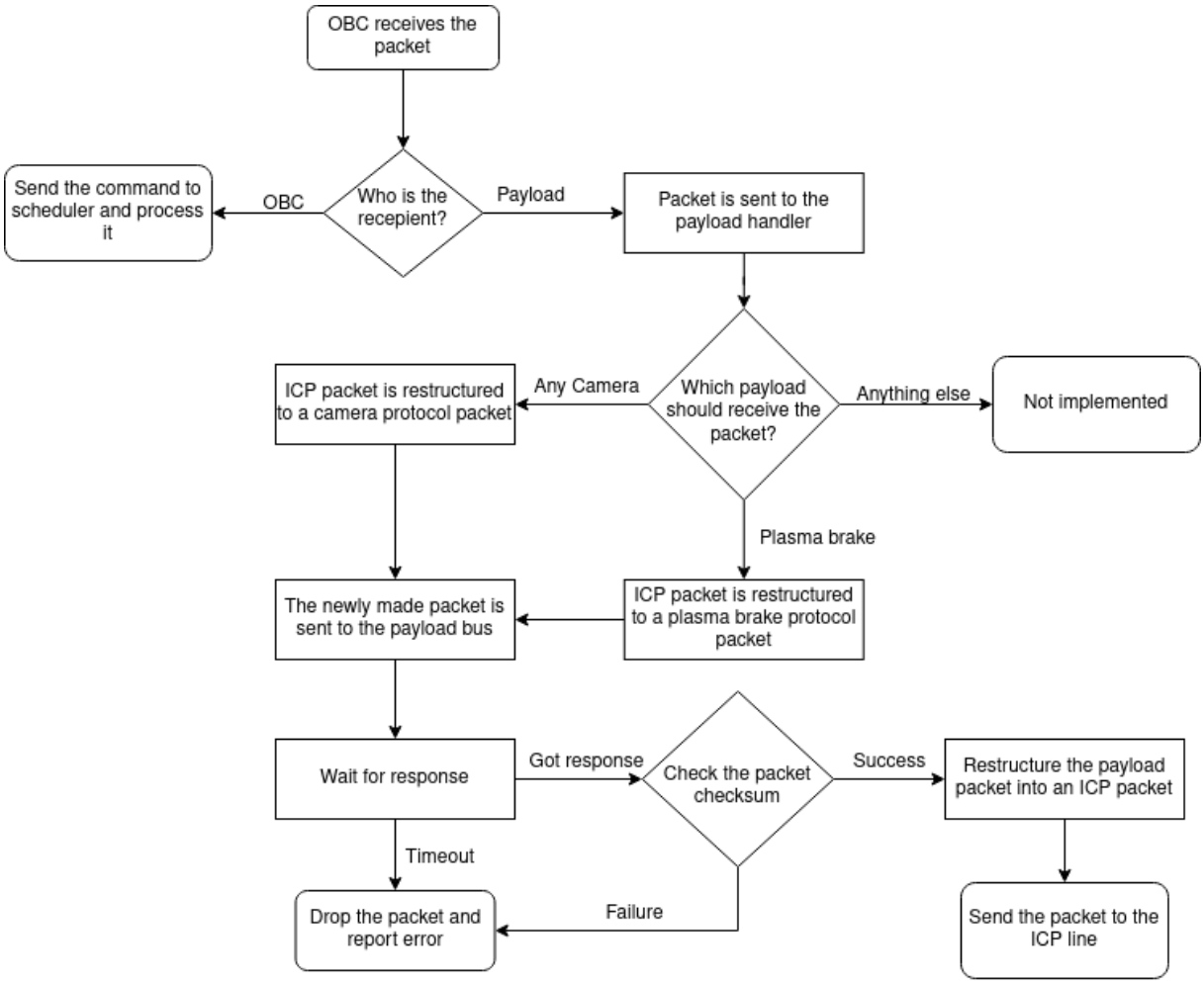

Figure 13: Payload interface logic.

Besides human errors, the radioactive environment in space could corrupt bits in the data in a packet [20]. To determine if the packet has the correct structure and contains valid data, the following checks are performed on the payload interface, and if any of them fails, an error is logged, and the packet is dropped:

- The packet is placed in the payload handler's task queue. It is dropped if too many packets are in the queue and it cannot be placed there during the timeout period.
- Determine which payload the packet is aimed at.
- Check that the packet has at least the minimum length to separate the header and data.
- The CRC of the packet is checked to verify the packet's integrity.

4.8 Plasma brake integration

After creating an interface for the camera, the author fulfilled the second thesis goal by adding the plasma brake to the ESTCube-2 OBC payload interface. Since the payload

handler was made to be modular, there were no significant problems with integrating the plasma brake.

The addition of a new payload was simple; the author had to add the plasma brake to the ID look-up table of the payload handler along with the packet restructuring logic to plasma brake protocol and back to ICP again. Finally, the corresponding CRC calculations and the previously mentioned package structure checks were added.

5 Testing

5.1 Camera testing

The testing process commenced with the ESEO cameras as the ESTCube-2 cameras were not assembled. The first captured image by the author using the ESEO primary camera can be found in appendix A Figure **18**. However, it should be noted that the images are flawed due to the incomplete finalization of the logic for image reconstruction at that particular moment. Subsequently, after successfully generating all required commands and translating ESEO camera responses through the CANTerminal, the author tested the same commands on the ESTCube-2 cameras.

5.1.1 Debugging sensors

While testing ESTCube-2 cameras, it was observed that initializing the sensor and taking pictures did not work. There were two types of errors; the first, more frequently encountered error was that when the sensor was initialized, it returned an Inter-Integrated Circuit (I2C) bus error. The second error was that the sensor took a picture, but the downloaded picture was faulty. Since the sensor connector was cross-compatible with the ESEO cameras, it was decided to test the sensors with the ESEO secondary camera to find out whether the error was in the sensor or the camera's motherboard.

