Development of the UnoSat Platform for stratospheric balloon payloads

Sebastian Scholz

Space Engineering, master's level (120 credits)
2023

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Master in Space technology and Instrumentation

University III Paul Sabatier

&

Joint Master Programme in Space Science and Technology - Spacemasters

Luleå University of Technology

Master Thesis

Development of the UnoSat Platform for stratospheric balloon payloads

Organisation: IRAP/Paul Sabatier: Université Toulouse III

Author: Sebastian Scholz
Supervisor (UPS): Hassan Sabbah
Supervisors (LTU): Soheil Sadeghi & Thomas Kuhn

Academic year 2021/2022
Abstract

The UnoSat Platform is a software and 3D structure platform built with Arduino Uno boards that allows easier implementation of a satellite project in a 1-n unit CubeSat format. It is designed to improve and speed up development of balloon-nanosat projects of the M2 TSI master. It provides a 3D-printable Lego based structure that can be extended by stacking multiple pieces on top of each other. The structure parts allow to quickly increase the number of Arduinos and add shields on top of already integrated Arduinos. Because of this, the structure allows for easy prototyping, and printed pieces can be reused in future projects. The platform also provides a communication system that is very efficient and reduces the possibilities of programming mistakes when implementing communication. It allows serializing and parsing data into and from a binary format, provides protection against transmission errors via cyclic redundancy checksum and allows dynamically sized message payloads. Additionally, it supports one way or both ways communication not only between a groundstation and an embedded device, but also between two embedded devices. The code on the embedded device is generated specifically for a communication configuration, which makes it very efficient in terms of processing and memory usage on the embedded device.
Résumé

La plateforme UnoSat est une plateforme logicielle et de structure 3D construite avec des cartes Arduino Uno qui permet de mettre en œuvre plus facilement un projet de satellite au format CubeSat à 1 unité. Elle est conçue pour améliorer et accélérer le développement des projets de ballons - nanosat du maître M2 TSI. Il fournit une structure extensible à base de Lego qui peut être imprimée en 3D en plusieurs pièces qui peuvent être empilées les unes sur les autres. Les pièces de la structure permettent d’augmenter rapidement le nombre d’Arduinos et d’ajouter des shields par-dessus les Arduinos déjà intégrés. De ce fait, la structure permet un prototypage facile, et les pièces imprimées peuvent être réutilisées dans de futurs projets. La plate-forme fournit également un système de communication très efficace qui réduit les possibilités d’erreurs de programmation lors de la mise en œuvre de la communication. Elle permet de sérialiser et d’analyser les données dans et à partir d’un format binaire, offre une protection contre les erreurs de transmission par le biais d’une somme de contrôle de redondance cyclique et permet de dimensionner dynamiquement les charges utiles des messages. En outre, il prend en charge la communication unidirectionnelle ou bidirectionnelle non seulement entre une station au sol et un dispositif embarqué, mais aussi entre deux dispositifs embarqués. Le code sur le dispositif embarqué est généré spécifiquement pour une configuration de communication, ce qui le rend très efficace en termes de traitement et d’utilisation de la mémoire sur le dispositif embarqué.
Acknowledgement

I would like to thank Peter von Ballmoos for the opportunity to do this internship at IRAP with the Université Toulouse Paul Sabatier. A special thanks to my supervisor at IRAP, Hassan Sabbah, for his help and guidance during the internship. Furthermore, I would like to thank Soheil Sadeghi and Thomas Kuhn for their remote support and guidance from the LTU side.

Lastly, I want to thank my friends and family for their encouragement and support throughout the master.
List of Figures

1. All structure components for the UnoSat. ........................................... 4
2. The 3D printed base structure part. ....................................................... 5
3. The 3D printed Base structure part with an Arduino Uno. ....................... 5
4. The 3D printed Base Head structure. ..................................................... 5
5. The 3D printed Full long side structure. ................................................. 6
6. The 3D printed 2 Straight structure. ....................................................... 6
7. The 3D printed 3 Straight structure. ........................................................ 7
8. The 3D printed Box structure. ............................................................... 7
9. The bottom of the 3D printed Top structure. ......................................... 7
10. Two Arduino Unos, one with an SD-Card shield in a full UnoSat structure. .. 9
11. Two Arduino Unos, one with an SD-Card shield in a full UnoSat structure taken apart. ................................................................. 9
12. Uno Sat example with SD-Card shield. .................................................. 19
13. Uno Sat example with RTK GPS module. ............................................. 20
14. Uno Sat example with Grove Beginner Kit for Arduino. ......................... 22
15. Uno Sat example with LoRa shield. ...................................................... 23
16. The front page of the new website. ....................................................... 25
17. The front page of the old website. ....................................................... 25
18. The front page of the new website on a smaller screen. ......................... 25
19. The editor view of a balloon project subpage on the new website. ........... 26

List of Tables

1. Layout of a LOG telemetry package as defined in the template. .................. 16
1 Introduction

Every year since 2016-2017, the students of the M2 TSI at Paul Sabatier: Université Toulouse III develop two balloon projects. These projects are built to simulate the development of an actual satellite project and are each flown on a stratospheric balloon by CNES, reaching an altitude of up to 30 kilometers. The balloon projects started out as a simple Arduino with some sensors, but improve and increase in complexity each year, as the students have more and more heritage to use when developing their project.

The first balloon project of 2016-2017 consisted of an Arduino Mega board with an SD-Card (Secure Digital memory storage card), which was used to store data onboard from the payload. The payload included temperature, pressure, humidity, and luminosity sensors, a gyroscope, accelerometer, magnetometer sensor and a GPS (Global Positioning System) receiver. For the GPS receiver, a u-blox module was used, because it is capable of providing GPS position data at altitudes above 20 kilometers, where many other GPS receivers do not work. Additionally, a GoPro camera was added, which contained its own battery and data storage. The whole system was developed in only one week and did not include any communication or groundstation.

In 2017-2018, the balloon project was also built with an Arduino Mega, with a Grove shield that provided more I2C (Inter-Integrated Circuit) ports. It included a type 5 radiation sensor, a Grove humidity and temperature sensor, a Grove barometer and a high accuracy temperature sensor. There was no communication and the data was stored on an SD-Card shield.