The sensors that returned the I2C error did not change their behaviour, but the images taken with the working sensors were not distorted. One of the distorted pictures can be found in appendix A Figure **17**. From this, it was concluded that the I2C error was due to a faulty sensor and that the ESTCube-2 camera motherboard caused the faulty image capture. After a discussion with Indrek Sünter, the author learned that ESEO had a similar problem with sensors. Only a few of the ordered sensors worked correctly during making cameras for that mission. After learning this, a systematic approach was taken to mark and filter all available sensors. In the end, only three image sensors were fully functional because some exhibited issues during the initialization of Phase Locked Loop to produce a high enough clock rate to enable image capture. Two were used on the flight model, and one remained in the engineering model to enable further software development.

5.1.2 Debugging memory

After resolving the issue with sensor I2C initialization, it was possible to focus on the image capture. First, all the connections in the electrical schematic diagram between the camera sensor and the processor were checked. From there, it was confirmed that the circuit was equivalent to that of the ESEO camera.

The author monitored the communication between the sensor board and the mainboard using a Saleae Logic 8 analyzer[21]. ESTCube-2 camera motherboard, with the sensor and

logic analyzer attached, is shown in Figure **14**. The author found that the correct signals are sent to the sensor, which returns the correct data. Meaning that the error must be in the processing of the picture. But since ESEO and ESTCube-2 cameras use the same software for taking and saving pictures, the schematics and printed circuit boards were compared in an attempt to spot differences.

Finally, the author found that different memory devices had been soldered on the ESTCube-2 camera, which were pin-compatible with those of the ESEO's. Namely, instead of IS42S16320D-7TLI[23] with 64 MiB of capacity, IS42S16160G-7TLI[22] with 32 MiB of capacity were soldered. After looking at the said integrated circuit's datasheets, the author found out that the error is very likely in the timings of the memories. Due to different signal dwell times, the data was not stored in the memory, leaving it with incorrect or missing data.

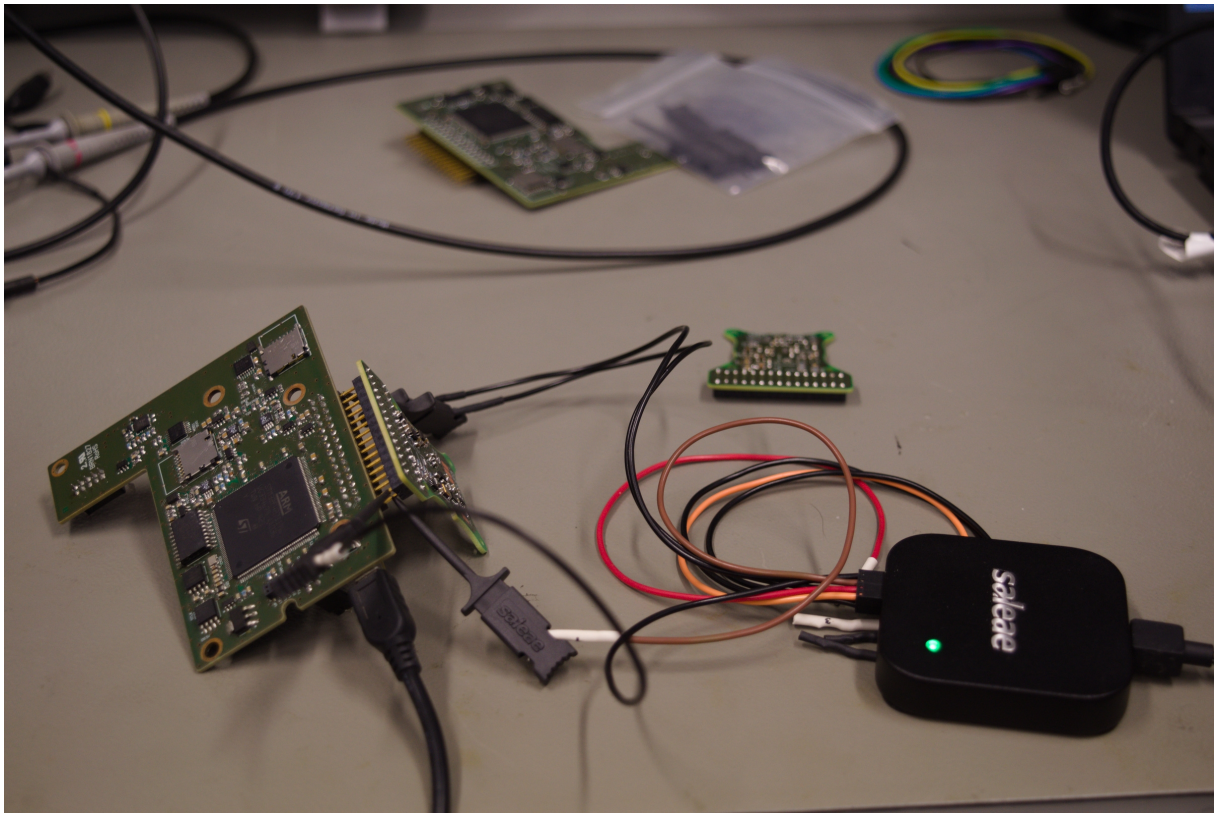

Figure 14: Debugging the ESTCube-2 camera sensor communication, (I. Sünter)

Following the soldering of the memories, a successful image from the ESTCube-2 camera was obtained. By doing this, it was possible to start testing all of the commands related to the sensor. Before taking further pictures, the author assembled the camera and had another team member focus the lens. Pictures taken with the cameras can be found in appendix B.