For 2018/2019, the students created two projects. For the first one, HAMLET (High Altitude Measurement and Live Telemetry), they decided to use a Raspberry Pi instead of an Arduino Uno. The Raspberry Pi was used to store data on it’s SD-Card and was also connected to an accelerometer, temperature sensor, GPS receiver and an X-bee low energy radio communication shield called Sigbee. The Sigbee module together with an antenna allows for communication over 20 kilometers. The Sigbee shield was used to send data and another one was used to receive the data at a groundstation, where it was displayed in graphs using a Python script. This marked the first year with a balloon project that had communication capabilities and they were able to transmit data during the first 30 minutes of the flight, before the signal became too weak. The second project focused on air quality. It was built with a Raspberry Pi and an Arduino Leonardo. They used the same Sigbee module for communication.

The balloon projects in 2019/2020 were the first to miniaturize the project into the CubeSat format. They designed their own PCB (Printed Circuit Board) and used a TEENSY 3.6 developer board [1], which contained an SD-Card and was connected to temperature and humidity, Ozone, UV, GPS, barometer and radiation sensors. They also sent data over a Sigbee module and had a Python based groundstation, just like the previous year. Additionally, they were the first to use a drone to test flight their system.

In 2020/2021, there were two balloon projects: The AEGIS project used an ESP32 TTGO LoRa board [2], which includes an LCD screen, SD-Card and a LoRa module. LoRa comes from ”long range” and is able to send data over 10 kilometers with low data rate and low power consumption [3]. It can be operated at different frequencies, for this project 440 MHz was chosen. LoRa communication can be direct from a sender to a transmitter, but it is also
possible to use LoRa gateways, which store the data for retrieval on the LoRa server [4]. The LoRa module was used in addition to a Sigfox module. Sigfox is a 0G global network where the data is send via base stations and stored in the cloud [5]. Both communication modules were used to transmit data. The whole project was built as a one unit CubeSat. The other project, ATILA, was focusing on measuring the air quality. It also used an ESP32 TTGO LoRa board with SD-Card and transmitted data using the LoRa module.

Last years student, in 2021/2022, created two balloon projects and one laser pointing project. KERES reused the sensor platform from the previous year and added more sensors. An Arduino Uno was used as the main controller, which was responsible for collecting the data and sending it to an ESP32 TTGO LoRa board, which forwarded it to the ground-station. SPARC (Stratospheric Particle Accumulation and Reaction Control) had for the first time a pair of reaction wheels onboard, as well as a particle collection system. Just like KERES, an Arduino Uno was used as the main controller and an ESP32 TTGO LoRa board for sending data to the groundstation. Both systems were using the LoRa communication with the groundstation on the launch site. This year, some data was also successfully received over 100 km in Toulouse at the university with a Yagi antenna. Both projects also included an ArduSimple simpleRTK2B module [6]. RTK (Real-time kinematic positioning) uses a base station with a well known position in addition to GPS positioning to increase its accuracy to a few centimeters. The module and base include a X-bee communication module to transmit location data and location corrections. Using this, centimeter precise position was available for the first time, although only at the ground station and not on board of the balloon, because no communication between the RTK system and the Arduino was implemented. The laser pointing project developed a structure on the ground that was using the RTK GPS position of the balloons to try and hit a light sensor on the balloons with a laser.

Most of the balloon projects from the previous year used an Arduino microcontroller as the main controller in their system. Arduinos are very well suited for the balloon projects, because they are cheap and easily available, so it is possible for each student to have one Arduino on which they can develop. It is also very easy to program the Arduino, it doesn’t require a lot of knowledge to interact with hardware components, and there are a lot of tutorials and other resources available online which document how to program on the Arduino. There also exist a lot of libraries, which are finished pieces of code, that can be imported and used. While other microcontroller require a special software, wire or other hardware to program, the Arduino just requires the programmer to connect the USB cable and to hit a button. Because of this, the UnoSat platform also uses Arduino microcontroller. Specifically, the Arduino Uno was chosen. It is very constrained on memory and available processing power, but it fits into the 10 by 10 centimeter constrains of the CubeSat format.

The students rely on heritage from old balloon projects when developing their own projects, which means it should be as accessible as possible. Because of this, the website of the M2 TSI master will be replaced by a new improved version, that includes a searchable section that can list articles about all past and future balloon projects. In the same spirit of easing access and sharing, special care was taken for the UnoSat platform to only use software and products that are free to use for students. The M2 TSI Master and the Université Paul Sabatier are part of the UNIVERSEH alliance, which is committed to provide easier access
and sharing of knowledge. This is why the code developed for the UnoSat platform is open source and accessible for everyone online at GitHub \cite{7} and GitLab \cite{8}. GitLab and GitHub also provide an area to ask questions and even to propose improvements. The files required to print the structure are also available online at GitHub accessible to everyone, but also the link to the project that was used to create the 3D models. This public accessibility allows not only the students from the M2 TSI master to use the UnoSat platform, but everyone can find, use and improve the system.

2 Heritage

The UnoSat platform is designed to reuse components from existing balloon projects of the M2 TSI master and to improve them.

In previous projects, the communication was done using multiple transport technologies, like X-bee, Sigfox and LoRa. But all projects were transmitting their data in plain ASCII text, without any checksum or verification of the transmitted data. The two balloon projects of 2022 both had a main Arduino which would send its data in an ASCII format via an UART connection to a TTGO LoRa32 V2.0 module, where the data would get saved on an SD-Card, prepended with a header and time and then sent via the integrated LoRa connection. On the groundstation side, the data was received in another TTGO LoRa32 V2.0 module and then forwarded via UART to the groundstation \cite{9} on a connected computer.

Both of the balloon projects of 2022 had an Ardusimple simpleRTK2B module \cite{6} on board and were able to get a centimeter precision GPS location, but the module was separate from the main onboard data handling system. Because the main onboard data handling system was unable to interface with the simpleRTK2B module directly, the module was also and unable to provide communication. Because of this, the position data was only available on ground while the balloon was within a 12 kilometer radius and wireless communication with the simpleRTK2B board was possible. Storing of the position or using the position for calculations or triggers on the Arduino was not possible.

Each year the students developed a new ground station software for their balloon projects. These groundstations are specific to the communication protocol used in each specific year. The groundstation from 2022 balloon project \cite{9} however was more flexible, because it allowed to dynamically change its interface via drag and drop and also dynamically add new graphs.

The UnoSat platform improves parts from the existing heritages and provides ready to use modules which can be used as a starting point for future projects. All modules are Arduino Uno based, as past balloon projects have shown that the Arduino is very suitable for the balloon project. The existing interface with the SD-Card and LoRa module have each been adopted into such a module (see Section 3.4.1 and Section 3.4.1). Building on experience with the Ardusimple simpleRTK2B module, an RTK GPS module was developed that improves on the existing usage by allowing direct interfacing with the RTK module, providing high precision GPS measurements on board (see Section 3.4.2).