After eliminating these problems, it was possible to test the RS485 interface of the cameras using the helper board. Through it, the author compiled a list of experiments to check if all critical functionality works nominally:

- Communication at different RS485 baud rates - to figure out at which baud rate the communication becomes unstable and how much
- Sending as many commands as possible in a short amount of time to test how the camera would handle such a case
- Firmware update - Remote firmware updates, enabling improvements after the launch
- Image capture and download

The author tried communicating with the cameras at the following baud rates: 9600, 19200, 28800, 38400, 57600, 76800, 115 200, 230 400, 460 800, 576 000, and 921 600. Higher baud rates were not tested because according to the ESTCube-2 system engineer, baud rates above 921 600 baud rates are unlikely to be used for communication with the cameras. At baud rates below 19200, packets were sent at 500 ms intervals; at baud rates below 230 400, packets were sent at 50ms intervals. With higher baud rates, packets were sent at 10ms intervals. When commands were sent more frequently than the delays mentioned, the camera got out of sync and dropped commands. Sending the reboot command to the camera until it started responding was sufficient to escape this situation. Sending commands and updating firmware was successful for all of the aforementioned baud rates. The author used the firmware update extension for firmware updates, which showed exactly which firmware pages were successfully uploaded and which failed.

The camera testing setup through the RS485 interface can be seen in Figure **15**. In addition, after doing all these tests with the helper board, the author tested the communication by sending commands from the satellite communication subsystem through OBC, which proved successful.

Figure 15: Testing the camera with the helper board. Both camera and the RS485 helper board are connected to the laptop.

5.2 Plasma Brake testing

The plasma brake commands were tested and all the responses it sent were validated. A fixed baud rate of 115 200 baud was used for all the tests as the payload was designed to work only at that baud rate[18]. Plasma brake commands were sent from the OBC using the ICP loopback system, where the subsystem can send ICP packets to itself and process them as if they originated from some other source. Compared to the cameras testing the plasma brake was relatively simple as the payload worked nominally. Figure **16** displays the plasma brake payload testing setup. The plasma brake is attached to the power supply which provides it with the same voltage as is expected from the satellite's main power bus, 7.4 - 8.4 V, because unlike the cameras it does not take 5 V as power input. In addition, the plasma brake on the Figure was attached to the OBC which generated and sent commands to the payload.

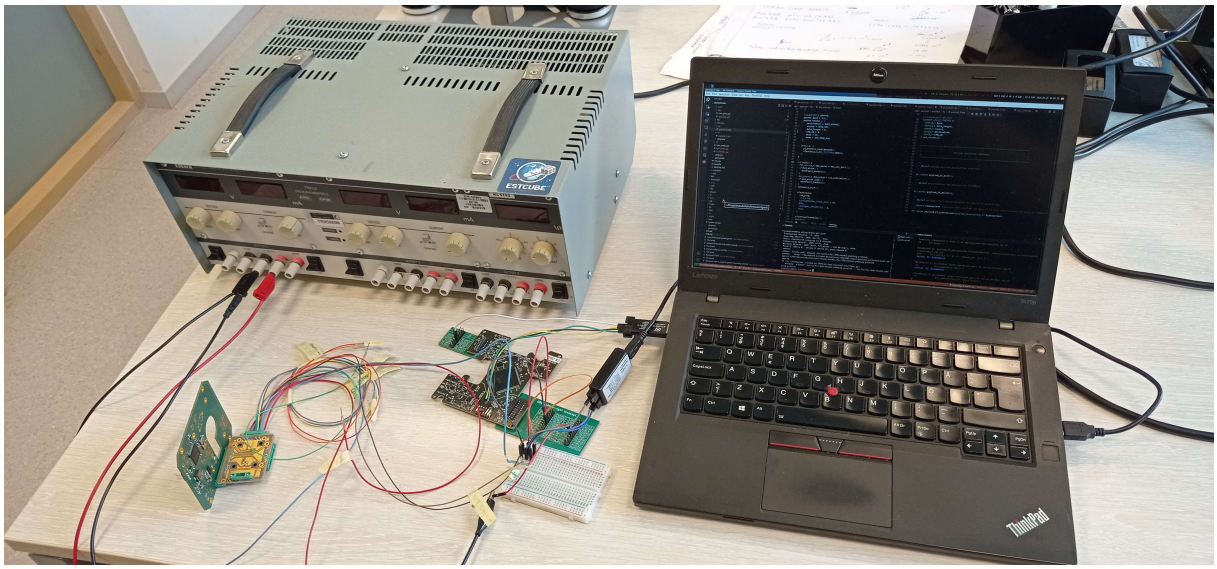

Figure 16: Testing the plasma brake through the OBC payload interface.

6 Future work

In the later stage of code analysis it was found that if the payload randomly reboots, there remains a risk that it will start listening to the payload bus again, even if a deafen command was sent before. This problem could be mitigated by powering off the device until communication with the other payloads is complete. Another possible way to fix this issue would be to save the current listening state to non-volatile memory instead of the internal RAM.

Another feature that would aid with using the cameras in space would be to connect CANTerminal directly to Mission Control. This way, it would be possible to send commands to the cameras through Mission Control as the cameras were tested in the lab. Directly connecting the camera user interface would reduce the number of potential software issues as all the commands have already been implemented in CANTerminal and would not have to be rewritten to the new environment. In addition, it would be possible to use the known environment for debugging the cameras in space.

Lastly, the author recognizes that the camera firmware lacks the feature of triggering an image capture using an interrupt. That feature would make it possible to take an image with both cameras at precisely the same time. Taking an image this way is required for calculating the Normalized Difference Vegetation Index, as it requires an image captured of the same area on Earth by both cameras. To implement such a command, it would be necessary to modify the interrupt handler to make it work in two modes. If the camera is listening to the payload bus, the interrupt will act as a trigger to capture an image. In the deafened state it would make the camera listen to the payload bus again. Creating this functionality and testing it remained out of the scope of this thesis.

7 Conclusion

The primary objective of this Bachelor's thesis was to test the ESTCube-2 payloads; EO cameras, and plasma brake, along with the necessary debugging and troubleshooting efforts to ensure their proper functionality. This thesis extensively explores the communication protocols, the graphical user interface for the cameras, and software development for integration and effective utilization of the ESTCube-2 payloads within the satellite. The main contributions of this thesis were:

- CANTerminal and its extensions were updated and could now be used to further develop and debug firmware for the ESTCube-2 cameras.
- OBC was provided with the capability to convert ICP packets into payload protocol packets and vice versa, including the required testing procedures.
- The remaining issues in ESTCube-2 flight model cameras were resolved and tested.
- The communication with the plasma brake and cameras through OBC was tested.

In addition, the author created an adapter for the cameras for testing and debugging communication through their payload bus connector. Using this, it was possible to simulate sending messages from the OBC to the cameras using the CANTerminal.

Acknowledgements

The author would like to express his deepest gratitude to his thesis instructors, Indrek Sünter and Kristo Allaje, for providing support, guidance, and expertise throughout the entire process.

The author would also like to thank other ESTCube team members for their guidance, encouragement, and for giving ideas to improve the quality of the work.

Additionally, the author acknowledges Terrence Andrew Davis for being a remarkable inspiration with his unique approach to programming and unwavering dedication.

Finally, the author would like to express his appreciation to his family and friends for their support in writing this thesis.

/signed digitally/

References

[1] Common European Research Classification Scheme (CERCS) Teadusvaldkondade ja -erialade klassifikaator PDF: https://wiki.ut.ee/download/attachments/16581162/Common%20European%20Research%20Classification%20Scheme.pdf

[2] ESA 75 years since the first liquid-fueled rocket launch, Link: https://www.esa.int/Enabling_Support/Space_Transportation/75_years_since_the_first_liquid-fueled_rocket_launch2, 2001, Retrieved: 24.10.2022

[3] NASA JPL, "20 Inventions we wouldn't have without space travel", Link: https://www.jpl.nasa.gov/infographics/20-inventions-we-wouldnt-have-without-space-travel, 2016, Retrieved: 24.10.2022.

[4] T. Villela, C. A. Costa, A. M. Brandão, F. T. Bueno, and R. Leonardi. "Towards the Thousandth CubeSat: A Statistical Overview.", 2019, 1-13, DOI:10.1155/2019/5063145

[5] Cal Poly – San Luis Obispo, CA, "CubeSat Design Specification", PDF:https://www.cubesat.org/s/CDS-REV14_1-2022-02-09.pdf

[6] Erik Kulu, "Nanosatelliitide tehnoloogia arengutrendid.", 2014, 16-17, HDL:10062/41683

[7] Silver Lätt et al. "ESTCube-1 nanosatellite for electric solar wind sail in-orbit technology demonstration", 2014, DOI:10.3176/proc.2014.2S.01

[8] A. Slavinskis et al., *Nanospacecraft fleet for multi-asteroid touring with electric solar wind sails*, 2018 IEEE Aerospace Conference, Big Sky, MT, USA, 2018, pp. 1-20, DOI:10.1109/AERO.2018.8396670

[9] Iakubivskyi et al. "Coulomb drag propulsion experiments of ESTCube-2 and FORESAIL-1", 2015, DOI:10.1016/j.actaastro.2019.11.030.

[10] ESTCube internal documentation, "ICP", https://confluence.tudengisatelliit.ut.ee:8433/display/EC2 2022, 24.10.2022

[11] S. Tammesoo "Suhtlusprotokoll ESTCube-2 alamsüsteemide vaheliseks suhtluseks", 2015, 16-21, HDL:10062/50465

[12] ESEO student satellite successfully launched to space, Link: https://www.esa.int/Education/ESEO/ESEO_student_satellite_successfully_launched_to_space, 2018, Retrieved: 30.03.2023

[13] European Student Earth Orbiter: ESA's educational Microsatellite Program, Link: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=2974&context=smallsat, 2013

[14] Private conversations with Indrek Sünter

[15] ESEO micro cameras, Link: https://www.esa.int/Education/ESEO/Micro_Cameras, 2018, Retrieved: 30.03.2023

[16] I. Sünter, H. Kuuste, A. Slavinskis, A. Agu, E. Ilbis, G. Olentšenko, I. Iakubivskyi, J. L. L. Seco, J. Kütt, R. Vendt et al., *Design and testing of a dual-camera payload for ESEO*, 67th International Astronautical Congress, B4, 2016

[17] MT9P031 - 1/2.5-Inch 5 Mp CMOS Digital Image Sensor datasheet, PDF:https://www.onsemi.com/download/data-sheet/pdf/mt9p031-d.pdf

[18] Finnish Meteorological Institute, (2020), ESTCube-2 Plasma Brake Communication Protocol, Unpublished internal company document

[19] Qt for Python wiki, Link: https://wiki.qt.io/Qt_for_Python, 2023, Retrieved: 30.03.2023

[20] Terrestrial Cosmic Ray Induced Soft Errors and Large-Scale FPGA Systems in the Cloud, Link: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1562&context=spacegrant, 2019

[21] Saleae homepage, Specifications, https://www.saleae.com/, Retrieved: 18.05.2023

[22] IS42S16160G-7TLI datasheet, Mouser, https://www.mouser.ee/datasheet/2/198/42-45S83200G-16160G-258274.pdf, Retrieved: 15.05.2023

[23] IS42S16320D-7TLI datasheet, Mouser, https://www.mouser.ee/datasheet/2/198/42-45R-S_86400D-16320D-32160D-258456.pdf, Retrieved: 15.05.2023

[24] Google Drive containing ESEO and ESTCube-2 camera testing pictures, Link: https://drive.google.com/drive/folders/1FLQkEJNvHdKl9ULAFjTYigpGZrGyHfvk, 2023

A Camera images

Figure 17 is a picture produced with incorrect memory timings. Figure 18 is the first image the author semi-successfully took with the ESEO primary camera. Figure 19 contains an image that was taken without optics while testing memories. Figures 20 and 21 show successfully taken monochrome photos of an ESTCube-2 team member Michelle Lukken and an observatory dome along with a playground. Other images taken during testing can be found in the Google Drive[24].