3 Platform Overview

The UnoSat platform proposes a solution for two problems that are required to solve in every balloon and satellite project: Structure and communication. Even though the platform
was developed to be used in balloon projects, it can also be used in satellite projects which face the same problems. For both problems, a solution was developed that can be reused with many different projects. It also provides a starting point for new projects, so that the students don’t have to reinvent common solutions every year.

3.1 Structure

In order to quickly develop a balloon or satellite project, an extensible and quickly adaptable structure for the UnoSat platform was designed. Its outside dimensions are 90 mm by 74 mm by \( h \), where \( h \) is the height that depends on the number and type of pieces used. Because of this, the structure conforms to the CubeSat form factor and can create a 1 to \( n \) unit CubeSat. The CubeSat format specifies that a 1 unit CubeSat must have a dimension of 10 by 10 by 10 centimeters. Each additional CubeSat unit increases the allowed height of the structure by 10 centimeters. The UnoSat structure is actually smaller than the specified 10 by 10 centimeters, which allows the students to add additional padding and isolation material around the core structure and still fit inside the CubeSat form factor.

These dimensions were chosen, because the CubeSat form factor is very commonly used for satellite projects. The balloon projects are also constrained in mass to a maximum of 2.5 kg, and the size constraints help to keep the mass constraints. The only Arduinos that would fit into the 10 by 10 centimeter area in a flat configuration are the Arduino Uno and Arduino Leonardo. The Arduino Leonardo has the same size and memory available as the Arduino Uno, so the Arduino Uno was chosen. The PCB has a dimension of 68.6 by 53.3 millimeter, but the power jack and Ethernet port overhang on one side [10]. The UnoSat structure was designed to tolerate the overhang by leaving out one side. The Arduino shields components that can be mounted on top of the Arduino Uno mostly also adhere to the size of the Arduino Uno PCB. This is why the UnoSat structure provides an inside payload area of 70 by 54 millimeters.

The structure is based on the Lego system, which means that it has small cylindrical extrusions on the top (which are called Lego heads), and holes on the bottom where the Lego heads can fit into. It is in theory compatible with actual Lego pieces in terms of dimensions, but this has not been tested. This was done to allow prototyping with Lego pieces rather than having to 3D print all pieces at the beginning. The Lego approach allows to quickly adapt the structure to new requirements by rearranging pieces or adding new pieces on top of old ones. As well as reusing so far unused already printed pieces for reasons of thrift, reusability and sustainability.
The structure consists of individual 3D printable pieces, that can be stacked on top of each other to create the structure. The different pieces are as follows:

**Base**
The base piece is the bottom of the structure, but it can still be put on top of another piece. In that case, cables to connect to the Arduino below have to be put outside of the structure. It has an area to put an Arduino Uno. The base piece has an opening in the front to leave space for the Ethernet and power ports of the Arduino. Its inside structure is mapped precisely to the form of the Arduino Uno with a padding of 0.5 mm, with structures to allow to fixate the PCB onto the structure. The outer dimensions of the base are 90 mm by 74 mm by 10 mm, not including the Lego heads.

![Figure 2: The 3D printed base structure part.](image)

![Figure 3: The 3D printed Base structure part with an Arduino Uno.](image)

**Base Head**
This part is a wall all around space for an Arduino in the middle, except for the front, where it has an opening in the front to leave space for the Ethernet and power ports of the Arduino. The outer dimensions are 90 mm by 74 mm by 10 mm, not including the Lego heads. The inner free area is 70 mm by 54 mm.

![Figure 4: The 3D printed Base Head structure.](image)
**Full Circle**
This part is a wall all around space for an Arduino in the middle without an opening. The outer dimensions are 90 mm by 74 mm by 10 mm, not including the Lego heads. The inner free area is 70 mm by 54 mm.

![Figure 5: The 3D printed Full long side structure.](image)

**Full Long Side**
This part is a long straight piece that can be used to build one of the side walls. Its dimensions are 90 mm by 10 mm by 10 mm, not including the Lego heads.

![Figure 6: The 3D printed 2 Straight structure.](image)

**2 Straight**
This piece covers 2 Lego heads and can be used to construct part of the wall. Its dimensions are 18 mm by 10 mm by 10 mm, not including the Lego heads.
3 Straight
This piece covers 3 Lego heads and can be used to construct part of the wall. Its dimensions are 26 mm by 10 mm by 10 mm, not including the Lego heads.

Box
The box allows to cover more height without covering all of the sides. The outer dimensions are 90 mm by 74 mm by 50 mm, not including the Lego heads. The inner free area is 70 mm by 54 mm.

Top
This piece is the very top of the structure and is the only piece that can not accept another piece on top of it. The outer dimensions are 90 mm by 74 mm by 50 mm. The inner dimensions are 70 mm by 54 mm by 7 mm.
3.1.1 Onshape

Access to 3D models of these pieces is provided via the GitHub repository [7], which allows to 3D print the pieces. In addition to that, access to the Onshape project has also been shared [11].

Onshape [12] is a website that provides an editor for 3D models. It can be used for collaborative editing and sharing of 3D model projects. It requires a payed account to access the editing features, but is free for students.

Having access to the project allows everyone with an Onshape account to copy the project and then change it and adapt the 3D models. The project is built with variables, which allows to easily change certain dimensions of all models. By editing the variable, the models update automatically. The following variables are available to change:

- **legoHeadDiameter**: This allows to change the diameter of the Lego heads, which are the mechanism to stick pieces together.
- **legoHeadHeight**: This parameter determines, how high the Lego head will be extruded from the top of the structure, and how deep the holes in the bottom will be.
- **legoHeadCenterDistance**: The distance between the centers of each Lego head. This controls how far apart the Lego heads are.
- **legoHeadPadding**: The padding that is applied to the hole in the bottom of the pieces where the Lego heads are inserted.
- **legoPartWidth**: This is the width of both walls, which contain the Lego heads and holes.
- **arduinoPadding**: Padding to the Arduino PCB on the inside.
- **groundThickness**: Used for the base and head part, defines the thickness of the middle part.
- **totalWidth**: The total width of the structure.
- **totalLength**: The total length of the structure.
- **totalHeight**: The height of the structure walls, not including the Lego heads.