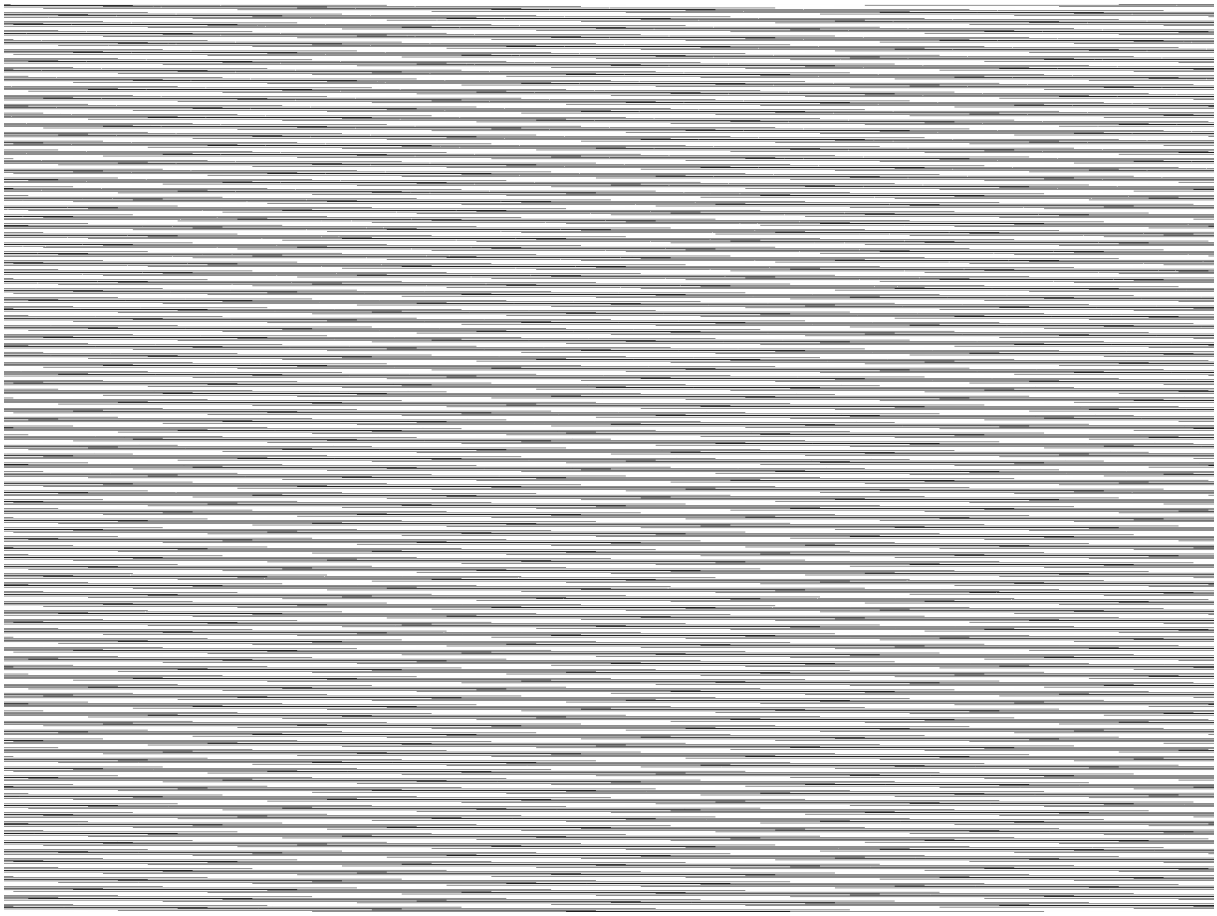

Figure 17: ESTCube-2 camera image with incorrect memory timings.

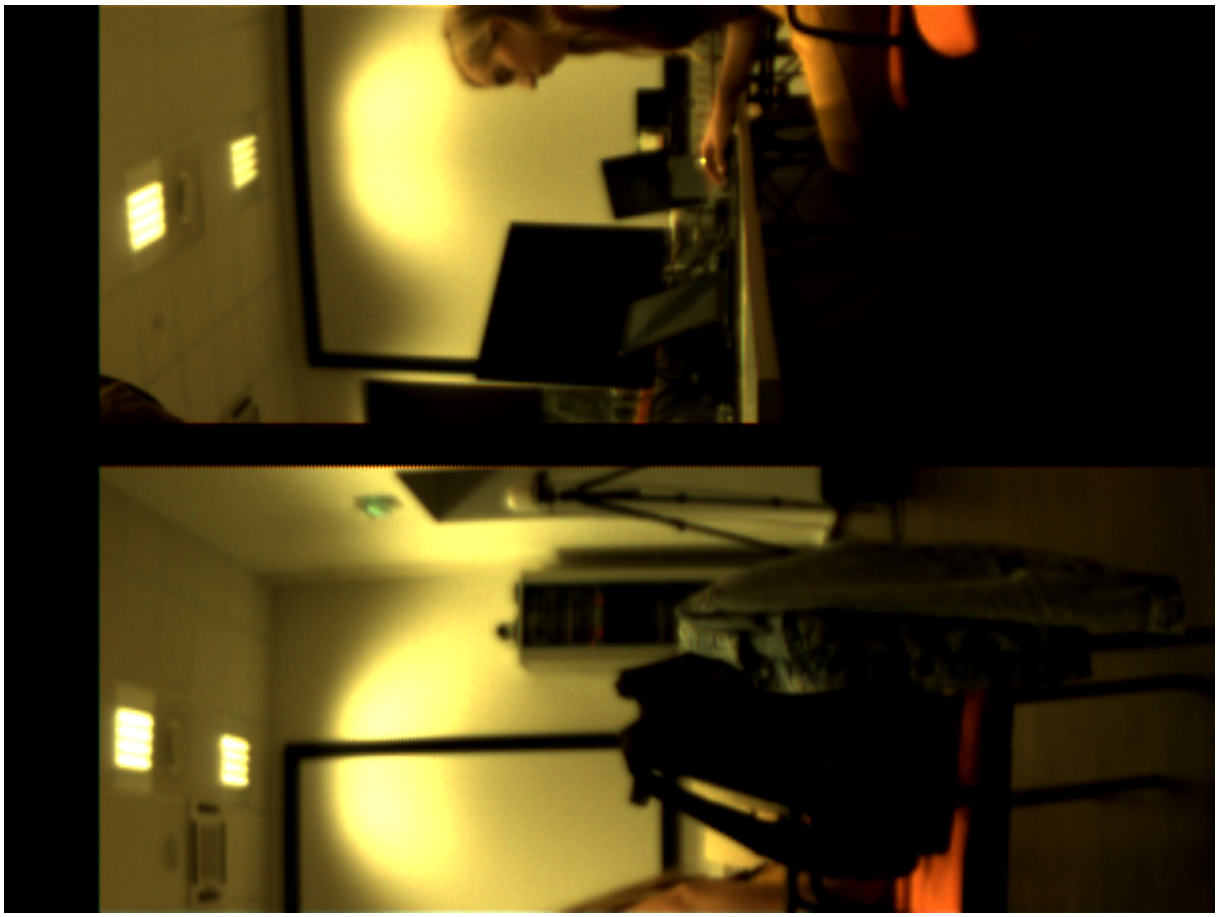

Figure 18: First picture taken with the ESEO primary camera and using updated PyCAM.

Figure 19: ESTCube-2 camera image without optics.

Figure 20: A picture of Michelle taken with an ESTCube-2 camera.

Figure 21: A picture containing a playground and a dome of a telescope tower taken with an ESTCube-2 camera.

B Camera helper printed circuit board design

Figures 22 and 23 show the pictures of the assembled helper circuit board. Figure 24 contains the board schematic, and Figures 25 and 26 contain both layers of the printed circuit board as seen in Altium Designer 18.

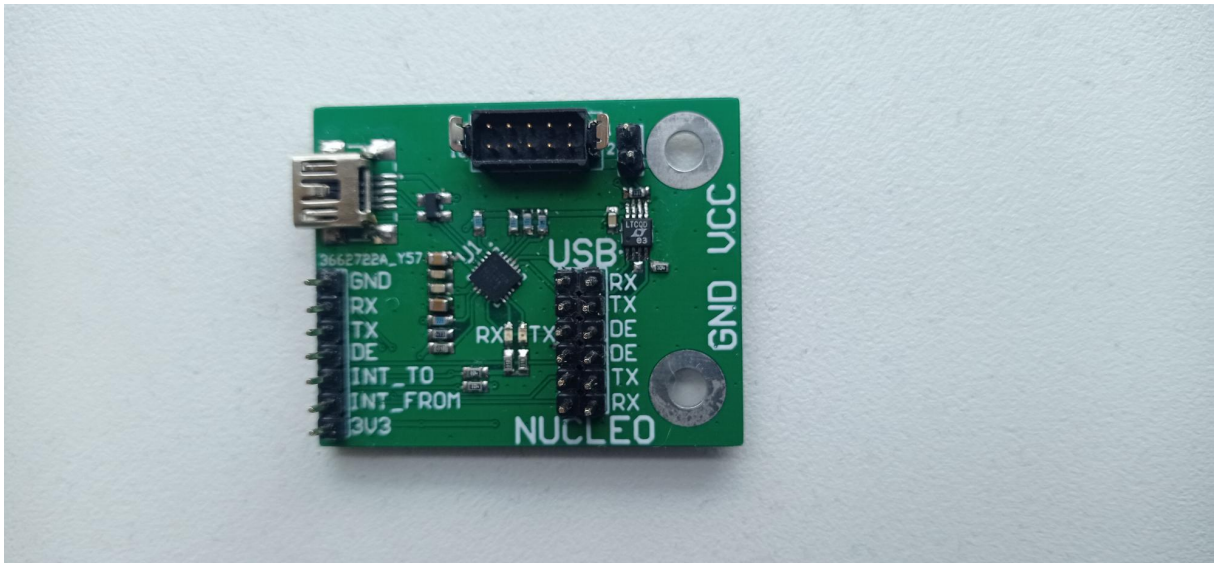

Figure 22: Top side of the camera helper printed circuit board.

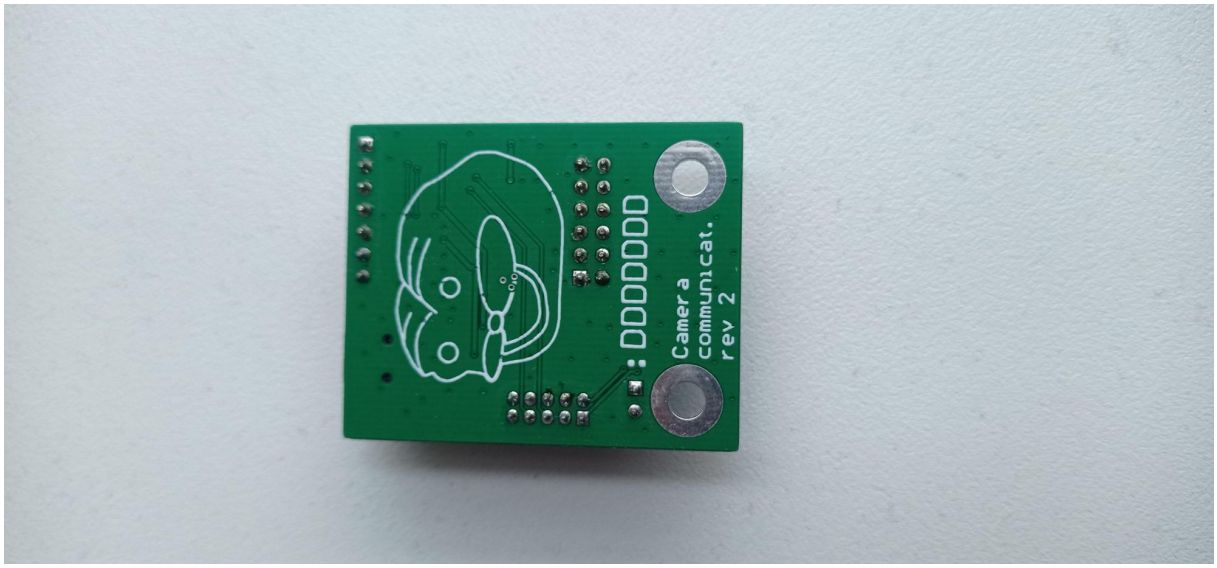

Figure 23: Bottom side of the camera helper printed circuit board.