In addition to these, which apply to all parts, the following variables only apply to specific parts:

- **pcbThickness**: For the Base, this defines how much space should be given for the Arduino PCB.
- **extendableHeight**: For the Box, this defines the height of the piece.
**Printing**

The ability of the pieces to stick together mostly depends on the print quality. The model gives a margin of 0.3 mm between the pieces, which equals the typical margin of the available 3D printers. This is enough to make most pieces stick relatively good, although not with the quality of actual Lego pieces. Especially the small pieces have a high chance of not being sticky enough, because of the limited number of Lego heads on the piece. This restriction will vanish with future 3D printers with a higher resolution.

As it is possible for the structural pieces to detach and be rearranged easily, the structure can be very helpful during the prototyping phase of the satellite project. But because of these qualities, it is not recommended to use the structure as it is during the balloon flight. However, the structure can be used either by merging the pieces in a 3D modeling software and then printing them as one piece, or by gluing individual pieces together. Because the pieces share a lot of surface area, this will result in a rigid and fixed structure that is in theory suitable for flight, although that has not been proven experimentally.

Figure 10 shows a basic UnoSat stack with two Arduino Unos, where one of the Arduinos has an SD-Card shield equipped. The structure is built using a Base part which holds the first Arduino Uno with the SD-Card shield, three Base Head parts, where two were configured with a height of 9.6 mm, and one with a height of 6.2 mm. Stacked on top of the head parts is another Base part which holds the second Arduino Uno. On top of that, two Base Heads are stacked on top of each other, and on the very top a Top piece is placed. Figure 11 shows the same UnoSat stack, but taken apart at the second base piece, so both Arduino Unos are separate.

Figure 10: Two Arduino Unos, one with an SD-Card shield in a full UnoSat structure.

Figure 11: Two Arduino Unos, one with an SD-Card shield in a full UnoSat structure taken apart.

To demonstrate the feasibility and to provide a demonstration a few pieces have been printed using 3D printers of the Campus FabLab at Université Toulouse III [13]. The following pieces are available for the students to use: 1 Box, 7 2-Piece, 1 3-Piece, 1 Straight, 2 Base, 3 Base Head (9.6 mm height), 2 Base Head (6.2 mm height) and 1 Top piece.
3.2 Communication

One major goal of the UnoSat project is to simplify implementation of the communication system and to improve the efficiency and reliability of the communication, resulting directly in increasing the amount of data that can be sent and the ability to detect malformed messages.

Previously, all balloon projects used an ASCII based communication method, where each message package starts with some ASCII characters, includes all data in a textual representation, sometimes separated by a comma character, and ends with a newline. No verification of the data was performed.

This approach is easy to implement and parse, but very inefficient. If a positive number is available on the Arduino with a precision of 32bit, it can take up to 10 characters to represent the number as text, which is equal to 10 bytes. Because a floating point value has a very high range, to represent a 32 bit floating point value in text at least 8 character are needed after the decimal point and potentially 38 characters in front of it. Otherwise the conversion from the number into text is going to lose precision. So if all possible values of a 32 bit float should be representable in text, then 46 bytes are needed. In both cases, just transmitting the number in its binary format takes only 4 bytes. Additionally, no conversion has to be made on the Arduino’s site, only a copy to be performed to access the data. The reason for this is that the Arduino and many other microprocessors use numbers in the little-endian format, meaning that the number is stored in memory with the least significant byte first. This representation was also chosen as the representation of the data while transmitting it from the Arduino to the groundstation and vice versa. Therefore transmitting the data in a binary format doesn’t only make the transmitted data more compact, allowing for higher data rates, but also reduces the processing power required to create and read these packages on the embedded microcontroller.

When transmitting the telemetry packages wireless, but also while transmitting via a wire, electrically induced changes to the data can occur. These can cause one or many of the bits in a transmission to change, which changes the meaning of the data in the message. Previous balloon missions did not handle this possibility at all, which means that there is a chance that the received data was incorrect. But because the data was transmitted in ASCII text, only a very few specific bits permutations are possible that still result in a valid changed ASCII number. Any other change would have been easily detectable, because the received text wouldn’t represent a number anymore. In the binary representation it is not possible to detect a change of a bit, because each permutation of the bits in a binary number yield a changed, but still valid binary number. Because of this, two bytes of Cyclic Redundancy Checksum (CRC) are added to each package, which makes it possible to validate the contents of the package on the receiver. The calculated checksum is a CRC16, which the sender calculates with the data to be transmitted. The cyclic redundancy checksum is calculated by combining each byte of the payload using some mathematical operations into a 16 bit integer. Changes in the bytes of the payload will result in a different outcome. The receiver can then calculate the same CRC16 checksum with the data it received. When the two checksum values match, there is a high chance that the data was transmitted without an error. This is enough to catch all packages where only one bits were changed, and most of the packages with multiple bit changes.
With a checksum the receiver can determine if it has received invalid data, but it can not repair the data and reconstruct the original message. This could be done however by using an error correcting code (e.g. a Reed-Solomon-Code) instead of a simple checksum.

Finally, when implementing communication software, one has to make sure that both the receiver and transmitter write and read the data the same way and that both use the same definition of the communicated data. It is very easy for the receiver and the transmitter to diverge, either because a change was made only for one of them, or because the implementation for one of them has a flaw that the other one doesn’t have. To counter these problems, the communication system used by the UnoSat platform has only one definition of the transmitted data and it generates the serialization and deserialization code automatically.

3.2.1 ECom

The UnoSat platform uses the ECom library for implementing communication. This library was developed by the author of this thesis as part of the ASTER Rexus project and already included the communication definition, Python module and C code generator to generate code for sending telemetry and receiving telecommands. The library was adapted to handle Arduino Uno specific idiosyncrasies like the lack of a 64bit double precision floating point type. It was also extended to improve C++ support from the generated code and abstracting buffer access & checksum generation so some Arduino libraries can be used. Most importantly, ECom was extended to allow Arduino to Arduino communication by extending the C code generator to be able to generate code for both sides of the communication.

Rexus (Rocket EXperiments for University Students) and Bexus (Balloon EXperiments for University Students) are programs that are managed by ESA, where student groups can launch their experiment on two sounding rockets or two atmospheric balloons once each year. The program includes a selection process, multiple reviews by ESA members, multiple tests and a one week launch campaign at the Esrange space center in northern Sweden. The ASTER project (Attitude STabilized free falling ExpeRiment) aims to develop a high-performance, low-cost and easy to integrate attitude control system for free falling experiments ejected from sounding rockets. It is part of the Rexus Cycle 13 and scheduled for launch early in 2023.

The library provides a Python module which can be used to parse and serialize communication packages on higher level systems, which is intended to be used for the groundstation. It also provides a Python code generation script, which generates parsing and serialization code in C / C++ for low level embedded systems such as the Arduino Uno. The code is generated in a way that minimizes memory usage and copy operations on the embedded device. It also supports variable sized messages, where the size is included as part of the message. The ASTER experiment has not been launched yet at the time of writing, so the library has not been used and tested on a system that was launched. But it has been in development for over a year, and it has passed multiple reviews as part of the ASTER project.

The library also includes a simple groundstation as an example, which can be used to read the telemetry from the Arduino. This is the easiest way to get started receiving telemetry on the computer from a connected Arduino.

Communication database
The central entity of the library is the communication database. This database is a directory
with some files in it, describing the different communication packages. This database will be the single definition of the communication and needs to be provided by the developer of a project.

All files in the communication database folder that end with a `.csv` contain data in the Comma-Separated Values (CSV) format. Each file contains a table where the values of a row are separated by commas. The first line/row always contains the header of the table, which holds the names of the columns.

The directory can contain the following files and folders:

- **sharedConstants.csv**: Constant values that are used in the communication, which are values that are the same and shared between the receiver and sender. Each row of the table defines one shared constant, by defining it’s name, value, data type and an optional description.

- **sharedDataTypes.json**: This file is the only one that doesn’t contain a table. Instead, it contains a JSON (JavaScript Object Notation) object, which provides a description of data types that are used in the communication. A detailed documentation of the syntax in this file is available online. Defining shared data types allows to send structured and nested data in a communication message. There are two special data types: If the communication is supposed to support sending messages from the Arduino to the groundstation, a `TelemetryMessageHeader` data type needs to be defined, which defines how the header looks like that is prepended to every telemetry message. If the communication is supposed to support sending messages from the groundstation to the Arduino, then a `GroundStationMessage` data type needs to be defined.

- **telemetry.csv**: Contains a table of possible types of telemetry messages. Each row in the table defines one message that can be sent by the Arduino and received by the ground station, by defining the telemetry packages name and description. Each telemetry package defined this way needs a corresponding csv file in the `telemetryArguments` folder which defines the content of the telemetry package.

- **telemetryArguments**: This folder contains the definition of the data that is sent with each telemetry package. The definition must be in a file with the same name as the telemetry type and a csv file extension. Each file must contain a table, where each row defines a datum that is send with the message, by defining its name, data type and description.

- **commands.csv**: Contains a table of all telecommands and describes an optional return value. Each row in the table defines one message that can be send by the groundstation and received by the Arduino. The row defines the telecommand package name, whether it is a debug command or not, and a description. Optionally, if there should be a response to a telecommand, the row contains the name of the response, its data type and a description. Each telecommand package defined this way needs a corresponding csv file in the `commandArguments` folder which defines the content of the telecommand package, which are also known as the arguments of the telecommand.

- **commandArguments**: This folder contains the arguments for the telecommands. If a telecommand has arguments, they must be in a file with the same name as the
telecommand and a csv file extension. Each file must contain a table, where each row defines an argument that is send with the telecommand, by defining its name, data type, an optional default value and its description.

- **configuration.csv**: A description of configuration parameters that can be changed with a configuration telecommand. This is advanced functionality that was used in the ASTER project to allow reading and changing of some persistent system configuration on the embedded device and could be used with the UnoSat. As of this moment it is not used in any of the examples.

- **units.csv**: A description of units (see Units) and their base data types used in the communication. Each row in the table defines a unit by giving it a name, an underlying data type and a description.

**Units**
The ECom library supports defining units, which can be used instead of one of the raw data types that are supported. A unit is just a wrapper around a base data type, but it has two important roles:

1. It carries information on the physical unit of the data that is transmitted, so it can be interpreted directly. This also allows any groundstation to display the value correctly.

2. In the C and C++ code, each unit defined in the communication database automatically creates a programming data type, which is used in generated function arguments. These data types need to be explicitly created from their underlying data type and must also be explicitly converted back into the underlying type. In C++, they support subtraction, addition and comparison with the same unit type, but not with any other data type. This design prevents almost all unit related mistakes in the code, because the programmer is forced to consider the unit when doing any arithmetic, and the code will not compile if a different unit is used than expected. The following example demonstrates this:

In the first version of the code, no units are used:

```c
int maxAllowedTemperatureCelsius = 25;
int temperature = readTemperatureSensor();
bool isTooHot = temperature > maxAllowedTemperatureCelsius;
```

This code contains a mistake, because the `temperature` is measured in Kelvin, but compared with a temperature in degree Celsius.

The same example, but this time with units, and the `readTemperatureSensor` function returns a `kelvin_t`:

```c
kelvin_t temperature = readTemperatureSensor();
celsius_t maxAllowedTemperature = 25_celsius;
bool isTooHot = temperature > maxAllowedTemperature;
```

In this code, the mistake is much more obvious, and the code doesn’t compile, because in the last line two incompatible types are compared. The code also would not have compiled too, if the developer forgot to specify the unit of one of the temperatures.
Units can also be defined multiple times with different base types. This allows to transmit data with the same unit, but a different accuracy.

**Python library**

The ECom Python library provides classes that help to build a groundstation which is able to receive and parse telemetry and to send telecommands.

The important classes are the following:

- **CommunicationDatabase**: Given a path to a communication database, this class will load all information from the database and make them available. It provides a list of constants, telecommands, telemetry types, data types, units and configurations. For the telecommands and telemetry types, each type contains a list of data elements that are send with the command / telemetry, including the type for that datum. This allows a groundstation to show a user interface for entering telecommand arguments that dynamically adjusts to the content of the communication database.

- **TelemetryParser**: With a `CommunicationDatabase` instance, the parser can parse messages out of a received raw binary buffer. It handles verifying the checksum and loads the data into Python data types.

- **TelecommandSerializer**: Given a `CommunicationDatabase` instance, the serializer allows to convert telecommands and their data into a byte buffer that can be send to the Arduino.

- **ResponseTelemetryParser**: This class also requires a `CommunicationDatabase` instance and is both a parser and serializer for telemetry and telecommand messages. It should be used instead of a `TelemetryParser` and `TelecommandSerializer`, because it adds extra functionality for response telemetry to telecommands.

**Code generation**

The Python module contains a script that can generate C and C++ compatible code to parse telecommands and serialize telemetry. The code can be generated in two configurations, which can be chosen with a command line parameter to the script:

- In the normal mode, for each defined telemetry, a function is generated that takes all data points of the telemetry, calculates the checksum, creates the serialized message in a data buffer, and then calls a `writeTelemetry` function with the buffer and the size of the buffer. The `writeTelemetry` function has to be provided by the programmer, and it is supposed to send the contents of the buffer to the receiver.