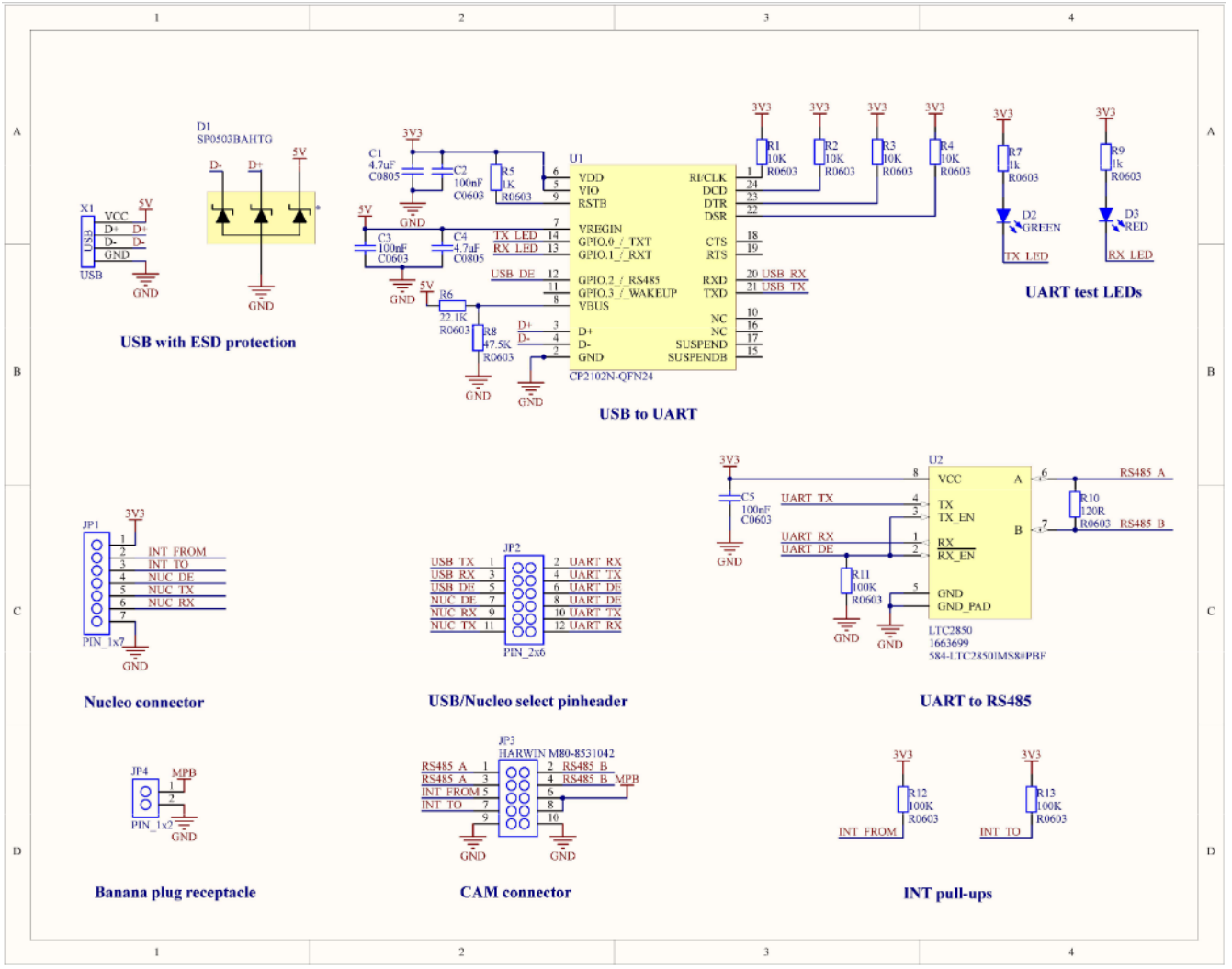

Figure 24: Camera helper printed circuit board schematic.