- In base mode the code generated is for the receiver of the telemetry. Typically this would be a groundstation parsing the telemetry with the Python library, but with this mode one Arduino (referred to as the base) is able to parse telemetry send from another Arduino. A `parseMessage` function is defined to parse the telemetry and verify the checksum, which can be called with buffer accessor and a message handler. The buffer accessor must provide access to a buffer, where the raw received message bytes are
stored. This can be a circular buffer. The message handler is a structure containing functions that will be called when a message of a particular type has been parsed. The functions accept the data from the telemetry as arguments.

**Generated code efficiency**

The code is generated to be as memory efficient as possible and to reduce the numbers of copies necessary. By reducing the number of copy operations of the data, the number of copies taking up stack space is reduced, thus reducing the amount of memory needed. In the normal mode, the data is copied only once into the package buffer, which is exactly as big as required to contain the whole message. In the base mode, the data also has to be copied once. It is not possible to avoid this copy in most cases, because usage of numeric data values has certain alignment restrictions on certain microcontrollers. This means that when using a value e.g. in an arithmetic operation, each value must be located on a memory address that is a multiple of some value. To fulfill this requirement, each datum is copied to a variable on the stack, where the compiler makes sure it is well aligned. This way, only the exact amount of memory for the data and compiler generated padding is required.

3.3 Software Template

The UnoSat platform provides a software template which can be downloaded on the UnoSat project page in the template folder. It can be used with an Arduino Uno to get a basic system that is able to send telemetry via its serial port to a connected computer. This template can be copied and used as a starting point to build a satellite system on top of it.

The code contained in the template is written in C and C++. To run this code on an Arduino Uno, there are three steps:

1. The code needs to be translated from its text form into a form that the CPU on the Arduino Uno can understand and execute. This is done through a compiler. The documentation in the README.txt file in the template contains the command that needs to be run to execute the compiler. The same command will also install the necessary compiler, if it is not already installed.

2. The compiled code needs to be transferred to the Arduino Uno and has to replace the code that was previously there. This process is called "flashing". For it to work, the Arduino Uno needs to be connected via an USB cable and there must be nothing connected to pins 0 and 1 on the Arduino. To code from the template project is automatically flashed directly after building the code by executing the command mentioned in the first step.

3. The Arduino needs to be restarted. This is also done automatically by the command of step one.

The template consists of multiple files and folders, which are detailed in the following sections.
Communication

The communication folder contains the definition of the communication protocol between the groundstation and the Arduino Uno. For the template, it contains no telecommands, two telemetry packages, as well as a definition of the telemetry header and LogLevel data type which is used in the LOG telemetry package.

The telemetry header is prepended to all telemetry packages and contains two synchronization bytes to find the start of a package, a two byte checksum containing a CRC16 checksum of the rest of the telemetry package, and a type indicating the type of telemetry. The size of the type indicator depends on the amount of telemetry packages defined. Because the template defines less than 256 telemetry types, it uses only one byte. The LogLevel is a one byte value that can have only one of the following values: LOG_DEBUG, indicating that the message is a debug message, LOG_INFO, indicating a general message or LOG_ERROR, indicating that the message is an error message.

The telemetry content packages are defined as follows:

• A HEARTBEAT package that contains an unsigned 32 bit time in milliseconds.

• A LOG package, which contains the time the log was generated as an unsigned 32 bit time in milliseconds, one byte indicating the severity level of the message (a LogLevel as defined before), a unsigned 16 bit number indicating the number of bytes in the message, followed by that number of bytes containing a message as text.

The full telemetry package for a LOG package looks as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Size (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sync byte 1</td>
<td>uint8_t</td>
<td>1</td>
<td>The first synchronisation byte.</td>
</tr>
<tr>
<td>sync byte 2</td>
<td>uint8_t</td>
<td>1</td>
<td>The second synchronisation byte.</td>
</tr>
<tr>
<td>checksum</td>
<td>uint16_t</td>
<td>2</td>
<td>The checksum of the message. The two sync bytes and the checksum are not included in the checksum, but the telemetry data is included.</td>
</tr>
<tr>
<td>type</td>
<td>TelemetryType</td>
<td>1</td>
<td>The type of telemetry that is send in this message.</td>
</tr>
<tr>
<td>time</td>
<td>uint32_t (ms)</td>
<td>4</td>
<td>The CPU time this message was generated.</td>
</tr>
<tr>
<td>level</td>
<td>LogLevel</td>
<td>1</td>
<td>The severity of this message.</td>
</tr>
<tr>
<td>size</td>
<td>uint16_t</td>
<td>2</td>
<td>The size of the message in bytes.</td>
</tr>
<tr>
<td>message</td>
<td>char</td>
<td>1 · size</td>
<td>The message text.</td>
</tr>
</tbody>
</table>

Table 1: Layout of a LOG telemetry package as defined in the template.

Source

The source code of the template is stored in the include and src directories. Both directories contain a generated directory, which contains code generated by the ECom library that handles the creation of the telemetry packages. The include directory contains header files, which are descriptions of code so it can be used in other parts of the code. The src
folder contains the actual code. The log.cpp file contains a helper for the logging telemetry message. It provides a way to enable or disable sending debug messages and provides facilities to send formatted messages, which are messages that are constructed during runtime based on some data. This provides a similar functionality to the printf function \cite{14} that can be used for non-embedded executables to print to the terminal. It should be noted that using the formatting comes with a cost of flash memory usage. Using one of the formatting functions to format a message with an integer uses an additional 5.1 percent of the available flash memory. Formatting a floating point value requires an additional 6.7 percent of the available flash memory.

The main.cpp file contains the startup and main loop of the program. It is a modified version of the standard Arduino software template, which consists of a setup and loop function. The setup function is executed once when the Arduino starts, and the loop function is executed repeatedly afterwards. In the setup function, sending of debugging messages is enabled and the output serial of the Arduino is initialized with a baud rate of 19200. This is the communication line with the ground station. After that, the system is able to communicate with the ground station and sends two LOG messages. When adapting the template for a real project, initialization code should be added to this function.

In the loop function, the current processor time is measured and then send as a HEARTBEAT telemetry package. Afterwards, the processor is halted until the time processor time reaches the time at the start of the loop plus an offset of one second. This way, the main loop runs very stable with a frequency of 1 Hertz, independent of the amount of time actually spent inside of the loop body. When adapting the template for a real project, reading of sensors and sending of telemetry and all other code that needs to run repeatedly should be added here.

There are two more functions in the main.cpp file which are added onto the standard Arduino template. The first function is a small helper method called delayUntil, which helps to halt the processor until a given time has been reached. The other function is called writeTelemetry and this function is called by the automatically generated communication code with a byte buffer and size that should be send. All outgoing communication passes through this function, which allows to easily adapt the transportation method. In the template, the function contains to code to send the telemetry through the UART port of the Arduino Uno, which ends up sending through an attached USB cable. When adapting the template for a real project, this function can be adjusted for the specific communication requirements of the project. The SD-Card example \cite{Section 3.4.1} demonstrates how to both send the telemetry via UART and simultaneously also write it to a log file on the SD-Card. The LoRa example \cite{Section 3.4.4} shows how to use a different medium and peripheral for sending the telemetry, instead of the serial connection via UART.

In addition to the include and src folder, there is also a lib folder. This can be used for dependencies that don’t exist as a PlatformIO library. Similar to the libraries provided by Arduino, PlatformIO has an ecosystem of libraries that can be automatically downloaded, by specifying a dependency on them in the platformio.ini file. This is the preferred method of including a dependency, but it is only possible if the library is available as a PlatformIO library. If that is not the case, the library code must be downloaded manually and placed into the lib folder. It can then be used from the code in the include and src folders.
Other
In addition to the above, there are certain files and folder that describe the project or are generated during building of the project.

The README.md contains a description of the template and documentation on how to get started and on how to use it. It describes how to build and flash the project, and includes the necessary commands.

The platformio.ini file is required to make the project work with the PlatformIO framework. The file describes that the project is targeting an Arduino Uno, and can also optionally list any additional libraries that PlatformIO will download to build the project.

The test directory can be used to write PlatformIO tests to evaluate and test the code of the project. The template does not include any tests.

The .gitignore file lists files that should not be uploaded to the project GitHub page. During building of the project, a directory named .pio and cmake-build-uno can get created. These store temporary files used during compilation and the final compiled binary that is flashed on the Arduino Uno.

All files in the template excluding generated files are text files which can be opened with a simple text editor. It is however recommended to use an Integrated Development Environment (IDE) for editing the source code in the template. The Arduino IDE can not be used, because it does not support structuring the code over multiple files, which is incompatible to the source code generation technique used by the ECom library. Because the template uses the PlatformIO framework for building and flashing, the PlatformIO IDE can be used for code editing. There project also contains project files that can be read by Clion with the PlatformIO extension.

3.4 Current Modules
The UnoSat platform contains some modules that are ready to use and can be downloaded on the UnoSat project page in the examples folder. They showcase how the template was used and modified to quickly built some functionality which is commonly required of most satellite systems, like local data storage using an SD-Card and reading and transmitting sensor data.

3.4.1 SD-Card
This example shows how to use the Seeed Studio SD-Card shield V4 with the UnoSat platform. The example is able to initialize the SD-Card while running (so it is possible to remove and re-insert the SD-Card without restarting the Arduino, although that is not recommended), read some information from it and write all telemetry to a log file on it. The log is stored in a log folder on the SD-Card. When restarting, a new log file will be created.
Pin usage
The SD-Card shield uses the following pins:

- D4 (SD_CS)
- D11 (SD_DI)
- D12 (SD_DO)
- D13 (SD_CLK)

Reading of the log files
The log files on the SD-Card are stored in the same format as if they were sent as regular telemetry, therefore they can not be read as simple text with a text editor. Because the telemetry is optimized for size and allows to detect invalid messages, the same applies to the saved files. They are as small as possible and if the SD-Card were to get corrupted, one could detect that while parsing the file.
To parse the log files, the ECom library contains a small tool called *logConverter* that can convert the saved file into a csv file. The tool can be downloaded and its documentation can be read here.

### 3.4.2 RTK GPS

This example shows how to get centimeter precision RTK GPS location using an Ardusimple simpleRTK2B module.

The sensor Arduino (middle in Figure 13) is responsible for reading and decoding the GPS position from the simpleRTK2B module (left in Figure 13) and sending it to the base Arduino (right in Figure 13). The base Arduino forwards the GPS position and any log from the sensor Arduino via the USB cable to a connected device.

The reason for using two Arduino Uno microcontrollers is because of the very limited memory resources available on the Arduino Uno. The library used to parse the location messages from the RTK module is very large, and uses more than 96% of the Arduino Uno’s available Flash memory.

Note that the centimeter precision positioning is only available with an RTK base station, which is a groundstation that the Ardusimple simpleRTK2B module communicate with in order to get GPS corrections and improve the positioning accuracy. Without the base station, the setup will provide GPS position with regular precision.

![Figure 13: Uno Sat example with RTK GPS module.](image)

In this setup, the base Arduino is on a different power supply than the sensor Arduino and the simpleRTK2B module. Both Arduino microcontrollers receive power either via the power jack or USB cable. The base Arduino also uses the USB cable to send telemetry to
a connected device. The Arduino systems must share a common ground, so a ground pin of
one Arduino must be connected to a ground pin of the other Arduino. (In the image, they
are connected via a breadboard.)

The communication line with the sensor Arduino needs to be connected as follows: The RX
pin (pin 7) on the Arduino is connected to the TX pin (pin 3) on the sensor Arduino. The TX
pin (pin 8) on the Arduino is connected to the RX pin (pin 2) on the sensor Arduino.

In this setup, the system is connected to power via the Arduino Uno, either by USB or
power jack connection. One of the ground pins of the Arduino Uno has to be connected with
one of the ground pins on the simpleRTK2B. The 5V pin of the Arduino has to be connected
to the 5V_IN port of the simpleRTK2B, as well as the IOREF pin, because the Arduino Uno
uses 0 - 5 volts on it’s IO ports.

The communication line with the RTK module needs to be connected to UART1 (pin 0
and pin 1), the only hardware UART port on the Arduino Uno, because it is the only one
that can operate at the required baud rate. The RX pin (pin 0) on the Arduino is connected
to the TX1 pin on the simpleRTK2B. The TX pin (pin 1) on the Arduino is connected to
the RX1 pin on the simpleRTK2B.

3.4.3 Grove simple kit

This example demonstrates how to use the UnoSat platform with two Grove Beginner Kit for
Arduino and two Grove LoRa Radio wireless 868 MHz communication modules to collect
data on one of the Grove kits and send it to the second Grove Kit and display it there. This
demonstrates how the UnoSat system can be used to establish communication between two
devices, where the sending Grove kit is collecting data and acts as the balloon side, and the
receiving Grove kit acts as the ground station displaying the data it receives from the other
kit.