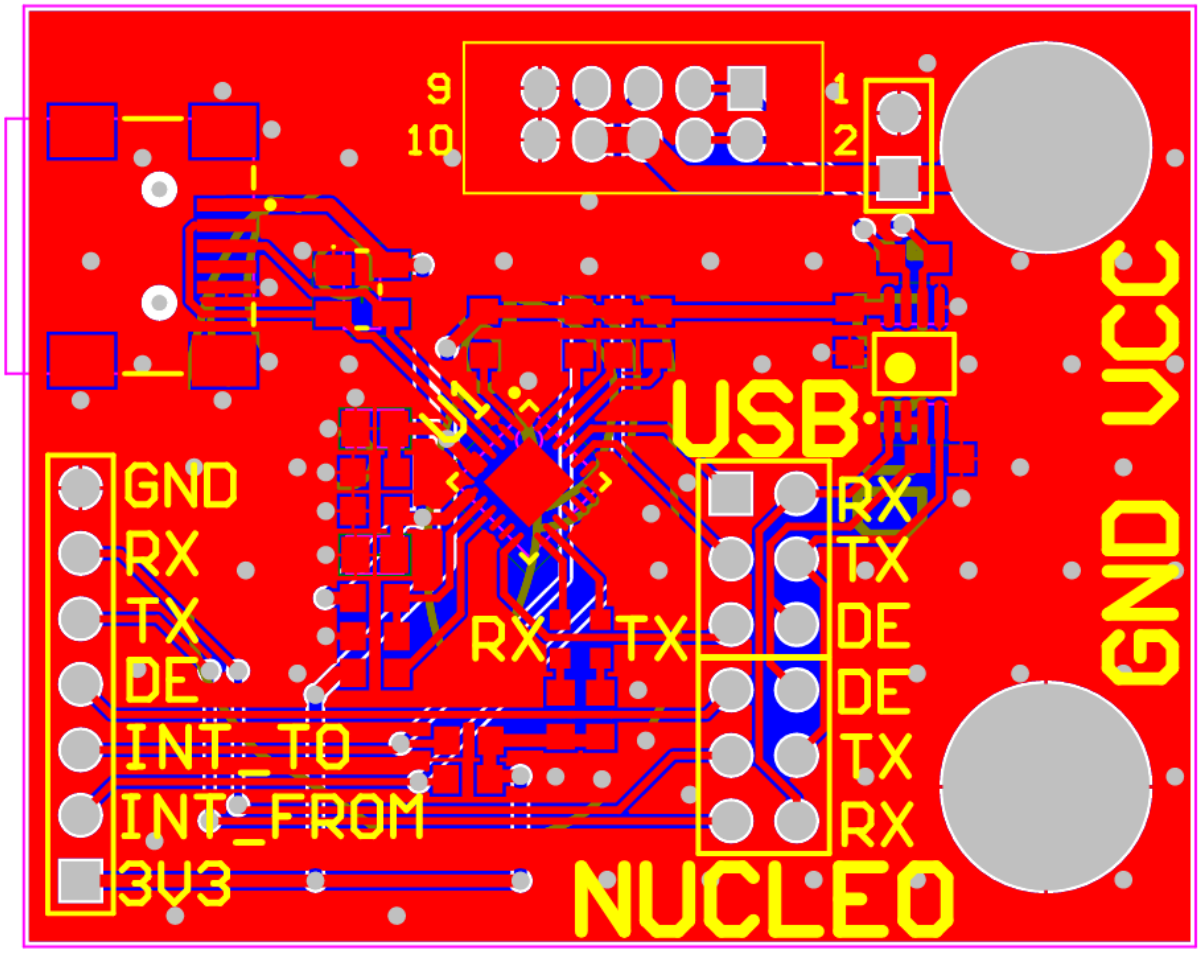

Figure 25: Top layer of the camera helper printed circuit board.

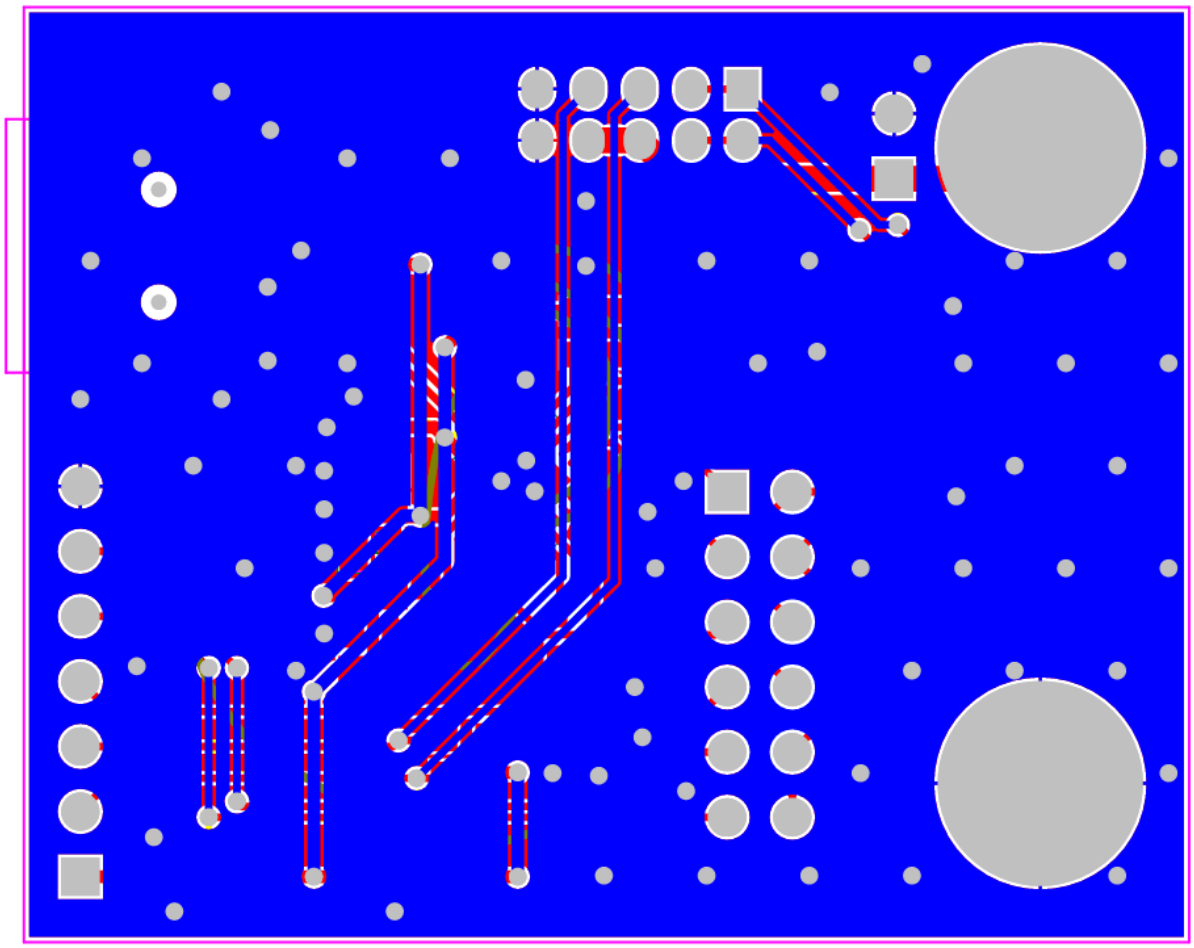

Figure 26: Bottom layer of the camera helper printed circuit board.

Non-exclusive license to reproduce thesis and make thesis public

I, Laur Edvard Lindmaa

1. herewith grant the University of Tartu a free permit (non-exclusive license) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   "Interface development for ESTCube-2 cameras and plasma brake"

   supervised by Indrek Sünter and Kristo Allaje

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Laur Edvard Lindmaa
20/05/2023