The Grove Beginner Kit for Arduino consists of an Arduino Uno compatible board and
10 additional Arduino sensors and actuators. These include a Grove LED, buzzer, OLED
display, button, rotary potentiometer, sound sensor, light sensor, temperature and humidity
sensor, 3-axis Accelerometer, and an air pressure sensor. The Arduino board and the sensors
are all integrated on the PCB and don’t require any cables to connect. However, it is possible
to break out individual sensors or the Arduino PCB and connect the with cables. It is also
possible to add other sensors. The board doesn’t fit into the form factor of a CubeSat, but it
is itself, with the additions of a few more sensors, a fully functional balloon payload. Together
with the code examples online, it provides a very easy way of building a balloon project.

The first Grove kit collects temperature, humidity and pressure data. The data is then
send using the LoRa module to the second kit and also display on the LCD screen of the
first kit. The second Grove kit does not use any of its own sensors. Instead, it just wait for
incoming messages via the LoRa module and then displays the contents on its LCD screen.
The LoRa modules are connected to the UART1 port of the Grove Kits.

3.4.4 LoRa

This example shows the basic template example with a LoRa 433 MHz Radio shield \[18\]. It demonstrates full wireless communication between a groundstation running on a device via a proxy Arduino Uno to a base Arduino Uno.
Pin usage
The LoRa shield uses the following pins:

- D2 (RFM95_INT)
- D9 (RFM9x_RS)
- D10 (RFM9x_CS)
- D11 (RFM9x_MOSI)
- D12 (RFM9x_MISO)
- D13 (RFM9x_SCK)

The pins overlap with the pins used by the SD-Card module in Section 3.4.1 which means that both modules can't be used together as is. However, this should not be a problem because the overlap is caused by both modules using the same SPI bus. Using the chip select pin to select at runtime which devices (SD-Card or LoRa) is active and switching between the two,
both shields should be able to be used together on the same Arduino Uno. This has not been
tested and will require slight modifications of the code to add the chip select pin handling.

4 Tests

To validate the UnoSat system, the functionality of all examples has been tested. In addition,
the following tests have been performed:

The Grove simple kit example (see Section 3.4.3) has been tested in a vacuum and cryo-
genic test. The Grove kit which uses the sensors was placed inside a vacuum chamber, and
the pressure was reduced to about 1 millibar. Then, while still in a vacuum, the chamber was
cooled down using liquid nitrogen. This verified the use of the Grove kit, its sensors and the
Lora module. It also verified that the basic UnoSat template is able to operate in conditions
that match those on the stratospheric balloon. The actual data from the tests is not included
here in this report, because these tests have also been done by the students of previous years
and the hardware has already been flown.

Another test was the demonstration that two receiver Arduinos and groundstations could
simultaneously receive the telemetry from one sender Arduino with the Lora shield. For
the test, the sender Arduino was programmed with the code from Section 3.4.4 and two
Arduino Unos were programmed with the receiver proxy code. Both receiver proxy Arduinos
were connected to two different computers each running the simple groundstation example
provided by the ECom library. They were also powered by the connected computers, while
the sender Arduino was powered using a battery. When the sender Arduino was powered
on, both groundstations received the same telemetry, proving communication with multiple
groundstations.

5 Public Outreach

In order to document and give new students access to usages of the UnoSat system on previous
balloon projects, a new website was created. Each year, the students have to write a report
at the end of the balloon project. This report is used by students of the following years to
learn about previous missions and to look for documentation when reusing previous projects.
The new website helps students to discover these old reports and to increase public outreach
for the balloon projects. The students are allowed to add a new page for their balloon project
to the web site through an online editor. On that page, they can present their project and
upload any documentation about it. This opportunity was also used to overhaul the current
website of the TSI M2 master in Toulouse.

The new website was created in a Docker container using Wagtail, which is based on the
Python framework Django. Wagtail as a Content Management System was chosen, because
it brings a lot of features already out of the box, like user and group management, page
permissions, a page editor with a preview and the ability to define custom pages. Wagtail
is also supported by Google and used by NASA for their website, so it hopefully should be
supported and under active development for the foreseeable future. During implementation
of the balloon website and re-implementation of the old master site, special care was taken
to make the website also work on mobile browsers with smaller screen sizes, which wasn’t
accounted for in the old masters website.
At the time of writing, the website has been finalized, and it is expected to replace the old website eventually at https://m2tsi.eu.

Figure 16: The front page of the new website.

Figure 17: The front page of the old website.

Figure 18: The front page of the new website on a smaller screen.
One of the main benefits of Wagtail is that most modifications of the website does not require any changes in the code, so the editor does not have to know any programming to change the site. This works by using templates. Templates are skeletons of web pages, without content. The content is given by the editor using the web editor interface shown in Figure 19. A model defines what can be configured for a specific template, and the data is then filled into the template. Only changes to the model or to the template itself require changes to the code.

6 Conclusions and perspectives

The students of the M2 TSI 2022/2023 are tasked to develop balloon to balloon communication with their balloon projects. The UnoSat platform has already demonstrated Arduino to Arduino communication, and is also able to communicate both ways. Communication with the LoRa 433 MHz Radio shield was also already demonstrated, so the students should only need to combine the examples to get a system that is able to communicate wireless both ways between balloons and a groundstation. Additionally, the 433 MHz frequency communication can also be received in Toulouse using the Yagi antenna at CSUT, using the groundstation lead by Nicolas Nolhier, as demonstrated by the balloon project in 2021/2022.

The UnoSat platform was presented to the M2 TSI 2022/2023 students. After the presentation, the system was installed on one of the students laptop and he was guided until the basic template was running on an Arduino Uno and he received telemetry. This was an introduction for the new students into the UnoSat system and will help the to adopt it in their new balloon projects. Usage of the UnoSat system is a requirement for them, so the system will at least be used in the next year, and if it proves useful, the years after.

In the future, the creation of the 3D models could be moved to CadQuery [19], which would even allow a procedural approach, programmable in Python, and a quick adaption for new parts or changes in the form factor.

In addition, work is currently ongoing to integrate the UnoSat communication system into the groundstation from 2021/2022. When this is done, the groundstation can be reused for every project using the UnoSat communication system. It could also be possible to allow editing of the communication protocol from the groundstation.

In conclusion, the UnoSat platform is giving the students a running start for their balloon projects, by providing tested and improved solutions to common requirements out of the box. The students will have more time to focus on their actual goal and don’t have to spend much time on the supporting architecture.
References